Automatic Inference of Energy Models for Peripheral Components in Embedded Systems

Nadir Cherifi, Thomas Vantroys, Alexandre Boé, Colombe Hérault, Gilles Grimaud

Automatic Inference of Energy Models for Peripheral Components in Embedded Systems

Nadir Cherifi[*+1], Thomas Vantroys[*], Alexandre boe[x], Colombe Herault[+2], and Gilles Grimaud[*]

[*]Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, F-59000 Lille, France
[x]Univ. Lille, CNRS, Centrale Lille, ISEN, Univ. Valenciennes, UMR 8520 - IEMN, F-59000 Lille, France
[+]Worldline e-payment services, Lille, France
[1]*nadir.cherifi@ed.univ-lille1.fr*
[*x]{*thomas.vantroys, alexandre.boe, gilles.grimaud*}*@univ-lille1.fr*
[2]*colombe.herault@worldline.com*

***Abstract*—Surrounding autonomous embedded devices are in a constant expansion. The advent and the rise of Internet of Things (IoT) enable these objects to take a giant step forward, especially regarding their large scale deployment in real-world applications of the everyday life. A significant part of these objects are battery-powered and energy-dependent. Thus, energy is a critical resource which greatly complicates the development of the embedded software. By decomposing the energy consumption of a battery-powered IoT device, we can see that peripheral components are the major contributors among the overall consumption. Indeed, these components are exploited and repeatedly used by the object to interact and communicate with its surrounding environment during all the application lifetime. Acquire the expertise to handle accurately, during the development stage, the behaviour of every on-board peripheral component is a big challenge to improve the development of IoT embedded applications.**

To guide the developer in this task, we propose an automated inference procedure of energy models for peripheral components. An accurate automata-based model of the energy consumption can be generated, with only little efforts from the developer, based on real runtime measurements, providing precise energy figures. The proposed process is focused on a lightweight code generation step and simple analyses of the energy output traces, allowing a quick regeneration of the models in the case of a peripheral component modification. We show the potentials of the proposed procedure by real experiments on real peripherals. The obtained results are satisfactory, and we believe that our proposition is able to enhance the embedded development in an energy-constrained environment.

I. Introduction

We can summarize the Internet of Things (IoT) as the idea of a system where the surrounding objects of the physical world are directly connected to the traditional Internet via wireless or wired connections. Todays life is more and more augmented with various connected objects in order to simplify common tasks and provide extra informations to the users. To fully achieve this objective and allow the objects to perform their basic functions (sensing, processing, actuation, communication), a complex application has to be embed on the object.

Acknowledgements: This work has been supported in part by IRCICA USR 3380 CNRS - Univ. Lille, F-59000 Lille, France.

In most of battery operated and wireless connected devices, the replacement or the recharging of the battery is binding, difficult or even impossible. For instance, connected smart meter for home and building, or buried connected sensors [1] for civil structure health monitoring represent perfect examples. Indeed, any small bug or messed up settings in a single source code function of the embedded application driving the object could drastically drain the battery, reducing the device lifetime or making it unusable. Moreover, the global energy consumption caused by a high concentration of connected objects in our every day life can be significant. In this way, the International Energy Agency (IEA) asks the developers to a better management of the object energy consumption, during the development stage [2]. Consequently, energy consumption and efficiency become one of the main concerns in the embedded software development.

In order to help the developer to take into account the energy constraint, we propose to focus on a specific part of an embedded system which is the peripheral components. These peripherals could roughly be classified into three categories: sensors, actuators and communication components. Usually, a connected object makes an intensive use of its peripherals, since they represent the only way to interact and to communicate with the environment. Due to the repetitive activations of the peripheral components, they are often designated as the responsible of a significant portion of the overall energy consumption. Furthermore, it is more complex for a developer to take into account the energy behaviour of peripherals than the one of microcontroller.

It appears that the design of an energy efficient embedded software requires from the developer a perfect knowledge of all the peripheral components used on the connected object. To achieve this goal, we propose to help the embedded developer to get an accurate comprehension of the hardware peripheral components used. The main contribution of this article is the proposition of an automatic energy model inference process dedicated to embedded peripheral components. The energy models generated are directly linked with the API (Application Programming Interface) functions of the embedded software

driver that controls the peripheral component activity. We believe that this schema allows the developer to get easily aware of the energy consumption, induced by each action performed by the embedded peripheral component. This work is based on an existing energy profiling framework already described and presented in a precedent research article [3].

This paper is organized as follows. In Section 2, we present some related works regarding energy model inference and energy profiling methods for embedded systems. We recall in Section 3, the inner working of the energy profiling framework. In Section 4, we present our contribution on the automatic generation of energy consumption models linked to the embedded software. We conclude in Section 5 by giving some possible future works.

II. Related Works

this work takes place at the crossing point of energy profiling, energy model creation, and small embedded systems. At the best of our knowledge, there have been only few studies that target a part of the same objectives. Regarding the energy profiling of small embedded systems, existing approaches could globally be divided into 3 distinct categories: hardware-based, software-based and simulation-based. Our approach is based on a runtime hardware method, allowing to acquire real energy figures of the embedded system during execution. In [4] authors describes ICount, a small and inexpensive design adding an energy metering with a little hardware overhead to the target board to be profiled. As an extension, the authors present the software energy profiler Quanto [5], which takes advantage of the on-board energy meter ICount. It instruments the underlying embedded device drivers to track the hardware components behaviours and account for the activities to which they contribute. Despite obvious advantages, the low accuracy of the ICount module makes it unsuitable for fine-grained energy consumption profiling.

Regarding industrial works, JTAG probes with shunt resistor based measurement circuits are used in the IAR I-jet probe [6] and the Atmel power debugger platform [7]. To profile the energy consumption of an embedded application, the first takes advantage of the Embedded Trace Macrocell (ETM) [8] integrated in some specific ARM MCUs. The second uses a statistical sampling method of the target board PC (Program Counter). The main drawbacks of these methods are their high cost and their proprietary schemes which restrict the usage on other embedded boards.

Regarding embedded energy model construction. The Eprof mobile profiler [9] is a fine-grained energy profiler for mobile applications. Eprof links each embedded component with a system-call driven Finite State Machine (FSM) energy model, describing its energy behaviour and the impacting system calls. Then, the prototype logs the application system calls and routines. The collected logs are analysed, and the amount of energy in the current FSM state is assigned to a each traced routine. Eprof solution is elegant, but no energy model inference mechanism is proposed. Besides, it takes advantage of Operating System (OS) facilities to perform its routines tracing, which is quite impossible to transpose on constrained embedded objects, where most of the time, there is no real OS implemented.

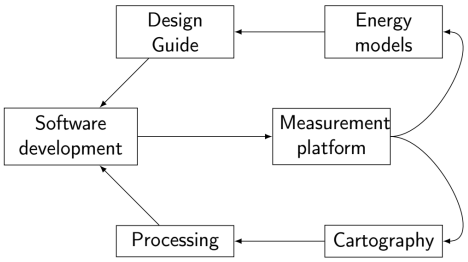

Fig. 1: The energy-based iterative development cycle, aiming to enhance and to simplify the software development for energy-constrained embedded applications [3].

Finally, an automata-based model mining method was proposed in [10]. While some objectives remain similar with our proposition, the mining method presented in their work does not focus on the energy consumption of embedded systems. Besides, no real help or guiding are provided to the developer which is in the core of our works.

III. Energy Profiling Proposition

In this Section, we describe in summary, the energy-oriented development methodology and the energy profiling framework. It uses a runtime hardware-based measurement technique, and it is able to help the embedded developer to energetically profile its embedded application source code. Built on this framework, the energy model inference proposition will be described in the next section.

A. Energy-oriented Development Cycle

To enhance the embedded development in an energy-constrained environment, we strongly think that not only a good measurement technique is needed, but the entire development cycle has to be centred on the energy issue. The proposed energy development cycle, illustrated in Figure 1, is focused on improving the energy-based embedded development. In this configuration, the *software development phase* and the *measurement phase* are at the center of the cycle. Indeed, they represent the main and critical steps upon which the entire cycle, and then the optimisation phases, are based on.

We can decompose the proposed cycle into two parts. The first (i.e. the bottom part) provides an energy cartography of the source code, and a visual feedback is presented to the developer. Following these processes, the developer will have in hands the inputs which allow him to modify his software, and replay a new energy measurement cycle. These processes are explicitly illustrated in the Figure 2, and explained in this Section. The second part of our development cycle (i.e. the upper part) describes an energy model inference process. The process is based on the energy cartography framework, and is detailed in the next Section.

The major part of the framework is developed in Python, the instrumentation process is developed in C/C++ using the

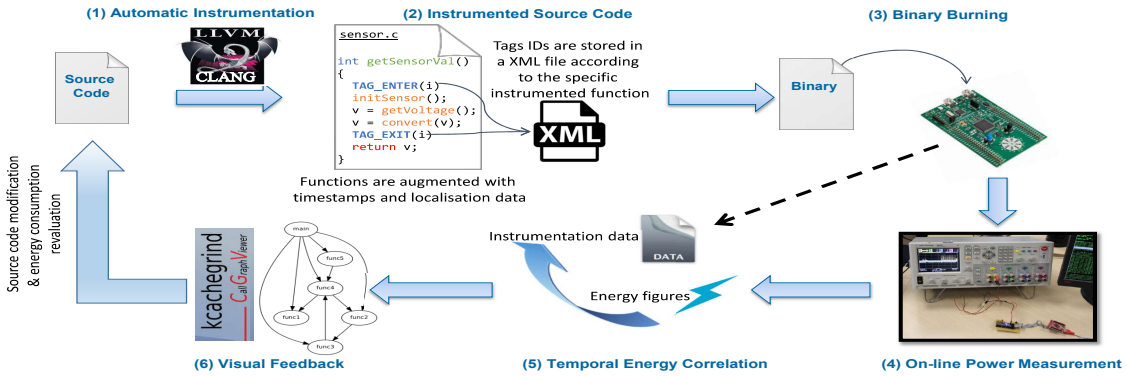

Fig. 2: The energy profiling cycle of the proposed energy framework [3].

Clang-LLVM framework [11], and the statistical processes are implemented using the R environment which communicates seamlessly with the rest of the framework using the Python *rpy2* package.

B. Automatic Source Code Instrumentation

The automatic (i.e. transparent) code instrumentation is introduced to get logical information from the embedded application on runtime. The major goal of our framework is to simplify and optimize the energy profiling phase. We decided to automatise the entire instrumentation procedure. We use the CLANG-LLVM framework to build a third-party application in charge of automatically instrument the embedded application source code (Figure 2.1). It inserts specific source code tags on functions entry and exit (Figure 2.2). The functions to be instrumented are specified by the developer, who can also use the framework to perform a more global and spread instrumentation. In this case, the framework is able to recursively search and instrument all the called sub-functions, by the ones specified by the developer, on the call graph.

Each inserted tag is only in charge of getting the current time (timestamps) from a hardware counter, the address of the running function, and the address of the calling function. The instrumentation data resulting from the execution of the inserted tags are directly saved in the embedded system memory (i.e. RAM), and are sent out via a serial interface at a high baud rate. Among these data, we use the saved timestamps to locate, the instrumented functions entries and exits (Figure 2.5), on the output energy traces. The measurement task, and the hardware counter on the embedded board are synchronized at the beginning of the experiment by a simple digital GPIO. Finally, using the gathered instrumentation data, we can easily compute the energy consumption of each instrumented function by retrieving at the same time a partial dynamic energy call graph.

C. Incremental Energy Instrumentation

The instrumentation performed by the framework is a valuable proposition to get logical data allowing to correlate the energy consumption with the executed embedded application source code. However, the more the number of instrumented functions is high, the greater the overhead caused by this instrumentation is sizeable. To solve this issue, we introduce an automatic incremental instrumentation feature. The global idea is to claim that the functions which are associated with a highly variable power behaviour on the output energy trace are functions that possibly perform multiple different treatments. Therefore, It is more interesting, in terms of knowledges on the power consumption, to deeply instrument the functions with a variable power behaviour than to instrument functions leading to more stable power.

To achieve this idea, we use a peak and variability detection algorithm implemented on the statistical R framework. If an instrumented function power curve exhibits a variability higher than a threshold, the called sub-functions are instrumented as well, and a new measurement phase is engaged.

D. Hardware Measurement Board

To effectively measure the power consumption, the energy framework is supported by a runtime hardware-based measurement approach. We use as measurement platform, a laboratory power analyser, the Agilent N6705A (Figure 2.4). It provides a high accuracy, a high resolution and a maximum frequency sampling of 50 kHz. It supports a large range of voltage and current configurations, which are sufficient to cover most of embedded device power requirements. The Agilent N6705A can supply up to fours ways, and simultaneously measure the current. This allows to deeply profile the energy consumption of an embedded system by measuring separately up to three external peripherals.

E. Energy Visual Feedback

Our energy framework links to each instrumented function entity all its occurrences, and to each occurrence, the energy consumption value and the caller function. This allows to compute a dynamic energy call graph. In order to help the developer to better understand the energy consumption of the embedded application, we provide two graphical feedbacks. The first simply draws the power curves of the experimentation synchronized by the timestamps of the function entries and exits. The second feedback consists in the presentation of a clear and a schematic view of the energy call graph (Figure 2.6).

We use in this way, the open-source visualizer, KCachegrind [12].

IV. The Energy Model Inference Process

The energy cartography framework, as presented in the previous section, allows a developer to understand the synchronous energy consumption of the embedded application. This task is almost transparent to the developer, who is asked only for few interactions on the framework process to get the desired results. However, this does not allow him to be proactive enough in the use of the embedded peripheral components on the embedded system. To go further, we propose in this paper, a process for automatic generation of energy models for embedded peripheral components.

We present the motivations behind the design of the energy model inference process, and its inner working. We illustrate its functioning with a simple example on a transceiver component of a hype long range communication technology which is LoRa [13].

A. Motivations

The overall motivation of our proposal is to provide to the developer a means to be proactive on the development of his embedded application under energy-constrained conditions. We assume that the embedded peripheral components are the main and critical part to handle in terms of energy consumption. Since the peripheral components represent the only way for a connected object to interact with its environment, they are constantly used by the application driving the object functionalities. An object will often use different sensors (e.g. temperature, gas, etc.) to collect various informations about its surrounding environment. It would send these data directly or after processing to the outside using a communication medium such as WiFi, Cellular or even LoRa and SigFox. Besides, according to the orders received from the outside or from its own initiative, the object could be required to activate actuators such as a motor engine to move in the environment, for instance.

Our objective is to help the developer to understand the energy consumption of these peripheral components. Indeed, we think that an inference process of energy models for peripheral components could help the developer to gain a better comprehension of the embedded system energy consumption. Generally, in the literature, energy model creation is a hard and a long task. The creation phase itself requires a long period of energy samples acquisition, and the second phase represents the validation of the energy model by proving its accuracy in real environment conditions. To overcome this barrier and simplify the handling of the generated models, we decide to go on a higher level of abstraction, and make each peripheral energy model centred on the API functions driving the component functionalities. This schema allows a reasonable generation time of the models, and ensure, thanks to the hardware measurement based framework, the accounting of the real world conditions.

Nevertheless, our motivations go even further. Indeed, the resulting energy models can be used by the developer to create an energy-based documentation of each peripheral. The documentation could highly differ from a manufacturer data sheet as the first is based on runtime energy measurements, and takes into account the surrounding environment (deployment environment noise, interaction with the target platform, etc.). Besides, a component data sheet is not always in accordance with the real energy figures [14]. Finally, a motivation and a long-term objective of our proposal is to be able to use the provided energy models as inputs for energy simulators in order to make them more reliable, and reduce the gap existing between the simulation and the real world [15].

Behind these motivations, to validate the usability of our energy model inference design, we base our reasoning on solid bases:

- The embedded peripheral components (i.e. I/O operations) represent the major part of a battery-powered object energy consumption. Frequently, these objects embed really low power microcontrollers that consume only a little part among the overall energy consumption. Hence, to help the developer, we should focus on the peripheral energy consumption and its modeling.

- Most of the time, the software API of the drivers controlling the embedded peripheral components is lightweight, and contains a limited number of software functions. This attribute allows us to ensure about the usability of our inference method which is focused on the driver API functions to build the corresponding energy models.

- The simple and natural aspect of the resulting energy models leads to a reasonable generation time. Thus, a fast revaluation and regeneration of these models could be made to take into account new or modified development attributes (environment radio noise, modifications on the target platform, etc.).

B. Overview

The energy model inference process proposed in this article takes place in the second part (i.e. the upper part) of our development cycle illustrated in Figure 1. To answer to the requirements of this process, we take advantage of our energy cartography framework to perform various energy measurement passes, and collect multiple energy samples of the embedded system during real execution sequences. The energy models generated after processing tend to resume the energy behaviour of each peripheral, and particularly, the energy states and the states transitions highlighted by the acquired energy samples. The energy model inference process is illustrated in the Figure 3. It is composed of multiple steps, described later in this Section. In summary, taking the example of an embedded peripheral component, the developer has to select the software driver of this latter, and annotate all the functions that he wants to include in the energy model. According to the annotations, a

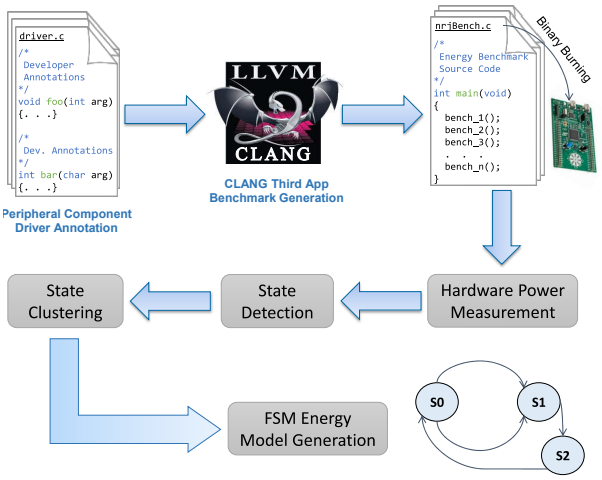

Fig. 3: The energy model inference process.

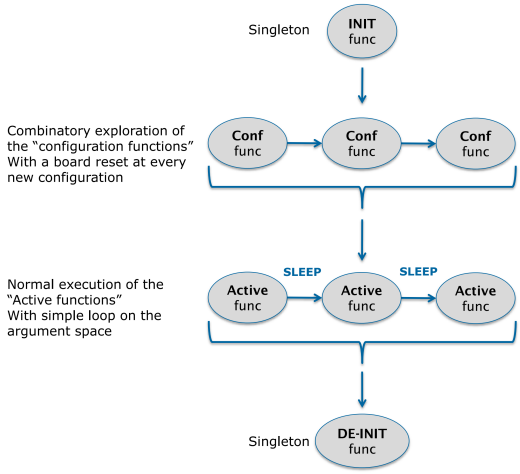

Fig. 4: Energy benchmark source code generation mechanism.

third party application designed using the Clang framework will generate multiple energy benchmark source code files, where each one represent a list of functions to be profiled. At this point, the energy framework described in Section 3, will be used to instrument the energy benchmark and profile their energy consumption, when executing them on the target hardware board. For each benchmark, multiple power measurements are performed, and an averaged energy trace is taken to be statistically compliant against noise.

The energy traces resulted are processed in order to detect all the existing energy states and the functions that have induced these states. Then, a state clustering step is performed in order to group similar states. Finally, An FSM-based energy model is generated for the selected peripheral component, where each transition in the FSM model represents one or multiple software driver API functions augmented by their arguments.

C. Energy Benchmark Source Code Generation

The energy benchmark source code generation represents the first step of our energy model inference process. The main objective is to simplify the creation of energy benchmark samples for the peripheral component to be profiled. To achieve this goal, we provide the developer with a lightweight JSON-based function annotation language, which allows him to guide the energy benchmark generation. For each peripheral component, the benchmark generator takes as inputs the software driver sources of the component containing the annotated functions. The generator analyses the annotations and generate multiple benchmark files that will be instrumented at their turns, using the instrumentation process of the energy cartography framework (c.f. Section 3). After that, each benchmark file goes for a runtime power measurement phase. The generator is a third party application developed using the Clang framework.

The lightweight language proposed allows a developer to select the type of an annotated driver function, the arguments space to be explored regarding the function parameters, and optionally the functions to explicitly call before or/and after the current annotated function. The dependences between the arguments,in terms of type and value, of all the functions included in a benchmark are resolved, within a certain limit of indirection, by the generator using the syntactic capabilities of the Clang framework. Regarding the function type, the developer can choose among 4 possibilities: Init, De-Init, Conf (i.e Configuration), and Active. The Init and De-Init functions are simply responsible for powering-up and powering-off the peripheral component, respectively. The configuration functions are the functions which are able to modify the internal state of the peripheral component. We can hold up as example, the modification of the transmission power or the frequency channel of a communication wireless component. Finally, the active functions represent a real and an effective workload of the peripheral component, such as the sending or the reception of communication frames. An example of such annotation is given below. This code listing focuses on having two possible configurations regarding the power transmission, and an exploration of the size argument space for the data sending function of the LoRa component module.

```
/** !MODELGEN! {
 "id":        "LoRa_Comm",
 "funcType":  "ACTIVE",
 "prevFuncs": [],
 "nextFuncs": [],
 "args": {
  "1": {"type":"uint8_t*", "act":"CREATE"},
  "2": {"value":{"inf":1, "max":251, step:50}}
}} */
void lora_send(uint8_t* buf, uint8_t size);

/** !MODELGEN! {
 "id":        "LoRa_Comm",
 "funcType": "CONF",
 "args": {
  "1": {"value":[POWER_MAX, POWER_LOW]}
}} */
void lora_setPower(pwr_t power);
```

The function types take an important place in the benchmark generation process. As we can notice it from the Figure 4, The generation follows a specific path to create all the benchmark files. Each benchmark file is composed, first by a single Init function, then multiple configuration functions, and multiple active functions, and finally a single De-Init function. The generator computes all the possible combinations

for the configuration functions, including their arguments space. For each combination, a new benchmark source code file is generated, and it includes this specific combination. Each file shares and contains the same Init, De-Init, and active functions. The active functions are included in a successive manner with local loops on the argument space defined by the developer, and a sleep time between each active function. The combinatory schema of the generation and the measurement phase could be seen as a time consuming processes. However, the software driver API of embedded peripheral components is most of the time lightweight. Consequently, the number of generated energy benchmark files for a component always remains reasonable.

D. State Detection

We describe in this Sub-Section, our detection process of the energy states which are present in an energy trace resulting from the runtime power measurement of a benchmark. The objective is to be able to isolate and determine the energy consumption and the different energy levels induced by the execution of particular functions of the software driver that controls the peripheral component.

First, we assume an energy trace as a simple time series. In Figure 5, we show an energy trace resulting from an energy benchmark which is related with a LoRa data sending of 6 frames of different sizes (1, 51, 101, 151, 201, 251 Bytes). The sending operation is preceded and followed by a power on and a power off operations, respectively. We give a formal definition of an energy trace and its characteristics:

Definition. (Energy Trace) A time series derived from an energy trace is a finite series of pairs (t_i, c_i) such as t_i represents the *timestamps (in Second)* and c_i describes the requested *current intensity (in Ampere)* by the target board under measurement. The attributes of this energy time series are:

- $i \in \mathbb{N}, \forall i \in [0, L]$ where L is the time series length: $t_i \in \mathbb{R}$ and $t_i < t_{i+1}$
- $i \in \mathbb{N}, \forall i \in [0, L] : t_{i+1} - ti = T$ where T represents the sampling period of the measurement (c.f. The sampling frequency is denoted as $f = \frac{1}{T}$
- The power measurement duration d (in Seconds) is obtained directly: $d = L.T$

The energy consumption E of the total trace or of a subset of it can be calculated. Assuming that the subset interval is $[t_a, t_b]$. We compute the energy consumption E (in Joules) by: $E_{t_{ab}} = \int_{t_a}^{t_b} P(t)dt$ with $P(t) = c(t).V(t)$. In all of the embedded object use cases studied, the voltage V (in Volt) is always fixed. Thereby, the energy trace is sufficient for the calculation of the energy consumption. The formula is then simplified: $E_{t_{ab}} = V \int_{t_a}^{t_b} c(t)dt$. Finally, we define an energy state as a subset of the energy trace denoting a real cut in the power consumption.

Regarding the detection of the energy states, we had already spoken in the previous section of a simple peak detection technique. However, in this specific case, our objective was much less critical. Indeed, the aim was to see whether there was a lot of peaks, sudden changes and a high variability in the current consumption behaviour of a function. Gaining this knowledge, we were able to state about the usefulness of exploring the called sub-functions.

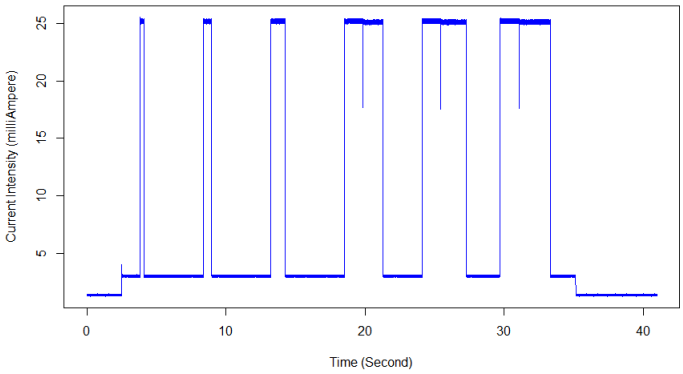

Fig. 5: An energy benchmark of a LoRa frames sending. The frames sizes are: 1, 51, 101, 151, 201, 251 Bytes.

1) The PETL Algorithm: In the aim of our proposition, We want to determine the energy states in an energy trace while keeping, at the same time, a good resistance against the noise and the peeks which are present in the power consumption trace. To achieve this goal, we have chosen to use a multiple changepoint detection algorithm proposed by Killick and Haynes in [16] and called PELT (Pruned Exact Linear Time). The PELT method is an "exact" parametric method that can be used to detect single as well as multiple breakouts (changepoints).

The PELT algorithm uses a common approach of detecting changepoints through minimising a cost function over possible numbers and locations of changepoints. To find multiple changepoints, the PELT method is first applied to the whole time series, then iteratively and independently to each partition, until no further changepoints are detected. The main assumption of the PELT algorithm is that the numbers of changepoints increases linearly with the increase of the data set. Hence, the changepoints are spread throughout the data and are not restricted to one portion of it. An important benefit of this algorithm is its speed, which has been shown to have a linear running time (i.e. O(n) complexity). The method is more accurate than approximate search methods and faster compared to other exact search methods. Finally, in our process, each changepoint identified denotes the end of the actual energy state and the start of a new one on the energy trace. The Figure 6 and Figure 7 show the results of the state detection process applied to an operation of multiple frame sending using a LoRa transceiver. A high transmission power and a lower one are used, and are illustrated by the difference in the power values in each case.

We want to highly stress that, the relation between a detected energy state and the software function inducing it, is maintained thanks to our energy cartography framework. In addition, some peripheral components can not be power measured individually (e.g. They are directly soldered on the hardware platform

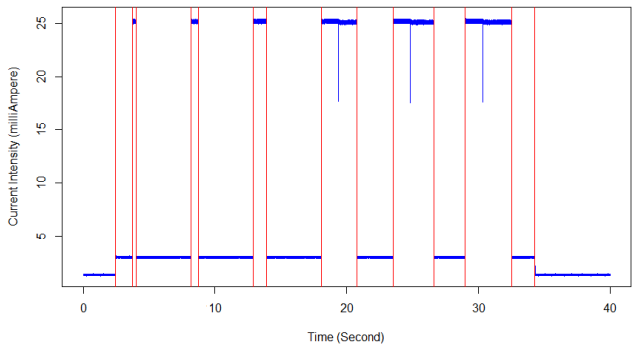

Fig. 6: The state detection results on the precedent LoRa sending energy benchmark. The frames are sent with a low transmission power.

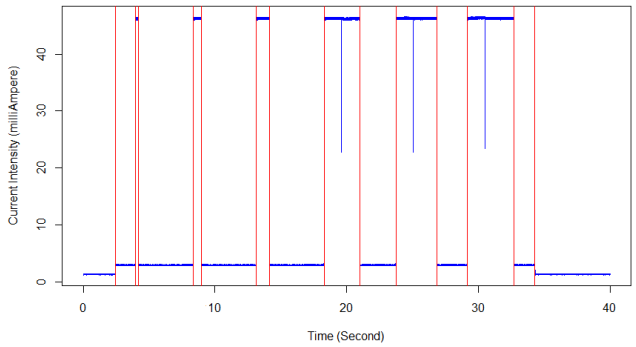

Fig. 7: The state detection results on the precedent LoRa sending energy benchmark. The frames are sent with a high transmission power.

board). We solve this issue, by using a base power state in our measurement. If a peripheral component is measured individually, this base power state is null, however, it equals to the mean power consumption of the hardware platform board, in an IDLE state of activity. Finally, some peripheral components could expose asynchronous energy consumption schema (i.e. The energy is still consumed after the execution completion of the software function). For these types of component, we implement a simple idea saying that: the energy effect of a function is finished, when the energy trace backs to a similar state than the one preceding the execution of the function.

2) Penalty Value Definition: The PELT algorithm takes as an input, in addition to the cost function to minimize, a specific term which refers to a *penalty* used to avoid under/over-fitting, and help to control the number of changepoints detected. In the statistic package used [17], there is multiple existing penalty types and methods for automatically (or semi-automatically) determine the penalty value. Nevertheless, after specific experimentations, none of these methods suit the accuracy that we look for in our state detection process. We decided to use a manual method for specifying the penalty value.

The use of a manual penalty comes with a basic rule: a lower penalty value results in more changepoints identified, whereas

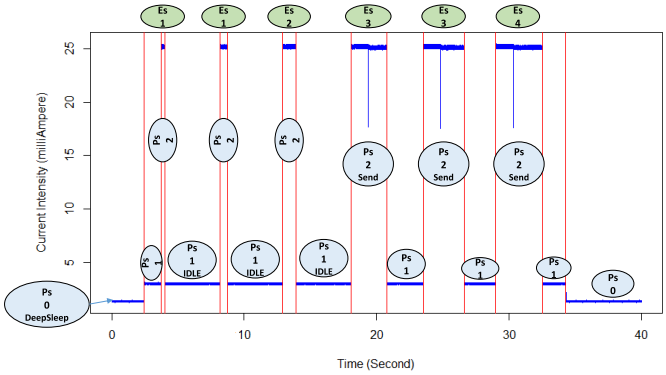

Fig. 8: Power and energy states after grouping.

a higher penalty leads to less changepoints discovered. To define the most appropriate penalty value for an energy trace, we follow a simple *elbow* approach. This method is defined by the variation of the penalty value and the computing of the number of changepoints identified against the penalty used. The exploration interval used for the penalty value is [0, 50] with a incrementation step of 0.5. The number of detected changepoints will fit a quick decrease which represents the changepoints that are mostly induced by the noise, and it will slow down until reaching the number of 0 identified changepoints. We choose as the most appropriate penalty value, the one which is situated after the decrease step, and provide the most stable number of changepoint among the neighbouring penalty values. Despite the penalty exploration space, the computation time of the elbow method remains relatively reasonable, especially thanks to the computation speed of the PELT algorithm, and the possibility to easily parallelize the processing.

E. State Clustering and FSM generation

After the execution of the state detection phase, all the energy states contained in all the energy traces are identified by the inference process. The last step consists in grouping all the similar energy states in order to form global states. Indeed, we propose two incremental groupings. The first concerns the *power states (Ps)* which are only defined by the power consumption of a state (i.e. Power of the state = mean Current of the state * Fixed Voltage). The second grouping is related with the *energy states (Es)* which are defined, in addition, by the duration time of the state (i.e. Energy of the state = Power of the state * duration time of the state). Besides, the energy states (Es) only have meaning for *"Active functions"*. At the end, each power state of the FSM-based energy model is augmented with the corresponding energy states. This decomposition into power and energy states is illustrated on the LoRa frame sending example, in the Figure 8.

The inference process has to group all the similar power states, and then the energy states, in order to form global states that are more descriminating. To achieve this task, we choose a clustering method, and we use the *Hierarchical Clustering Analysis* (HCA) [18] approach for its simplicity

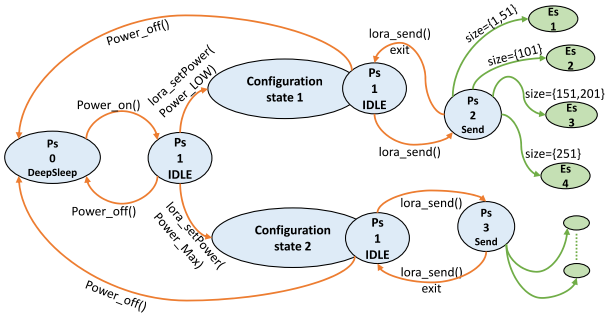

Fig. 9: FSM-based energy model for a part of the LoRa component transmission activities.

of implementation and its efficiency. This method classifies a set of points in $\mathbb{R}$ according to the distances (e.g. euclidean) between these points, forming clusters or classes. The method performs a grouping by taking the two nearest points at each time. Hence, a new point is created, and it represents the average distance between the two points of the group. At the same time, a dendrogram tree is built and is populated with all the performed groupings at each step. To retrieve the formed clusters, the dendrogram tree must be cut at a chosen level. This can be done manually, but in order to get an automated process and accurate clusters grouping, we decide to use, instead of the euclidian distance, the Ward's criterion proposed by J. H. Ward in [19] and implemented in the R package, *FactoMineR*. This criterion uses the analysis of variances to form new clusters, and it exposes two accurate metrics which are, the within inertia (i.e. The variance inside a cluster), and the between inertia (i.e. The variance between clusters).

After the formation of the power and the energy states clusters, we take advantage of our energy cartography framework to retrieve which functions induced a power/energy state, and in which cluster this state is located. Following the same operation, and iterating over all the identified states, we succeed to form the desired FSM-based energy model (cf. Figure 9).

V. Conclusion and Future Works

In this paper we present a novel inference method of energy models for embedded peripheral components. We describe its inner working and show a validation example on a real embedded component. The proposed process is built on an energy cartography framework that allows to find energy consumption hotspots in an energy-constrained application, and correct them. The energy model inference method is introduced, to go further and to provide to the developer a means for being proactive during the development phase. This is achieved by targeting the peripheral components which are designated as the main energy consuming parts of an embedded application. Besides, to maximise the comprehension of the developer and make the model inference revaluation easier, we choose to center and focus the energy models directly on the API functions driving the peripheral component activity.

The further works that we want to explore are in line with the same theme. The benchmark source code generation of our process could be improved to find, in a more flexible manner (i.e. with minimal developer interventions), all the dependences between the benchmarked software functions. Moreover, we aim to enhance the profiling granularity of our cartography framework, and so the model inference process, in order to be able to display and handle a *basic block* resolution level rather than a function one. Finally, at long-term, we plan to reduce the gap between hardware-based energy profilers and simulation ones. By taking advantage of our hardware-based energy model inference, we can build an interface that allows to feed an energy simulator with real-world based and flexible embedded component energy models. We strongly believe that this way, a bridge could be built between hardware-based and simulation-based energy profilers.

References

[1] K. M. Z. Shams and M. Ali, "Wireless power transmission to a buried sensor in concrete," *IEEE Sensors Journal*, Dec 2007.

[2] I. E. Agency, "More data, less energy: Making network standby more efficient in billions of connected devices. https://www.iea.org/publications/freepublications/publication/MoreData_LessEnergy.pdf," 2014.

[3] N. Cherifi, G. Grimaud, A. Boe, and T. Vantroys, "Toward energy profiling of connected embedded systems," in *8th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2016, Larnaca, Cyprus, November 21-23, 2016*, 2016.

[4] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale," in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*. New York, NY, USA: ACM, 2007.

[5] 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*. Berkeley, CA, USA: USENIX Association, 2008.

[6] A. Lundgren and L. Frimanson. (2014) I-jet iar systems datasheet. http://datasheet.octopart.com/i-jet-iar-systems-datasheet-14433983.pdf.

[7] Atmel Corporation. (2016) Ateml power debugger. http://www.atmel.com//Images/Atmel-42696-Power-Debugger_UserGuide.pdf.

[8] ARM Limited. (2011) Embedded trace macrocell architecture specification. http://infocenter.arm.com/help/topic/com.arm.doc.ihi0014q/IHI0014Q_etm_architecture_spec.pdf.

[9] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems*.

[10] R. Medhat, S. Ramesh, B. Bonakdarpour, and S. Fischmeister, "A framework for mining hybrid automata from input/output traces," in *2015 International Conference on Embedded Software (EMSOFT)*, 2015.

[11] Clang: A c language family frontend for llvm. https://clang.llvm.org/.

[12] J. Weidendorfer. Kcachegrind, a call graph viewer. https://kcachegrind.github.io/html/home.html.

[13] SEMTECH, "What is lora? http:/www.semtech.com/wireless-rf/internet-of-things/what-is-lora."

[14] Q. Li, M. Martins, O. Gnawali, and R. Fonseca, "On the effectiveness of energy metering on every node," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, May 2013.

[15] N. Cherifi, G. Grimaud, T. Vantroys, and A. Boe, "Energy consumption of networked embedded systems," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, Aug 2015.

[16] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal detection of changepoints with a linear computational cost," *Journal of the American Statistical Association*, 2012.

[17] R. Killick, "Package changepoint: Methods for changepoint detection. https://cran.r-project.org/web/packages/changepoint/changepoint.pdf," 2016.

[18] C. Cecil and J. Bridges, "Hierarchical cluster analysis. http://dx.doi.org/10.2466/pr0.1966.18.3.851," *Psychological Reports*, 1966.

[19] J. H. W. Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, 1963.