Parameter-Aware Energy Models for Embedded-System Peripherals

Birte Friesel, Markus Buschhoff, Olaf Spinczyk
Embedded System Software Group
TU Dortmund
Dortmund, Germany
Email: {birte.friesel,markus.buschhoff,olaf.spinczyk}@tu-dortmund.de

***Abstract*—Energy models support monitoring and prediction of energy use, which is essential for the development and usage of transiently powered systems. However, model generation is a time-consuming and repetitive task. Also, available energy modeling solutions typically assume hardware configurations to be constant, although configuration changes can significantly impact hardware behaviour. Here we present a work-in-progress algorithm for the automatic generation of configuration-aware energy models for system peripherals. We determine the influence of configurable hardware parameters on model attributes and generate functions to describe it. We also propose a new transition energy model to improve energy accounting accuracy without additional overhead. Initial tests show promising results with mean absolute model error less than $1.5\,\%$ for various hardware configurations.**

I. Introduction

The ongoing Internet of Things and Industry 4.0 trends have given rise to a range of embedded systems pervading many aspects of daily life. Each of these needs energy to function, which often cannot be delivered by a wired power supply. Batteries provide an alternative power source, but need to be replaced or recharged when empty. For remote or dense deployments, such as several thousand smart shipping containers in a warehouse, regular battery management is an impractical solution.

Energy harvesting addresses this issue, but poses new challenges: Now, the task is to avoid an unexpected loss of power, which is caused by energy demand exceeding the energy available from the battery and/or via harvesting. Although this can be tested by placing the system in a typical environment and observing its power state, such tests are time-consuming and need to be repeated whenever the environment changes. They also contribute to a "black box" view of the system: the energy use of individual components is unknown.

Here, energy models provide a remedy. For example, a model for a radio transceiver can calculate its energy consumption according to the actual number of bytes to transmit. Models can be used both at design-time (*offline*) to determine the expected system lifetime without lengthy tests, and at runtime (*online*) for energy accounting and adaptive behaviour.

It is useful to embed models at the hardware-software boundary (i.e., by modeling effects of individual driver functions) and use separate models for each system peripheral. Additionally, models should be aware of hardware configuration (*parameters*), which may change at design- or run-time and affect energy consumption and hardware timing.

978-1-5386-4155-2/18/$31.00 ©2018 IEEE

Thus, energy models become independent of high level implementations. This allows developers to choose soft- and hardware best suited for their use case by comparing different combinations in simulation. Additionally, modeling the entire hardware configuration space supports system designers when making trade-offs, e.g. to determine the best compromise between energy use and expected packet loss for a radio chip.

Generating these models, however, is a tedious and error-prone endeavour and cannot be averted by relying on gut instinct or datasheets instead [1], [2]. Existing solutions are often use-case-specific and show significant differences in model quality, as each developer needs to decide on a compromise between accuracy and simplicity [3].

In this paper, we present our work-in-progress framework for automatic generation of configuration-aware energy models for individual embedded-system peripherals using priced timed automata. Although we rely on well-known components, we are not aware of any general-purpose modeling framework providing all of these features. We also present a novel model of transition energy (i.e., the energy used during execution of driver functions), which improves model accuracy without affecting energy accounting performance.

Of course, our approach still has limitations which we expect to address in the future. Most notably, it assumes that hardware/driver interactions are deterministic and correspond to a simplified automata model. Nevertheless, we have already successfully used it for energy accounting on several dozen radio chips in an IoT Smart Warehousing prototype [4].

In the next section, we list a selection of related work. We present our model in sections III and IV, and the generation algorithm in section V. We conclude with preliminary results in section VI and discussion and future work in section VII.

II. Related Work

There is a variety of automatic modeling frameworks for smartphones and laptops mapping power consumption to hardware states, e.g. *DevScope* and *PowerBooter* [5], [6]. However, both rely on manually specified regression formulas and do not account for transition costs, negatively affecting their accuracy [7]. Entirely transition-based models can also

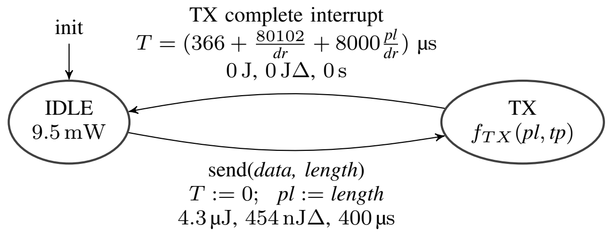

Fig. 1. Partial PTA for a CC1200 radio transceiver with configurable payload length (*pl*), data rate (*dr*) and TX power (*tp*). Timer *T* is reset when calling *send*; the TX state is left once T reaches the expected transmission time specified by f_t. Other states, transitions, and functions omitted for brevity.

be automatically generated, but do not support permanent state changes, such as turning peripherals on or off [8].

Models for WSNs typically focus on the radio module, as it is the most power-hungry component [9]–[11]. They often use automata models, but require manual energy data calculation and do not consider variable hardware parameters.

Symbolic regression can be used to determine basic natural laws [12]. Similarly, parameter influence on model attributes can be automatically determined [7]. However, we are not aware of publications using symbolic regression for energy modeling or general-purpose modeling frameworks using automatic detection of parameter influence.

III. Model

We use parameterized priced timed automata (PTA) to model hardware behaviour. A PTA is a deterministic finite automaton (DFA) with timers, transition durations, and costs. Timers can be reset by transitions and cause other transitions when reaching a certain value; costs are modeled as transition energy and state power.

Each PTA state corresponds to a hardware state, and each transition corresponds to a driver function or hardware interrupt. Thus, our model is placed right at the hardware-software boundary. Fig. 1 shows part of an example energy model for a radio chip. The radio is either IDLE or in TX (transmit) mode. Calling *init* initializes the radio into IDLE mode, whereas *send* initiates a transmission and hence causes a state change to TX. Once transmission is complete, an interrupt indicates that the radio is in IDLE mode again. Radio behaviour can be configured using the parameters *pl*, *dr*, and *tp*, which we will explain in section IV.

For each state, we model the hardware's mean power consumption P. For each transition, we model its duration t (i.e., how long it takes to execute the corresponding driver function) and total energy E consumed by the hardware during this time. We also model relative energy ΔE, which is the total energy adjusted for the energy consumption caused by the preceding (and still active) state q as shown in Fig. 2. We define $\Delta E = E - (P_q \cdot t)$.

As interrupts are instantaneous, we let $t = E = \Delta E = 0$ for interrupt transitions. Instead, we model their timeout T, which is the time spent in the previous state (i.e., the delay until the interrupt fires).

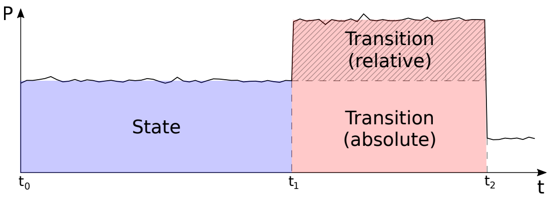

Fig. 2. Relative (hatched area) and absolute (red) transition energy.

The benefits of relative transition energy are twofold. First, online energy accounting needs timestamps to calculate the energy spent in each state. If transition energy is ignored, a single timestamp (marked t_0/t_2 in Fig. 2) at the end of each transition is sufficient, as it marks both the end of the current and the beginning of the next state. It is also sufficient when using relative transition energy, as the state-related (absolute) part of the transition energy has already been subtracted. By contrast, two timestamps (t_0/t_2 and t_1) are needed for absolute transition energy, thus increasing energy accounting overhead.

Second, as driver functions need to communicate with hardware to alter its state, changes caused by driver functions typically happen at the end rather than the beginning of their execution. So, during a transition, the original state is still influencing the hardware power consumption, which in turn influences the transition energy E. As relative energy accounts for this, it better captures the communication and transition overhead. We also expect it to be easier to model.

IV. Model Parameters

We consider two kinds of model parameters: Hardware configuration (global parameters) and driver function arguments (local parameters). Global parameters influence hardware behaviour permanently once set, whereas local function arguments are only relevant for the corresponding transition.

The model shown in Fig. 1 contains the global parameters payload length (*pl*), data rate (*dr*), and TX power (*tp*). Functions may change global parameters. For instance, *pl* is set to the second argument of `send(...)`, and *br* is updated by calling `setBitrate(`*br*`)` (not shown in the example PTA).

For each state and transition, $\vec{p}$ denotes the current parameter values (only global parameters for states, global and local ones for transitions). Whenever a transition changes global parameters, the new values only go into effect after its execution completes – during execution, the old parameters remain valid.

V. Model Generation

Model generation relies on measurements of all hardware states and transitions with as many parameter combinations as feasible. It assumes that measurements consist of runs through the hardware/driver automaton and provide duration, consumed energy, and current parameter values for each state and transition in each run. We developed a highly automated workflow to generate benchmark programs and annotate measurements for this analysis [13].

The model generation process consists of two steps: Identification of relevant parameters for each model attribute (power, duration, (relative) energy, and timeout), and modeling of parameter-dependent attributes using functions. We use the median of all available measurements for parameter-independent model attributes.

We also build a static parameter-aware reference model (a look-up table mapping parameter values to median power/energy/duration of states and transitions for the respective parameter values), as well as additional reference models which ignore certain parameters. These are used to determine the influence of hardware parameters on model attributes.

A. Determining Parameter Influence

We use a simple heuristic: If the mean error of the parameter-aware reference model is significantly lower than the mean error of the reference model which ignores the presence of a single parameter i, this is likely caused by parameter i influencing the modeled hardware attribute. If the mean errors are near equal, we assume that i does not influence the model attribute in question.

Since these models are static, the root mean square error of each model is identical to the standard deviation of the underlying data (using the median as expected value μ). It is therefore sufficient to compare the standard deviations of appropriately partitioned measurement data. For a model attribute X, these are

- $\overline{\sigma_X}$ (the mean standard deviation of measurements used to create the parameter-aware reference model) and
- $\overline{\sigma_{X,i}}$ (same, but ignoring parameter i).

We define a difference of at least 50 % to be significant: If $\overline{\sigma_{X,i}} > 2 \cdot \overline{\sigma_X}$, we assume that parameter i influences model attribute X.

As this is best illustrated in an example, consider a radio module with configurable bitrate and payload length. Measurements are available for bitrates 100, 250 and 1000, and payload length 16 and 32. To determine the influence of bitrate and payload length on the txComplete interrupt timeout, we consider the following three values.

$$\begin{aligned}\overline{\sigma_T} &= \text{mean}\{\sigma_{T,(100,16)}, \sigma_{T,(100,32)}, \sigma_{T,(250,16)}, \dots\}\\ \overline{\sigma_{T,\textit{bitrate}}} &= \text{mean}\{\sigma_{T,(*,16)}, \sigma_{T,(*,32)}\}\\ \overline{\sigma_{T,\textit{length}}} &= \text{mean}\{\sigma_{T,(100,*)}, \sigma_{T,(250,*)}, \sigma_{T,(1000,*)}\}\end{aligned}$$

We proceed by building functions for each model attribute with at least one relevant parameter.

B. Building Model Functions

We first analyze each relevant parameter in isolation and fit a set of domain-specific functions using least squares regression. After having determined the best function to predict the individual influence of each parameter, we combine the functions into a single model function.

To minimize the risk of overfitting, we only consider simple linear, logarithmic, exponential, square, inverse, and square root dependencies, as well as the amount of one and zero bits in the parameter's binary representation.

For each relevant parameter i, we partition the measurements so that, in each set, i is variable and all other parameters are constant. This is the partitioning scheme we already used to calculate $\overline{\sigma_{X,i}}$.

For each partition and each influence type (denoted as g) mentioned above, we fit $a \cdot g(\vec{p}[i]) + b$ using least squares regression and note the root mean square (RMS) error of the fitted function on the training data. We also note the RMS error of a static reference function, which simply uses the median of all data in the respective partition as a model.

The function with the lowest mean RMS over all partitions is the best description of the model influence of parameter i we have. However, if its mean RMS is higher than the mean RMS of the reference function, this indicates that our function set is unable to properly describe the parameter's effects. In this case, we update the model to consider the attribute to be independent of the parameter.

For example, to determine the influence of bitrate on the previously mentioned txComplete timeout, we run a set of regressions on all measurements with a payload length of 16 Bytes, and a separate set of regressions on all measurements with a payload length of 32 Bytes. Assuming the inverse function has the lowest mean RMS on both data sets, we determine an inverse influence of bitrate on transmission time.

This leaves us with a set F of functions, each of which describes the influence of a single parameter. However, we still need to combine them into a single model function.

At this point, we need to consider parameter interdependencies. For instance, transmission time might simply be $\frac{\textit{length}}{\textit{bitrate}}$, or a combination of $\frac{\textit{length}}{\textit{bitrate}}$ and *bitrate* (due to a fixed-length preamble). To avoid having to make assumptions about these interdependencies, we consider all possible combinations of the individual parameter functions as specified in equation 1.

$$g(\vec{p}) = \sum_{F' \in \mathcal{P}(F)} \left(a_{F'} \cdot \prod_{f \in F'} f(\vec{p}) \right) \tag{1}$$

This combined function is in turn fitted on all measurements using least squares regression with $a_{F'}$ as regression variables. We update the model with $g(\vec{p})$ and the optimized regression variables $a_{F'}$. For combinations of non-interdependent parameters, we rely on least squares regression to determine $a_{F'} \approx 0$.

Returning to the txComplete timeout example once more, $F = (f_{\textit{bitrate}}, f_{\textit{length}}) = ((\textit{br}, \textit{len}) \mapsto \textit{br}^{-1}, (\textit{br}, \textit{len}) \mapsto \textit{len})$ is a typical result. We refer back to Fig. 1 for the corresponding model function.

VI. Preliminary Results

So far, we have generated models for four peripherals: Two radio chips (Nordic nRF24L01+ and Texas Instruments CC1200) with configurable transmission power, bitrate and packet length (and other parameters, which we leave out for now), a temperature sensor (LM75B) with configurable alarm thresholds, and a synthetic peripheral with configurable power consumption behaviour.

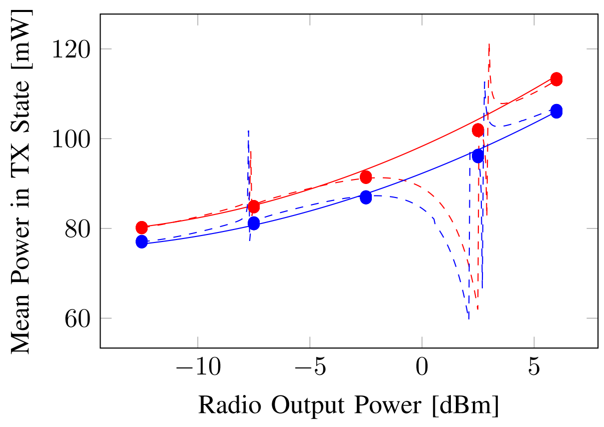

Fig. 3. TX power prediction using our approach (solid) and symbolic regression (dashed) for two distinct bitrates. Dots indicate measurements.

We ran measurements repeatedly to decreases the chance of noisy data affecting the parameter influence detection. To determine model error, we used 200 Monte-Carlo cross-validation runs with data split into $\frac{2}{3}$ training and $\frac{1}{3}$ validation.

Preliminary results show that parameter influence detection and regression modeling work reliably. For CC1200 TX power and txComplete timeout, the model error is 0.7 % and 0.1 %; for the nRF24 module it is 1.2 % and 0.01 %. By comparison, parameter-unaware model error ranges from 16 % to 87 %. As expected, we do not detect parameter influence for LM75B settings. Automatically generated models for the synthetic peripheral consistently reflect the programmed power consumption function. Apart from a correction of CC1200 TX power offsets, all models were created automatically.

We also examined symbolic regression as implemented by the Python3 *gplearn* toolkit. Although it generates functions with a lower mean absolute error than our results, they contain 50 to 100 sub-terms and suffer from overfitting, even when tuning the toolkit's *parsimony coefficient* to favour simple functions over complex ones. Fig. 3 shows a striking example: The symbolic regression model around 2 dBm is not correct.

We assume that this is due to the sparse nature of available parameter values, which in turn is caused by limited measurement time. We expect to follow up on this later.

Finally, relative transition energy proved valuable: Compared to models without transition energy, it decreased model error by 0.1 to 85 % depending on hardware and use case. In terms of model accuracy, the difference between relative and absolute energy is low – we did not find conclusive evidence favouring one over the other. However, given the difference in accounting overhead pointed out in section III, relative transition energy should be used for online energy models.

VII. Conclusion and Future Work

We have presented a work-in-progress algorithm for automatic generation of parameter-aware energy models for embedded system peripherals. It determines parameter influence by comparing standard deviations of appropriately partitioned data sets and generates model functions using two stages of least squares regression on a set of domain-specific function prototypes.

Although we operate under a simplified driver and hardware model without hidden transitions and without functions (such as feature toggles) with more than one possible destination state, preliminary results are promising. We hope to address these limitations in the future.

Acknowledgment

This work was partly supported by the German Research Council (DFG) within the Collaborative Research Centre SFB 876, project A4.

References

[1] N. Zhu and I. O'Connor, "Energy measurements and evaluations on high data rate and ultra low power wsn node," in *10th International Conference on Networking, Sensing and Control (ICNSC)*. IEEE, 2013, pp. 232–236.

[2] D. Harrison, "Busting myths of energy models for wireless sensor networks," *Electronics Letters*, vol. 52, pp. 1412–1414(2), August 2016.

[3] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder, "On the accuracy of software-based energy estimation techniques," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6567, pp. 49–64.

[4] R. Falkenberg, M. Masoudinejad, M. Buschhoff, A. K. Ramachandran Venkatapathy, D. Friesel, M. ten Hompel, O. Spinczyk, and C. Wietfeld, "PhyNetLab: An IoT-based warehouse testbed," in *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2017.

[5] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha, "Devscope: A nonintrusive and online power analysis tool for smartphone hardware components," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 353–362.

[6] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.

[7] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *USENIX Annual Technical Conf*, vol. 20, 2011.

[8] M. B. Kjærgaard and H. Blunck, *Unsupervised Power Profiling for Mobile Devices*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 138–149.

[9] H.-Y. Zhou, D.-Y. Luo, Y. Gao, and D.-C. Zuo, "Modeling of node energy consumption for wireless sensor networks," *Wireless Sensor Network*, vol. 3, no. 01, p. 18, 2011.

[10] O. J. Adinya and L. Daoliang, "Transceiver energy consumption models for the design of low power wireless sensor networks," in *2012 IEEE Student Conference on Research and Development (SCOReD)*, Dec 2012, pp. 193–197.

[11] B. Snajder, V. Jelicic, Z. Kalafatic, and V. Bilas, "Wireless sensor node modelling for energy efficiency analysis in data-intensive periodic monitoring," *Ad Hoc Networks*, vol. 49, pp. 29 – 41, 2016.

[12] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[13] M. Buschhoff, D. Friesel, and O. Spinczyk, "Energy models in the loop," in *Proceedings of the 8th International Sumposium on Internet of Ubiquitous and Pervasive Things (IUPT 2018)*, to appear.