J4CS: An Early-Stage Statement-Level Metric for Energy Consumption of Embedded SW

Vittoriano Muttillo
Center of Excellence DEWS
University of L'Aquila
L'Aquila, Italy
vittoriano.muttillo@graduate.univaq.it

***Abstract*—This work presents an early-stage statement-level metric for energy consumption of embedded SW. In particular, based on an existing assembly-level analysis and some profiling activities performed on a given C benchmark, it defines a metric related to the average energy consumption of a generic C statement for a given target processor. Such a metric, evaluated with a one-time effort, can be then used to rapidly estimate the energy consumption of a given C function for all the characterized processors. Two reference embedded processors are then considered in order to show an example of usage of the proposed metric.**

***Index Terms*—Embedded SW, Energy Consumption, Profiling, Benchmarking, Metrics**

I. Introduction

Energy consumption is one of the most critical design issues in the embedded systems domain. In fact, the need to guarantee an even longer life for all battery-powered devices is one of the main problems that affect the design activities. In particular, especially at the system-level of abstraction, the choices made by designers can drastically influence the final system energy consumption, since different optimizations can be considered in the whole *Electronic System Level* (ESL) design flow. For this, different energy consumption models can be taken into account to estimate the energy consumption of the final system implementation. Such models can be related to processors, *Application Specific Integrated Circuit* (ASIC), memories, and the interconnections among them. Moreover, the models can be at different levels of abstraction and granularity, mainly depending on the required estimation accuracy. Since this work focuses on embedded processors, Figure 1 shows the typical abstraction levels involved in a classical ESL design flow for embedded processors development [1]. The first abstraction level, called *Functional*, also catches very few non-functional static processor features, as average *Clock cycles Per Instruction* (CPI), static power dissipation, etc. The *Architectural/ISS* abstraction level involves the knowledge of the *Instruction Set Architecture* (ISA) and it is normally supported by a so-called *Instruction Set Simulator* (ISS) to perform several kinds of dynamic analysis. The *Pipeline-accurate Architectural/ISS* abstraction level adds details about the pipelines behaviour to the simulator, so considering a more refined processor model. Finally, the *Cycle-accurate Micro-Architectural* abstraction level introduces further details about the processor architecture in terms of *Control Unit*

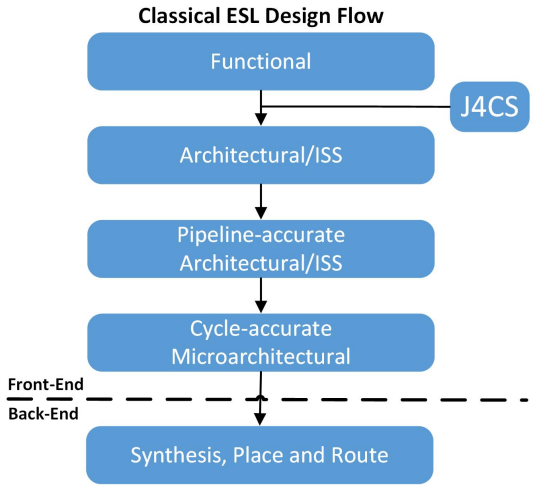


Fig. 1. Classical ESL design flow for embedded processors.

and *Data Path* allowing a cycle-accurate analysis of the final implementation. The work presented in this paper is located between the first two levels of the design flow, since it proposes a *statement-level* metric, called J4CS (*Joule for C Statement*), to measure the average energy consumption associated to the execution of a generic C statement by means of a given target processor. However, while considering J4CS, two main issues have to be discussed. The first one is related to the definition of "generic C statement". This work adopts the same approach presented in [3], where it has been defined by adopting an empirical approach: it refers to the way a common profiling tool as *gcov* [2] performs C statements identification and counting when profiling their execution. The second issue is related to the fact that J4CS is influenced also by the used C compiler. Some possible ways to manage such an influence are to specify also the used one (possibly giving rise to a J4CS for each processor/compiler pairs), or to consider the average of the results obtained by using the most diffused C compilers. In any case, the issue can be managed by means of a statistical characterization of J4CS, i.e., by evaluating a set of values related to *Min*, *Max*, *Average*, *Standard Deviation* and by trying to identify an associated statistical distribution. Such a characterization is then performed basing on the assembly-level analysis presented in [4], as explained in Section III, and

by exploiting the framework developed in [3]. Obtained J4CS can be so assigned to each statement of a C function and used, by means of an host-based source-level profiling, to estimate the total amount of energy consumed when providing specific inputs.

The remainder of the work is organized as follows: Section II describes the works related to the power/energy consumption estimation and evaluation problem. Section III formally defines the J4CS metric proposed in this work. Then, Section IV presents the J4CS metric evaluation procedure for two reference embedded processors, while Section V shows how it is possible to use the obtained values to estimate and compare the energy consumption associated to the execution of a given C function. Finally, Section VI closes the paper with some conclusions and future works description.

II. Related Works

Starting from lower-levels of abstraction, such as gate or Register-Transfer (RT) ones [7], a lot of works consider the problem of estimate power/energy consumption by using very time-consuming simulators [8]. Other works start from an accurate modeling of the target ISS [5] [9], but this still requires a considerable time for both the modeling and simulation activities. Some works try to increase the abstraction level, by going towards the system level one. This is often done by directly considering source-code [10], but also this kind of analysis could involve different time-consuming activities strictly related to the need of taking into account the peculiarities of the considered target processors. The same problems arise in approaches that involve the introduction of some kind of *Virtual Instruction Set* (e.g., [11]), but that still requires some explicit detailed knowledge of the final processors architecture. With respect to the source-code analysis, the work in [12] presents a statement-level timing estimation (and related energy consumption estimation that depends on timing), by evaluating the power/energy metrics directly on the base of timing and profiling activities. A work that tries to bridge the gap between the reduced simulation time of high-level dynamic analysis of code with the accuracy of low-level dynamic analysis is [13], that introduces an intermediate pseudo instruction set for analyzing application and HW architectures, using an approach still similar to [11]. Finally, a statement-level energy estimation based on GCC has been proposed in [14] and [15].

In such a context, the work presented in this paper is close to the work in [4], but its purpose is to reduce the time of estimation activities by means of a strategy that allows to quickly evaluate and select processors in an early-stage analysis. In fact, the estimation of the energy consumed during SW execution is very fast since it is based only on a host-based source-level profiling. More detailed ISA-related analysis, if needed, can be then performed by focusing only on the selected processors.

III. Metric Definition

The power consumption of a microprocessor [9] during the execution of a given program can be evaluated as:

$$P_{tot} = P_{dyn} + P_{stat} = C_L \cdot V_{dd}^2 \cdot f + V_{dd} \cdot I_{leak} \quad (1)$$

where P_{tot} is the total power consumption made up of dynamic and static power contributions, C_L is the average switched capacitance per clock cycle during the execution of the program, V_{dd} is the supply voltage, $\boldsymbol{f}$ is the clock frequency, and I_{leak} is the current leakage, i.e., the current that flows through the circuit to ground. Considering the execution time associated to the given SW program, it is possible to evaluate the total energy consumption as:

$$E_{tot} = P_{tot} \cdot t = C_{tot} \cdot V_{dd}^2 + V_{dd} \cdot I_{leak} \cdot t \quad (2)$$

where C_{tot} is the total switched capacitance. Changing the clock frequency (and so decreasing/increasing the program execution time) doesn't change C_{tot} [9] and so the energy consumption decrease/increase linearly with the scaled frequency with the slope proportional to the amount of leakage.

Considering the average power consumed by a microprocessor while running a program, it is possible to simplify the Equation 2 by considering $\bar{P} = I \times V_{dd}$, where I is the average current and V_{dd} the voltage supply. So, the average energy consumed by a program can be expressed by: $\bar{E} = P \times N \times \tau$, where N is the number of program clock cycles and τ is the clock period [6].

Thus, while taking into account the formulas described above, the method proposed in this work exploits some benchmark activities on a specific set of C functions to evaluate a metric related to the average energy consumption per C statement, as described below, to estimate a statistical interval of energy consumption.

A. Definition of J4CS

As said in [4], many embedded microprocessors have a statistical property of constant energy consumption for each executed assembly instruction. So, the proposed idea is to apply the same approach to a higher abstraction level (i.e., statement-level) by characterizing the energy cost (e.g., minimum, maximum, average) associated to the execution of a C statement. This is done by performing several simulations to consider a lot of execution paths depending on the inputs.

In order to perform statistical analysis, some assumptions must be made:

- if the program has a huge amount of lines of code (LOC), then the energy consumed for each instruction can be considered constant without great loss of accuracy [9];
- the entire statements set is considered homogeneous between C operators and variables (as the analysis is given from a statistical point of view, each statement contributes in the same way for the evaluation of the total energy consumption);
- an average number of assembly instructions per C statement is considered;

- all the assumptions made in [4] respect to pipeline stages, number of clock cycles and so on must be considered as valid too.

Under these assumptions, it is possible to define the average energy of a machine instruction [4] as $\bar{E} = \frac{\bar{P}}{\phi \cdot f}$, where $\bar{P}$ is the mean power consumption of a microprocessor, $\boldsymbol{f}$ is the frequency and ϕ is the processor power efficiency (related to the MIPS parameter, normally provided on processors datasheets [16]). Starting from this equation, using a profiling step in order to find the total number of assembly instructions executed (N) and the total number of C statements executed (M), by using gcov [3], it is possible to define a source-level energy consumption metric as presented below.

Definition III.1. *J4CS (Joule for C Statements).* Considering a single C function, J4CS is the ratio between the number of assembly instructions executed by the target processor executing the function and the number of executed C statements multiplied by the average energy of a machine instruction execution, i.e.: the J4CS metric is the average energy consumption associated to a C statements executed on a specific processor, and it is defined as:

$$J4CS = \frac{N \times \bar{E}}{M} \tag{3}$$

IV. Evaluation of J4CS

A. General Framework

To evaluate the metric for a given processor it is needed, at least, to: define a set of relevant C functions to be used as benchmark [3]; for each function belonging to the benchmark, to identify a way to stimulate (i.e., to execute) it by means of relevant input data sets; to identify a tool to perform source-level profiling in order to count the number of executed C statements for each input; to identify tools to compile the C function for the target processor and to simulate its execution in order to obtain total number of executed assembly instructions. Naturally, such steps must be applied for each different processors that have to be characterized. However, it is worth noting that it is an one-time effort since J4CS, once evaluated, is available "for free" for any successive estimation activities. So, to support J4CS evaluation, a proper framework has been adapted. Additionally, such a framework is also able to evaluate statistics on the metric itself. The following paragraphs describe the general features of the proposed framework while processor specific features are described later.

1) Inputs Generation: To evaluate J4CS, a module that (semi)automatically generates inputs for the benchmark functions has been used. In particular, for each function they have been randomly generated 1000 input data sets. Moreover, for each function, different data types have been considered (i.e., *int8, int16, int32,* and *float*) to analyze the results with respect to the internal architecture of the considered processor. Each input data set is then stored in a header file to be included in the function at compile time.

2) Profiling on the Host Architecture: After the inputs generation phase, a tool to count the number of executed C statements is needed. This value is obtained by performing a profiling of the benchmark functions by means of the *gcov* [2] profiler for each generated input. The total number of executed C statements for each function is simply the sum of the single profiling numbers associated to each statements. It is worth noting that such a profiling is performed one-time on the host platform since it is independent of the target processor.

3) Profiling on the Target Processor: The last data needed to calculate the J4CS metric is the number of assembly instructions executed by the target processor for each function and input set in the benchmark. So, for each target processor there is the need for an *Instruction Set Simulator* (ISS) (Fig. 2).

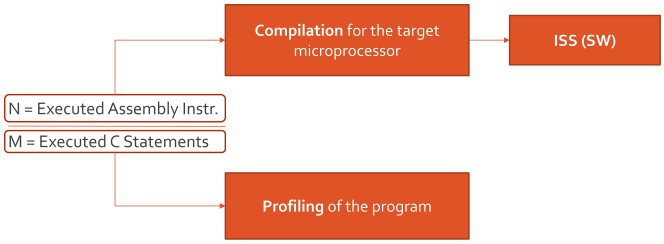


Fig. 2. J4CS Evaluation Framework.

B. Processor Specific Framework: Two Examples

J4CS has been evaluated by considering some specific processors. In this work two processors have been analyzed [3]: the LEON3, a 32-bit synthesizable soft-processor compatible with SPARC V8 architecture [18], and the ATmega328/P [19], a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. The execution has been performed with a software simulation of the processor by using the *Instruction Set Simulators* (ISS) GAISLER TSIM [17] and SimulAVR [20].

In order to evaluate results in a real scenario, two boards have been considered to take voltage and frequency information (the power information can be found in the processor datasheets): LEON3FT-RTAX [21] and Arduino Uno [22]. The processors parameters used to evaluate the metric [18] are shown in Table I;

TABLE I
Board characteristics.

Parameters	LEON3FT-RTAX	Arduino UNO
Clock	25 MHz	16 MHz
V_{dd}	3,3 V	5,0 V
$\bar{P}$	500 mW	60 mW
MIPS	20	16
ϕ	0,025	0,0037

It is worth noting that LEON3-FT is a System-On-Chip design based on LEON3FT core, and it has the same ISA of the classical LEON3 processor. Therefore, the number of assembly instructions executed by the ISS is the same

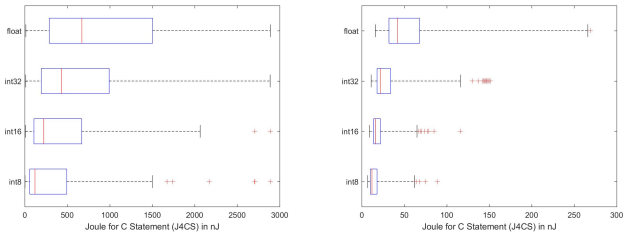
for both processors since they rely on the same compiler. So, considering these characteristics, the average energy consumption associated to each executed assembly instruction is: ($\bar{E}_{RTAX} = 0,8\ nJ/Instr.$ and $\bar{E}_{ATMEGA328/P} = 1\ nJ/Instr.$). The obtained results for the executions of the benchmark functions are summarized in Table II.

TABLE II
J4CS MEASURED ON LEON3FT-RTAX AND ARDUINO UNO (IN NJ)

Data Type	Min	Median	Max	AM[1]	SD[2]	Var[3]	GM[4]	85%[5]	95%[6]
LEON3FT-RTAX int8	1	36	869	100,73	137,03	18779	48,33	220	338
LEON3FT-RTAX int16	1	67	868	140,66	173,38	30061	75,33	312	523
LEON3FT-RTAX int32	4	129	868	213,11	208,60	43513	131,98	451	814
LEON3FT-RTAX float	4	202	869	270,45	240,16	57676	173,18	597	814
LEON3FT-RTAX AVG	5	108,5	869	181,24	189,79	37507	107,20	348,25	622,25
Arduino UNO int8	7	12	89	14,43	7,45	55,63	13,19	20	30
Arduino UNO int16	9	16	116	18,57	8,89	79,12	17,06	25	38
Arduino UNO int32	11	21	151	33,87	32,45	1053,4	26,28	48	146
Arduino UNO float	16	41	269	61,44	52,06	2710,6	48,34	78	204
Arduino UNO AVG	10,75	22,5	156	32,08	25,21	974,7	26,22	42,75	104,5

[1]AM: Arithmetic Mean; [2]SD: Standard Deviation; [3]Var: Variance;
[4]GM: Geometric Mean; [5]85%: 85^{th} Percentile; [6]95%: 95^{th} Percentile;

For each function, different data types have been considered (*int8, int16, int32,* and *float*). In fact, both timing [3] and energy, especially the average ones, change with respect to the dimension of data. Fig. 3 shows the distribution related to J4CS evaluated for LEON3 and ATMega ISS, with respect to the reference benchmark. The described evaluation process of J4CS for the two processors has required a total of near 10 non-consecutive hours on a standard workstation (Intel i7, 1.5 GHz, 16 GB RAM). However, as highlighted before, this is a one-time effort to make available J4CS for subsequent analysis (as shown in the next section).



(a) J4CS for LEON3FT-RTAX. (b) J4CS for Arduino UNO.

Fig. 3. J4CS BoxPlot results.

V. J4CS-BASED ENERGY CONSUMPTION ESTIMATIONS

The availability of J4CS is very useful for very fast early-stage estimation, comparison, and selection. In fact, by having available J4CS for different processors, with a single host-based profiling it is possible to estimate the energy consumption of a function of interest for the whole processors set. As an example, given a target function *tf()* and considering a specific *golden input* $\boldsymbol{x}$, by means of a host-based profiling (that take less than a second on the same workstation described in the previous section) it is possible to count the number of executed C statements during the execution of tf($\mathbf{x}$) (e.g., 100). Then, as shown in Fig. 4 (the x-axis is in a logarithmic scale), it is straightforward to compare the whole processors set by multiplying 100 for the related J4CS. Depending on a possible energy consumption constraint it is then possible to select a specific processors or, at least, to reduce the set to the ones to be considered for further analyses.

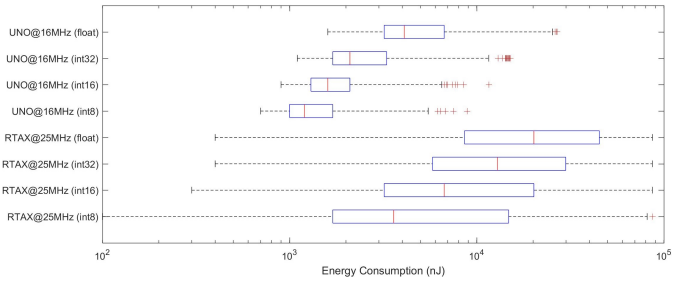


Fig. 4. J4CS-based SW comparison.

VI. CONCLUSION AND FUTURE WORK

This work has presented a metric useful to estimate in an early-stage design phase the energy consumption related to the execution of embedded SW on a target processor. This metric is good for very fast estimation, comparison and selection activities. Then, more accurate approaches at lower abstraction levels can be used for more precise and time-consuming estimations. Other than in the pure SW domain, this metric can be easily exploited into specific HW/SW Co-Design methodologies and tools (e.g., [24]), in order to consider energy requirements during system-level design space exploration. In fact, it is worth noting that this metric can be evaluated also in the HW domain, by using *High-Level Synthesis* (HLS) tools and *HDL simulators* able to provide energy information as output. Such values can be used to substitute the $N \times \bar{E}$ numerator value in Definition III.1. Moreover, J4CS can be also useful in ESL energy consumption estimation approaches that rely on the availability of an estimated energy consumption for each statement composing the ESL specification (e.g., [25]). As future work, some interesting opportunities, still at early-stage, are related to the use of HW profilers [26] to evaluate estimation errors directly on-target, and to the combined exploitation of the *Affinity* metric [23] to reduce such errors by identifying a proper distribution subset, and to the exploitation of a more detailed static analysis of source-code in order to assign different weights to different statements.

ACKNOWLEDGMENT

This work has been partially supported by the ECSEL RIA 2017 FitOptiVis project.

REFERENCES

[1] Y. H. Park, S. Pasricha, F. J. Kurdahi and N. Dutt, "A Multi-Granularity Power Modeling Methodology for Embedded Processors," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 4, pp. 668-681, April 2011.
[2] GCov Profiler, https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[3] V. Muttillo, G. Valente, L. Pomante, V. Stoico, F. DAntonio, and F. Salice, CC4CS: an Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies, In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18), ACM, New York, NY, USA, 2018, pp. 119-122.
[4] J. Castillo, H. Posadas, E. Villar, M. Martinez, Energy consumption estimation technique in embedded processors with stable power consumption based on source-code operator energy figures, In Proc. DCIS, 2007.
[5] V. Tiwari, S. Malik and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 4, pp. 437-445, Dec. 1994.
[6] V. Tiwari, S. Malik, A. Wolfe, Instruction level power analysis and optimization of software, In Proc. of VLSI Design, IEEE, 1996, pp. 326-328.
[7] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 4, pp. 446-455, Dec. 1994.
[8] T. Simunic, L. Benini and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361), New Orleans, LA, 1999, pp. 867-872.
[9] A. Sinha and A. P. Chandrakasan, "JouleTrack-a Web based tool for software energy profiling," Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), 2001, pp. 220-225.
[10] Eric Senn, Nathalie Julien, Johann Laurent, and Eric Martin. 2002. Power Consumption Estimation of a C Program for Data-Intensive Applications. In Proceedings of the 12th International Workshop on Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation (PATMOS '02), Bertrand Hochet, Antonio J. Acosta, and Manuel J. Bellido (Eds.). Springer-Verlag, London, UK, UK, 332-341.
[11] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice and D. Sciuto, "A multi-level strategy for software power estimation," Proceedings 13th International Symposium on System Synthesis, Madrid, 2000, pp. 187-192.
[12] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, Source-level execution time estimation of C programs, CODES, 2001, pp. 98-103.
[13] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, Timing and Energy Estimation of C Programs, Digital System Design Architectures, Methods and Tools, 2008, pp. 115 123.
[14] L. Bogdanov, "Statement-level energy simulation in embedded systems using GCC," 2016 XXV International Scientific Conference Electronics (ET), Sozopol, 2016, pp. 1-4.
[15] L. Bogdanov. Look-up table-based microprocessor energy model, TECHSYS 2016 Proc., ISSN 2367-8577, p. II-180 185, TU-Sofia, 2016.
[16] S. Furber, "ARM System-on-Chip Architecture (2nd Edition)", Addison-Wesley, 2000.
[17] TSIM2 ERC32/LEON simulator, https://www.gaisler.com/.
[18] LEON3 processor, https://www.gaisler.com/.
[19] ATMega328/P, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.
[20] SimulAVR, an AVR Simulator, http://savannah.nongnu.org/projects/simulavr.
[21] LEON3-FT SPARC V8 Processor LEON3FT-RTAX, https://www.gaisler.com/doc/leon3ft-rtax-ag.pdf.
[22] Arduino Uno, https://store.arduino.cc/arduino-uno-rev3.
[23] L. Pomante, D. Sciuto, F. Salice, W. Fornaciari, and C. Brandolese. Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC. In IEEE Transactions on Computers, 55(5):508-519, 2006
[24] V. Muttillo, G. Valente, D. Ciambrone, V. Stoico, and L. Pomante. HEPSYCODE-RT: a Real-Time Extension for an ESL HW/SW Co-Design Methodology. Proceedings of the Rapido'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '18). ACM, New York, NY, USA, Article 6, 6 pages.
[25] Berardinelli L., Di Marco A., Pace S., Pomante L., Tiberti W. (2015) Energy Consumption Analysis and Design of Energy-Aware WSN Agents in fUML. In: Taentzer G., Bordeleau F. (eds) Modelling Foundations and Applications. ECMFA 2015. Lecture Notes in Computer Science, vol 9153. Springer, Cham
[26] A. Moro, F. Federici, G. Valente, L. Pomante, M. Faccio and V. Muttillo, "Hardware performance sniffers for embedded systems profiling," 2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES), Ancona, 2015, pp. 29-34.