Accurate Power Consumption Evaluation for Peripherals in Ultra Low-Power embedded systems

Gautier Berthou, Kevin Marquet, Tanguy Risset, Guillaume Salagnac

Accurate Power Consumption Evaluation for Peripherals in Ultra Low-Power embedded systems

Gautier Berthou, Kevin Marquet, Tanguy Risset, Guillaume Salagnac
Univ Lyon, INSA Lyon, Inria, CITI,
F-69621, Villeurbanne, France
Email: firstname.lastname@insa-lyon.fr

***Abstract*—We propose a methodology to measure, model and simulate power consumption of peripheral devices of a low-power embedded micro-controller, while keeping a reasonable development cost. This methodology is experimented against the low-power MSP-EXP430FR5739 platform that includes non-volatile RAM for intermittent computing purposes and a handful of peripherals. The experimental measurements enable the characterization of the consumption of the peripherals, while many existing comparable studies do not provide power consumption for peripherals. These measurements are integrated into a simulator that targets low-power peripheral-intensive applications, as are most of IoT embedded programs. The accuracy of the power consumption estimation is within a 5% error on intermittent embedded computing using peripherals.**

***Index Terms*—Embedded systems, Simulation and emulation, Power estimation and optimization, Energy model, Energy measurement**

I. Introduction

In ultra low-power embedded systems, energy consumption estimation is very important for battery lifetime analysis and power management in general. The presence of increasingly small and low-power communicating systems motivates the need for an accurate estimation of their power consumption.

In these systems, peripheral devices such as light-emitting diodes (LEDs), timers, analog-to-digital converters (ADCs), serial buses, radio transceivers or various sensors often consume more than computational parts, such as CPU and memory. This work addresses the issue of accurately modeling and simulating the execution and energy consumption of a low-power system.

In the field of embedded programming, software access to peripherals is usually performed through a finite amount of API commands, called *driver calls*. Rather than simulating every single assembly instruction of these driver calls, we choose to measure the energy consumption and the duration of each of these calls. Indeed, a driver call usually consists of several writes to peripheral control registers, which leads to intermediate peripheral states that are out of the scope of this paper. Then, during simulation, the driver calls are identified and *abstractly* executed.

This paper includes the following contributions:

- A simplified power model of low-power embedded systems. This power model leverages an abstraction of peripherals which is built from the *driver API* and not from the peripheral data sheet, unlike what is usually done. This choice allows us to obtain precise simulations of embedded software code using peripherals at a reasonable development cost.
- A complete methodology for ultra-low power system simulation, including a low-cost experimental platform able to precisely measure the current consumption of peripheral operations and the use of an existing CPU simulator generator (ArchC [1]) to quickly obtain a cycle-accurate simulation.
- The experimentation of the whole methodology: precise energy measurement of all peripheral devices routines, simulator generation and validation of the energy consumption simulation on the low-power MSP-EXP430FR5739 [1] platform that includes non-volatile RAM for intermittent power purposes and a handful of peripherals.

The paper is organized as follows: Section II presents the state of the art on power modeling, simulation and energy measurement for ultra-low power systems. Software and peripheral devices models are presented in Section III. Section IV presents the power monitoring device and Section V presents experimental results that validate the simulator accuracy.

II. Context and state of the art

a) Ultra-low power sensors: With the increasing deployment of IoT Sensors [2], [3], power consumption of the end devices has become a crucial issue. Even though many research trends focus on wireless network optimization [4], the use of new technologies such as MEMS or embedded NVRAM is increasingly studied for device power consumption optimization as battery life is the main concern. The development of harvesting technologies [5]–[8] enables to deploy low-power sensors in transiently-powered environments while using a capacitor to store energy instead of an actual battery [9], [10]. These systems propose, by inserting system snapshots in non-volatile memory – or other remanent storage technology –, to recover the system after a power outage [11]–[13]. But none of these studies estimate peripheral power consumption.

b) Power modeling and simulation: Three main approaches can be used to evaluate the energy consumption of embedded systems: theoretical analysis, simulation and

978-1-7281-6728-2/20/$31.00 © 2020 IEEE

[1] http://www.ti.com/lit/ug/slau343b/slau343b.pdf

hardware measurement. To yield realistic results, simulation must be calibrated with real measurements. One issue is to establish a power model of all hardware components: CPU, memory and peripheral devices.

Part of this problem is similar to *power state tracking*, that has been successfully used in the past [14]–[16]. The power state of each component is tracked by manually modifying device drivers. Compared to these works, our power model holds more information than just a power state. Indeed, power, duration and energy consumption information is associated to every kernel service, including driver primitives.

Profiling power consumption on real hardware has already been studied, but either the peripheral devices were not studied [17] or the targeted platforms were high-end systems [18].

SysWCEC [19] proposes a model where the microcontroller may have several power modes while the peripherals may only have two states, on and off. It is focused on the worst case through static analysis and exhaustive path enumeration to handle all cases in a multi-task model. Another work [20] is dedicated to the automation of power model extraction for peripheral devices. It leverages only high current gaps, which are observed with peripherals such as radio chips, but does not target less consuming peripherals. However, its model provides a sound baseline for simulators such as the one we built.

c) Energy consumption measurement: Energy consumption measurement for low-power embedded devices is not an easy task. The main issues are cost, design complexity, dynamic range and accuracy. Various approaches exist to measure power [21].

Energy consumption evaluation requires precise measurements and a *model* of the platform. For ultra-low power systems, allowing intermittent execution, a variety of models has been proposed [22]–[24]. The EPIC modeling tool [25] supports temperature variation and clock drifts in power consumption estimation. These models do not address power consumption of the peripherals.

One of the only methodologies that really addresses power consumption measurement for peripherals is EMPIOT [26]. It proposes an accurate, low-cost power measurement platform and targets wireless IoT devices. The platform proposed in Section IV is similar to EMPIOT, but is more suited to the specific dynamic range and resolution needed on ultra-low power devices. Indeed, EMPIOT is able to measure up to 400 mA with 100 μA resolution, while our measurement platform measures up to 26 mA with 6 μA resolution.

Other works [18], [27] use acquisition methods that do not allow measuring with a current range as wide as required by transiently-powered platforms.

Like the works mentioned above, the general method used here to compute the energy consumed is to record the current drawn by the platform and integrate it over time. A completely different approach consists in using a coulomb counter [28]. However, this work does not use a coulomb counter because of the known imprecision in the measurement of the capacitance of a capacitor.

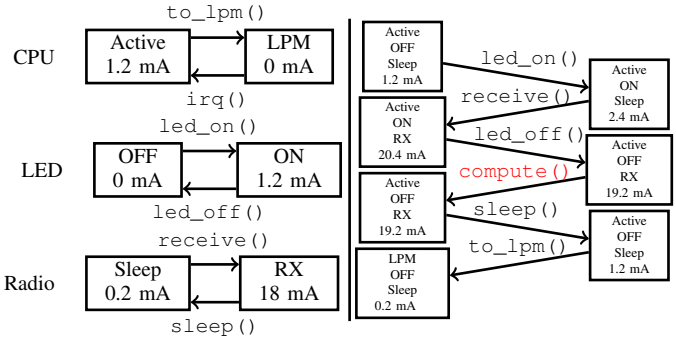

Fig. 1: Example showing how the platform's power state evolves with driver calls. On the left, the power state machines of the CPU (LPM stands for Low-Power Mode), LED and radio chip with arbitrary values. On the right, the evolution of current consumption following a particular software trace. The `compute()` call does not alter platform power state.

Another energy-related tool is PEEK [29]. It provides a physical platform for energy measurements, however it solely aims at proposing static software optimizations to improve overall application energy consumption.

III. Platform power model

This paper relies on a platform model that takes all devices into account: CPU, memory, on-chip and off-chip peripherals.

A. General model and assumptions

A software program is modeled as a control flow graph where each node consists in a sequence of either regular assembly instructions or calls to driver functions. Software code that does not call any driver function may change the application state but may not impact the platform power state. On the contrary, only driver code is allowed to modify the peripherals and thus, the platform power state. Device driver routines are modeled as atomic function calls. Fig. 1 illustrates this model, applied to a simplified example platform.

In this model, peripheral states are not expected to change on their own. Strictly speaking, asynchronous calls may allow peripherals to change without the explicit intervention of software code. Timers are examples of such peripherals, since their counter registers do not require any software intervention to update their values, but the counter evolution does not modify the timer power consumption. Our only strong assumption is that an interrupt is always raised – and hence can be caught by our model – when a peripheral gets into a state that changes its power consumption on its own.

The software model described in this section imposes that peripheral configuration and operations are clearly separated from application code. This is the case today in most embedded programs: peripherals are only accessed through a well defined API referred to as *driver calls*. It is compatible with both bare-metal and embedded system programming habits.

B. Power model for peripherals

The behavior of any peripheral can be modeled using a finite state machine. It can be as simple as a two-state on/off

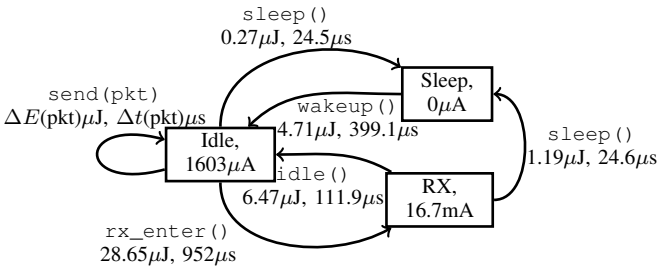

Fig. 2: Driver state machine of the driver for CC2500EMK radio daughter board. State is `idle` after driver initialization. The values were measured as shown in Section V.

machine, or it can be much more complex as, for instance, a radio chip state machine. These state machines are usually documented in the hardware datasheet. In general, it is not necessary to model peripherals at a fine grain. Only the state machine of the driver API itself, which is either a subset of the actual device state machine or a higher-level finite state machine, may be considered. The proposal of this work is to use these *driver state machines*, such as the one represented in Fig. 2, to model the power consumption of the peripheral devices. In a driver state machine, each state is considered to have its own power state, *i.e.*, current consumption, assumed to be constant until the peripheral state changes. This is a simplifying assumption but, as discussed in Section IV, it is sufficient to obtain a fair estimation of reality.

Each *driver call*, *i.e.*, transition between driver states, has a given cost in execution time and energy. The arguments passed to the driver routines might have an influence on these metrics. Sending a radio packet is a typical example of such a parameterized driver routine because the duration and the energy consumption of the call grow with the size of the packet. However, from our experience, most of the durations and energy values associated to *transitions between states* do not depend on the driver call parameters, but this strongly depends on the driver API comprehensiveness.

A typical example of such a power state machine is the one of the radio chip used in this study. The hardware chip is Texas Instruments' CC2500EMK daughter board. Fig. 2 shows the power state machine of its driver. As one might notice, there are fewer states than in the state machine specified in its datasheet. The driver developer has exposed only a subset of the actual hardware states of the radio chip. Fig. 2 depicts only three states: `Idle`, `Sleep` and `RX`. There is no dedicated state for data transmission. Indeed, the driver call for transmission starts from the `Idle` state, temporarily switches to the hardware transmission state, then switches back to `Idle` state when the radio packet is sent. Hence, from the driver perspective, the state did not change.

The micro-controller itself also has a power state machine. It has several operating modes: an active mode enabling software to make progress, and the low-power modes disabling several CPU components. The power state automaton of the micro-controller is shown in Fig. 3.

Considering separate state machines instead of explicitly

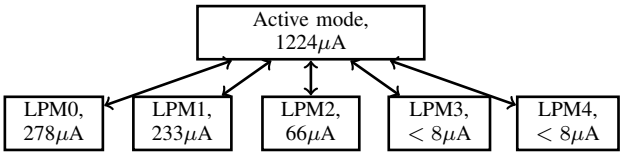

Fig. 3: Power state machine of the MSP430FR5739 micro-controller. Initial state is `Active mode`. The values were measured as shown in Section V.

enumerating all power state combinations at platform-level, in addition to using a high-level driver API, makes this paper's proposal realistic and scalable to any embedded system.

C. Power supply model

This work focuses on two power supply models. The first model consists in supplying the platform with continuous supply. It corresponds to battery-powered scenarios where the battery is able to supply steady power during years. The second model consists in harvesting energy and storing it into a capacitor. Power outages are likely to occur often. Hence, this model requires a power manager in order to schedule charge and discharge phases. Both these models also assume that the platform is supplied with a constant, steady voltage supply. This assumption is realistic since voltage regulators are used in continuously-powered systems and in some power managers for intermittent systems based on energy harvesting [2].

Next section presents the experimental platform built to measure the elementary power consumption of both CPU instructions and device driver calls.

IV. Power Monitoring Device

There is a plethora of ways to measure instantaneous current. All of them have their advantages and drawbacks, as well as their own performance profiles: supply voltage perturbation, amount of noise, dynamic range, *etc.*. In the particular case of this work, the measurement platform must provide high sampling frequency because power consumption may change every dozen of instructions; it must also provide high dynamic range.

The measurement circuit relies on a simple design, similar to that of EMPIOT [26]. The circuit, shown in Fig. 4, includes a shunt resistor of 0.2 Ω in series on the high-end of the supply voltage of the device under test (DUT). The voltage across the shunt resistor is amplified using Texas Instruments' INA212 operational amplifier with a gain of 1000 V/V. The output of the operational amplifier is fed to Texas Instruments'

[2]https://e-peas.com/types/energy-harvesting/

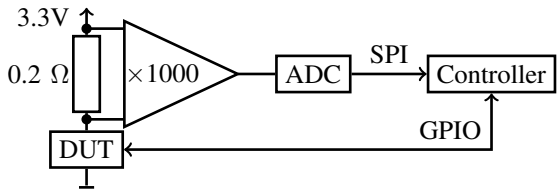

Fig. 4: Measurement circuit schematics.

ADS8661 12-bit Analog-to-Digital Converter. The controller circuit is a Raspberry Pi 3 Model B. Its regulated 3.3V power rail powers the device under test, while the unregulated 5V power rail powers both the operational amplifier and the ADC. Before measuring, the monitoring device is calibrated using a set of known resistors that emulate fixed, constant current loads. The calibration is a linear regression.

The circuit actually measures the voltage across the shunt resistor, which is linear to to the current $i(t)$ drawn by the device under test. As mentioned in Section III-C, the supply voltage V_{CC} is constant over time. The integral of the current can be substituted by the integral average of the current, multiplied by the elapsed time. In our case, the current data is discretized so we used the discrete current average $\overline{I}$ as an approximation of the integral average during the sampling time interval $[t_0, t_1]$. ΔE, defined as the amount of energy consumed between t_0 and t_1, is then:

$$\Delta E = V_{CC} \times \overline{I} \times (t_1 - t_0) \quad (1)$$

The current $i(t)$ is obtained from the ADC values: $i(t) = I_{step} \times X_{ADC}(t) + I_{offset}$, where I_{step} is the current increment for each ADC step, X_{ADC} the value returned by the ADC and I_{offset} the measurement offset due to the operational amplifier. The average of the ADC data over $[t_0, t_1]$ defines $\overline{X_{ADC}}$. We then obtain Equation (2), which links energy consumption and ADC sampled values. After calibration, $I_{step} = 6.4$ μA/step and $I_{offset} = -4.0$ μA.

$$\Delta E = V_{CC} \times (I_{step} \times \overline{X_{ADC}} + I_{offset}) \times (t_1 - t_0) \quad (2)$$

The whole monitoring device achieves a dynamic range between 8 μA and 26 mA, samples at 170 kHz and presents a noise that corresponds to 6 LSB. High-frequency current noise induce errors in energy measurement. But since integrated high-frequency signals result in small quantities, the overall error is expected to be small as well.

This power monitoring methodology has the advantages of being simple, low-cost and compatible with any platform since the only hardware requirements are basic: a power supply pin on which to plug the shunt resistor, and a couple of GPIOs. The value of the shunt resistor and the gain of the operational amplifier are specific to the targeted platform though.

V. Experimental Results

The platform depicted in Section IV enables to measure the instantaneous current drawn by the device under test. For instance, Fig. 5 shows the evolution of the platform current during a radio emission of a 128-byte long packet (`rf_send_packet(pkt)` driver call in Table I). In this specific example, the behavior of the radio transmission is simplified during simulation, but its simulated energy consumption is still accurate, as shown in Table II.

A. Regular driver calls

Measurements of duration and energy consumption for driver calls, as well as state transitions, are given in Table I. Some driver calls *appear* to consume no energy, because their consumption is already accounted in the consumption of the platform over the measured duration.

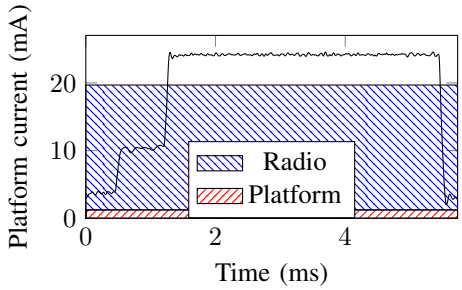

Fig. 5: Instantaneous current consumption (black curve) for a transmission of a 128-byte long radio packet. Hatched areas depict the equivalent surface corresponding consumption of the radio (blue) and the rest of the platform (red). These averaged values are used when simulating such a transmission.

TABLE I: Examples of driver calls measurements for the radio, temperature sensor and accelerometer, as well as the difference in measured current levels between `on` and `off` states of the memory protection unit, accelerometer and LED. $\Delta E(x)$ and $\Delta t(x)$ are represented on Fig 6.

Driver call	Δt	ΔE	from	to
`rf_init`	451 μs	10.3 μJ	uninit	init
`rf_config`	575 μs	4.5 μJ	init	idle
`rf_rx_to_idle`	112 μs	6.0 μJ	RX	idle
`rf_wakeup`	399 μs	3.1 μJ	sleep	idle
`rf_idle_to_sleep`	25 μs	0.2 μJ	idle	sleep
`rf_rx_to_sleep`	25 μs	1.1 μJ	RX	sleep
`rf_rx_enter`	952 μs	28 μJ	idle	RX
`rf_send_packet(x)`	$\Delta t(x)$	$\Delta E(x)$	idle	idle
`temp_init`	159 μs	0 μJ	uninit	init
`temp_sample`	76 μs	0.2 μJ	init	init
`accel_init`	55 μs	0 μJ	uninit	off
`accel_on`	1025 μs	46 μJ	off	on
`accel_off`	10 μs	0 μJ	on	off
`accel_sample`	171 μs	0 μJ	on	on

Driver state	$i_{on} - i_{off}$
`MPU`	0.54 μA
`Accelerometer`	385 μA
`Single LED`	1230 μA

Some drivers might only present a simple, two-state machine. For instance, LEDs are based on GPIOs, but since it only makes sense to use them as output, then the state machine of each LED has two states: `on` and `off`. Table I also shows current measurements for some peripherals modeled as `on`/`off` drivers. Their power model could be more complex, however in practice, their driver routines currently only achieve transitions between two states. Current is computed by averaging the platform current during some time.

B. Driver routines with parameters

A parameterized driver call is exemplified by the `rf_send_packet` driver call of radio peripheral. The measured energy and duration of the packet-sending driver routine are shown in Fig. 6. For each packet length from 1 byte to 254 bytes, packet emission was measured 64 times.

Energy measurements have a standard deviation not higher than 3.6% of the average value. Duration measurements have a standard deviation of at most 11.4 μs in a few cases, and 1.8 μs in average. The relationship between energy and packet length may be modeled as simply as a linear regression: $\Delta E(L) = 2.39 \times L + 47.71$ μJ, where L is the packet length. The relationship between run-time and packet length is better modeled as a *two-part* linear regression which parameters change at 64 bytes, that is the size of the internal transmission FIFO of the radio peripheral. For packets smaller than 64 bytes, $\Delta t(L) = 38.1 \times L + 1263.3$ μs. For larger packets, $\Delta t(L) = 32.0 \times L + 1646.3$ μs.

The discrepancy between energy and time behaviors originates from the energy of populating the FIFO being a few orders of magnitude lower than the energy of actually sending the packet. Hence, the influence of the FIFO limitation is less visible regarding energy in comparison to time.

C. Integration in System Simulation

The model described in Section III-B was implemented on a simulator [3] built on top of ArchC [1]. ArchC is a language for CPU architecture and Instruction Set Architecture description. It aims at generating cycle-accurate SystemC code for simulation purposes. We wrote an ArchC model of the MSP430X instruction set, in order to run the binary images compiled for MSP430FR5739 without any modification.

While the genuine software part is run in a cycle-accurate fashion, the driver part is run symbolically: instead of actually running every single instruction of a driver call, the whole routine is bypassed and only its functional effects are simulated. The time duration and energy consumption are directly taken from the measurements presented above.

The simulator comes with two power models: (i) continuous supply and (ii) energy harvester with a power manager that stores energy into a capacitor and powers the device under test through a voltage regulator as mentioned in Section III-B. In the second scenario, the voltage conversion is considered conservative so far. When the capacitor voltage drops below a certain threshold, the simulator generates an interrupt and calls the software-defined interrupt handler of the kernel as it would be done in a real scenario. Then, when the capacitor voltage drops further to a lower threshold, the simulator virtually switches the device off, refills the capacitor and restarts the device by running the reset entry of the kernel interrupt vector.

Fig. 7 illustrates the simulation, under continuous supply, of a specific application which repeatedly senses accelerometer and temperature data and sends the data over radio. The graph also shows actual current values measured on the same application on real hardware.

D. Application benchmark

The simulation is tested against a benchmark made of the following applications:

a) LEDs: counts from 0 to 255, displays the counter on the LEDs and waits 1 ms before incrementing the counter.

b) Accelerometer: turns on the accelerometer, performs 10 measurements and turns off the accelerometer, waits 1 ms, repeated 256 times.

c) Radio: puts the radio to sleep mode, waits 1 ms, puts the radio to idle mode and sends a 128-byte long packet, repeated 256 times.

d) Complete-WSN: uses accelerometer, temperature sensor and radio. Senses acceleration and temperature N times and sends this record over radio as a single packet, with N varying from 1 to 31. The radio is sleeping while sensing data and the accelerometer is always on. The beginning of the simulation of this application is shown on Fig. 7.

The results of these simulations are shown in Table II. Simulation achieves less than 1% time estimation error and less than 5% energy estimation error [4] which makes this methodology suitable for a precise estimation of ultra-low power embedded systems with important peripheral usage.

VI. Conclusion

The main contribution of this work is a new simplified power model that enables the accurate simulation of ultra low-power embedded systems using peripherals. This is achieved by modeling driver calls at coarse grain, in a parametric way, while the rest of the code is simulated at instruction-level.

We validate the proposed model by implementing a simulator that may be adapted to support any Instruction Set Architecture and any peripheral. The simulator targets an MSP-EXP430FR5739 platform with an accelerometer, a temperature sensor and an external radio chip.

[3] https://github.com/gberthou/archc-msp430x/tree/sytare-syscalls

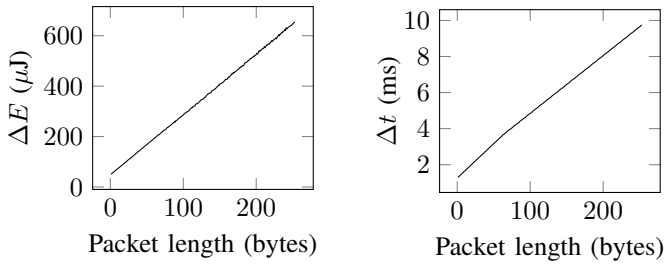

Fig. 6: Measured (a) energy consumption (ΔE) and (b) duration (Δt) of radio emission for different packet lengths.

[4] The higher error in energy estimation is partly due to accumulated rounding errors; we are currently working on this aspect.

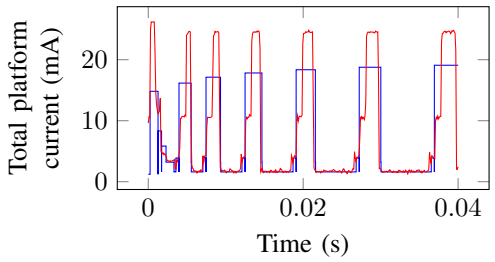

Fig. 7: Excerpt of simulation record (blue) and actual measurement (red) for the Complete-WSN application, using radio, accelerometer and temperature sensor.

TABLE II: Comparison, for execution time and energy consumption, between measurements and simulation of the benchmark applications on the MSP-EXP430FR5739 platform.

Application	Measured Δt	Simulated Δt	Δt error		Measured ΔE	Simulated ΔE	ΔE error
LEDs	274 ms	274 ms	0.0 %		5596 μJ	5595 μJ	0.0 %
Accelerometer	948 ms	959 ms	1.2 %		17113 μJ	16309 μJ	4.7 %
Radio	1815 ms	1836 ms	1.2 %		102706 μJ	107285 μJ	4.5 %
Complete-WSN	783 ms	781 ms	0.3 %		16114 μJ	16315 μJ	1.2 %

The energy consumption of the peripherals and driver calls has been precisely obtained using a low cost measurement platform, hence accessible to any embedded system designer. For this platform, the simulator achieves good estimation: less than 1% time estimation error and less than 5% energy estimation error.

An important aspect of this work is that it is suited to intermittent systems simulation as well. We are currently investigating the use of this work to (i) statically estimate viability of ultra-low power embedded applications with respect to specific intermittent power sources and (ii) provide a run-time decision helper for the application to choose between several possible tasks to execute depending on the available energy and task characteristics.

ACKNOWLEDGMENT

This work is partially supported by Inria (IPL ZEP).

REFERENCES

[1] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo, "Archc: a systemc-based architecture description language," in *16th Symposium on Computer Architecture and High Performance Computing*, 2004.

[2] C. Morais, D. Sadok, and J. Kelner, "An IoT sensor and scenario survey for data researchers," *Journal of the Brazilian Computer Society*, 2019.

[3] A. Kozłowski and J. Sosnowski, "Energy efficiency trade-off between duty-cycling and wake-up radio techniques in IoT networks," *Wireless Personal Communications*, 2019.

[4] S. Popli, R. K. Jha, and S. Jain, "A survey on energy efficient narrowband internet of things (NBIoT): Architecture, application and challenges," *IEEE Access*, 2019.

[5] J. Hester, K. Storer, and J. Sorber, "Timely execution on intermittently powered batteryless sensors," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017.

[6] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith, "Design of an RFID-based battery-free programmable sensing platform," *IEEE Transactions on Instrumentation and Measurement*, 2008.

[7] Y. Lee, S. Bang, I. Lee, Y. Kim, G. Kim, M. H. Ghaed, P. Pannuto, P. Dutta, D. Sylvester, and D. Blaauw, "A Modular 1 mm^3 Die-Stacked Sensing Platform With Low Power I2C Inter-Die Communication and Multi-Modal Energy Harvesting," *IEEE Journal of Solid-State Circuits*, 2013.

[8] A. Colin, E. Ruppel, and B. Lucia, "A reconfigurable energy storage architecture for energy-harvesting devices," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.

[9] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *IEEE Embedded Systems Letters*, 2015.

[10] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent execution without checkpoints," *Proc. ACM Program. Lang.*, 2017.

[11] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac, "Peripheral state persistence for transiently-powered systems," in *Global Internet of Things Summit (GIoTS)*. IEEE, 2017.

[12] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac, "Sytare: a lightweight kernel for NVRAM-based transiently-powered systems," *IEEE Transactions on Computers*, 2019.

[13] A. R. Arreola, D. Balsamo, G. Merrett, and A. Weddell, "Restop: retaining external peripheral state in intermittently-powered sensor systems," *Sensors*, 2018.

[14] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level power profiling for low-power wireless networks," SICS, Tech. Rep. 2011:05, 2011.

[15] K. Klues, V. Handziski, C. Lu, A. Wolisz, D. Culler, D. Gay, and P. Levis, "Integrating concurrency control and energy management in device drivers," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, 2007.

[16] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: Tracking energy in networked embedded systems," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, 2008.

[17] A. Acquaviva, L. Benini, and B. Riccó, *Energy Characterization of Embedded Real-Time Operating Systems*. Springer US, 2003.

[18] S. Schubert, D. Kostic, W. Zwaenepoel, and K. G. Shin, "Profiling software for energy consumption," in *2012 IEEE International Conference on Green Computing and Communications*, 2012.

[19] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat, "Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, 2018.

[20] N. Cherifi, T. Vantroys, A. Boe, C. Herault, and G. Grimaud, "Automatic inference of energy models for peripheral components in embedded systems," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2017.

[21] A. Di Nisio, T. Di Noia, C. G. C. Carducci, and M. Spadavecchia, "High dynamic range power consumption measurement in microcontroller-based applications," *IEEE Transactions on Instrumentation and Measurement*, 2016.

[22] J. San Miguel, K. Ganesan, M. Badr, and N. E. Jerger, "The EH model: Analytical exploration of energy-harvesting architectures," *IEEE Computer Architecture Letters*, 2018.

[23] V. Shnayder, M. Hempstead, B.-r. Chen, G. Werner-Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *SenSys'04 - Proceedings of the Second International Conference on Embedded Networked Sensor Systems*, 2004.

[24] T. Bouhadiba, M. Moy, F. Maraninchi, J. Cornet, L. Maillet-Contoz, and I. Materic, "Co-simulation of functional systemc tlm models with power/thermal solvers," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, 2013.

[25] S. Ahmed, A. Bakar, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola, "The betrayal of constant power $\times$ time: Finding the missing joules of transiently-powered computers," in *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2019.

[26] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, "EMPIOT: An energy measurement platform for wireless IoT devices," *Journal of Network and Computer Applications*, 2018.

[27] F. Fummi, G. Perbellini, D. Quaglia, and A. Acquaviva, "Flexible energy-aware simulation of heterogenous wireless sensor networks," in *2009 Design, Automation Test in Europe Conference Exhibition*, 2009.

[28] O. Hahm and S. Adler, "Profiling energy consumption of wireless sensor nodes with almost zero effort," in *2012 IEEE International Conference on Communications (ICC)*, 2012.

[29] T. Hönig, H. Janker, C. Eibel, W. Schröder-Preikschat, O. Mihelic, and R. Kapitza, "Proactive energy-aware programming with peek," in *Proceedings of the 2014 International Conference on Timely Results in Operating Systems*. USENIX Association, 2014.