To appear in: Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things (CPS-IoTBench 2019)
Montreal, Canada, 15 April 2019

GenEE: A Benchmark Generator for Static Analysis Tools of Energy-Constrained Cyber-Physical Systems

Christian Eichler, Peter Wägemann, Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

ABSTRACT

To guarantee the safe completion of a specific task in an energy-constrained (i.e., battery-operated, energy-harvesting) cyber-physical system (CPS), information on the task's worst-case energy consumption (WCEC) is necessary. To determine upper bounds on the WCEC, analysis tools conducting static program-code analysis make sound but conservative assumptions on the program's dynamic behavior. When using existing source-code benchmarks, knowledge of their possible program paths and thus their dynamic behaviors are not available. This lack of knowledge leads to missing baselines, which prevents comprehensive evaluations of the accuracy of WCEC analyzers, that is, the difference between the actual WCEC and analyzer's reported upper bound is unknown.

In this paper, we present GenEE, a benchmark generator that enables in-depth evaluations of WCEC analysis tools for energy-constrained CPSs. GenEE combines small program blocks in a way such that all necessary characteristics of the benchmark are available after the generation process together with the benchmark. Since peripherals (e.g., transceivers) are of major importance when analyzing the energy consumption of CPSs, GenEE generates benchmarks that (de-)activate these components while tracking all possible dynamic behavior. Evaluations with our open-source prototype of GenEE on an ARM Cortex-M4 platform show that the generator produces programs with device interactions and known characteristics, most importantly, the program's actual WCEC.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Hardware** → Power and energy.

KEYWORDS

benchmark generation, cyber-physical systems, energy constraints, worst-case energy consumption (WCEC), static code analysis

1 INTRODUCTION

Cyber-physical systems (CPSs) increasingly operate in environments with energy constraints. Battery-operated devices are an example of this type of system and can additionally have mechanisms to harvest energy from their environment (e.g., by using solar panels or piezoelectric harvesters). Two basic operation principles exist for the operation of these systems with intermittent energy supply: Reactive approaches [14, 15] insert checkpoints into the software, at which they resume their activity from after an energy outage. In contrast, proactive approaches do not even start the operation when risking a depletion of the energy storage during this operation [22, 26]. For these proactive approaches, additional a priori knowledge on the worst-case energy consumption (WCEC) for each operation (i.e., task) is necessary, which is determined by a static code analysis before the system's runtime. During runtime, these WCEC values are then combined with the node's current state of charge (i.e., energy level) to decide whether to start a particular operation or wait in deep-sleep mode for enough harvested energy.

Several static program-code analysis tools exist to determine upper bounds on the actual WCEC [8, 11, 23, 25]. Although the bounds reported by the WCEC analyzers are sound and enable reliable forward-looking scheduling, their accuracy (i.e., the difference between actual WCEC and reported upper bound) is unknown when analyzing code from existing benchmark suites, since extracting all relevant information from existing code is not feasible [12, 19]. With our presented tool GenEE[1], we pursue a different approach: Instead of benchmarking analyzers using existing source code, we *generate benchmarks* in a way that the baseline (i.e., the actual WCEC) is known. We achieve this by weaving together small program blocks with known properties to generate a new complex benchmark.

Embedded, energy-constrained systems usually have additional memory constraints due to their limited program storage. Consequently, the use of programs from existing benchmark suites [7, 9] might not be feasible when the executable's size exceeds the memory. Since the GenEE generator is parameterizable with a complexity measure for the generated benchmark, the approach produces suitable evaluation scenarios for a variety of target platforms. In addition to the CPU's power usage, devices (e.g., transceivers, sensors, actuators) play a significant role in CPSs' overall energy consumption. To address this issue, GenEE can handle device information and produces benchmarks that involve switching on/off devices. Specifically, the contributions of this paper are:

(1) GenEE, a benchmark generator that produces complex executable programs that contain interactions with external devices and whose baseline (i.e., actual WCEC) is known
(2) A discussion on GenEE's open-source implementation
(3) Evaluations on an ARM Cortex-M4 validating that GenEE generates complex benchmarks with known baselines

For GenEE, we could benefit from our work on benchmark generation for worst-case execution-time analysis [24]. However, we had to extend the generation approach, since execution time and energy consumption have a non-trivial relationship in CPSs where active devices significantly contribute to the energy consumption, as we also show in our evaluation (see Section 4).

[1] GenEE's name originates from Generating Evaluations for Energy-constrained CPSs.

CPS-IoTBench '19, April 15, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6693-9/19/04…$15.00
https://doi.org/10.1145/3312480.3313170

The remainder of this paper is structured as follows: Section 2 provides necessary background information on WCEC analysis, outlines our system model, and motivates the key problems that arise when benchmarking static WCEC analysis tools. Section 3 presents the GenEE approach of generating benchmarks and gives an overview of important parts of the GenEE implementation, which is evaluated in Section 4. Section 5 discusses related and future work, and Section 6 concludes.

2 PROBLEM STATEMENT

In this section, we first outline our system model (Section 2.1), give general background information on WCEC analysis (Section 2.2), and eventually outline the major problems encountered when benchmarking WCEC analyzers (Section 2.3 & 2.4).

2.1 System Model

GenEE is designed to benchmark WCEC analysis tools that determine WCEC bounds for a single-threaded application running on one cyber-physical node. The node comprises a single-core processor that does not exhibit timing anomalies and has multiple peripherals (i.e., sensors, actuators and transceivers) connected. These devices are software-controlled, that is, the processor switches on/off these devices. The devices can be internal to the microcontroller (e.g., ADC devices) or external (e.g., transceivers). An activated device leads to a constant increase in power consumption of the whole node. An example of our system model is the setup we use in our evaluation in Section 4. Here, we consider an ARM Cortex-M4 processor with 4 KiB instruction cache (2-way set associative, LRU cache-replacement policy) and attach four constant-power consumers (i.e., resistors) that serve as devices, which are switched on/off by the processor with memory-mapped I/O operations. The CPU's microarchitecture is of limited complexity as common for embedded systems, even though our system model allows pipelining and instruction caches.

2.2 Background: WCEC Analysis

We now outline the major problem when benchmarking static WCEC analysis approaches, which is the absence of baselines. The basic working principle of WCEC analyzers for a specific task is divided into two phases: a hardware-independent program-flow analysis and a hardware-cost analysis, which are combined in an integer-linear-program formulation [11, 17]. Solving the formulation eventually yields an upper bound on the actual WCEC. During the program-flow analysis, the program's structure, specifically its control-flow graph, is analyzed and upper bounds on operations, such as loop bounds and recursion depths, are derived. The target-specific hardware-cost analysis determines the energy consumption for each component in the control-flow graph, specifically each basic block, thereby also considering the set of possibly active devices. Since the goal of static WCEC analysis is to determine upper bounds for a specific task or function call, each phase involves pessimistic assumptions for a sound WCEC analysis result.

2.3 Problem # 1: Missing Baselines

Unfortunately, it is unknown *how pessimistic* the analysis result is when running a WCEC analysis on a specific benchmark. That is, the upper bound reported by the WCEC analyzer is possibly one order of magnitude larger than the actual WCEC [20], which makes the analysis result a sound but impractical estimate for the dynamic behavior during the system's runtime. Figure 4 showcases a benchmark's energy-consumption profile (i.e., a histogram of energy consumptions) along with the benchmark's WCEC. Any sound upper bound will be located to the right of the marker *actual WCEC*. In general, it is impossible to determine the actual WCEC from arbitrary programs, as it is infeasible to determine non-trivial properties from programs [19]. Also, an explicit path enumeration of all possible program paths and then using the maximum observed energy consumption is computationally infeasible since too many paths exist even in small programs [12]. Consequently, current WCEC-analyzer evaluations compare their improvement on a relative scale (i.e., with and without improvement) [25]. However, for a comprehensive evaluation on an absolute scale, a ground truth (i.e., the actual WCEC) is inevitable.

Our Approach: Instead of using existing benchmark programs whose baselines (i.e., the actual WCEC) are unknown for comprehensively evaluating WCEC analyzers, we generate benchmarks in a way that allows tracking all relevant knowledge, most importantly, the actual WCEC.

2.4 Problem # 2: Missing Configurability & Resilience of Complexity

Furthermore, when conducting evaluations of WCEC analysis approaches using existing benchmark suites for embedded systems [7, 9], the missing configurability of the benchmarks' complexity is a relevant problem. For example, when trying to run WCEC analyzes on an entire suite comprising several benchmarks, some might not fit into the microcontroller's program memory [20].

The necessity to have small benchmarks fitting into the embedded processor's instruction memory can lead to trivial benchmarks: For example, benchmarks can have a single program path [7, 27], which leads to the histogram over the benchmark's energy consumption with different input values having only a single bar, independent from the input values. These programs are unsuitable for benchmarking WCEC analyzer because the WCEC can trivially be determined by measuring the single path through the program and thus poses no challenge to WCEC analyzers.

The problem of unknown or trivial complexity and the missing configurability of the complexity becomes even more critical when benchmark suites are not publicly available under an open-source license: These suites complicate research by requiring permission for publication and hinder open-source artifact evaluations of analysis approaches [3, 4, 16]. A further issue with the complexity of source-code benchmarks is that it can be significantly reduced by compiler optimizations. For example, for EEMBC®'s ULPMark™ [4] differences of 15 % were reported when changing the degree of optimization but still using the same compiler [18], which means that this suite is not resilient against compiler optimizations. The missing resilience can lead to misinterpretations of evaluation results when the degree of optimization is not reported.

Our Approach: GenEE *is an open-source benchmark generator with a configuration option for the program's complexity that is resilient against compiler optimizations. Furthermore, the generated programs include the use of devices, have an input-dependent control flow, and expose a high variety in their energy consumption at runtime.*

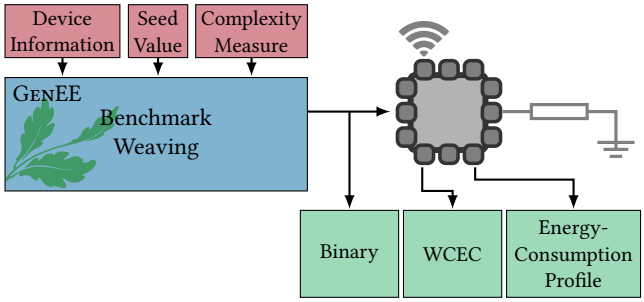

Figure 1: Overview of the GenEE benchmark generator with its input and output parameters

3 THE GENEE BENCHMARK GENERATOR

In this section, we present GenEE, a benchmark generator for the evaluation of static analysis tools of energy-constrained cyber-physical systems, discuss how it solves the problems identified in Section 2, and give insight into the current implementation.

3.1 Solution #1: Generating Benchmarks with Known WCEC

In a nutshell, GenEE generates benchmarks by assembling pseudo-randomly selected pieces of code (i.e., code patterns) that contain interactions with devices, while retaining knowledge about the state of these devices and the generated worst-case code paths (see Section 3.1.2). Based on the knowledge about worst-case code paths, GenEE inserts code patterns that (de-)activate devices (see Section 3.1.3), and eventually derives the benchmark's WCEC via measurement on the target platform (see Section 3.1.4). In the following, we give an overview of GenEE's input and output parameters.

3.1.1 Input and Output Values. Figure 1 illustrates the parameters to generate benchmarks with configurable complexity and a known actual WCEC: As part of the configuration, the user provides information about the devices (e.g., sensors, motor drivers, and generic consumers) connected to the system. Specifically, the device information is a mapping of available devices to the memory addresses, which control general-purpose input/output pins.

In addition to the device information, GenEE receives a *seed value* that has two purposes: First, it enables reproducible benchmarks since GenEE is built in a way that the same seed input value generates the same benchmark. To enable this property, all pseudo-random decisions during the weaving process are based on this input seed value (i.e., a 32-bit integer value). Second, this seed value is used as a *worst-case input value*, a value that leads to the WCEC when executing the generated benchmark with this input. We provide further details on the notion of the *worst-case input value* and the resulting *worst-case path* in Section 3.1.2.

Finally, aside from the device information and the seed value, GenEE's generation algorithm is based on a *complexity measure*: Increasing this integral value allows the tool to generate larger benchmarks, which have a longer execution time and thus can consume more energy and require more program memory. More precisely, the complexity measure approximates the number of (low-level) assembly instructions in the benchmark's executable binary along the worst-case path. To determine the actual WCEC of the benchmark, the generated binary's worst-case path is executed on the hardware of the target platform triggered by the worst-case input value, while the energy consumption is being measured (see Section 3.1.4). Besides the execution with the worst-case input value, the execution with other (randomly selected) input values leads to GenEE's last output, the benchmark's energy-consumption profile.

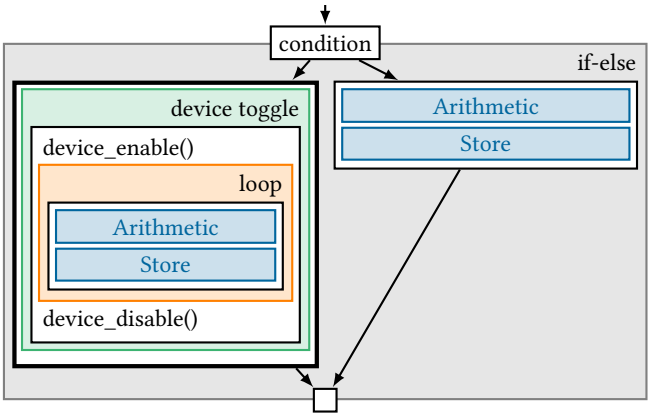

Figure 2: Excerpt of code patterns with device activations woven into a generated benchmark. The code has a worst-case path (left) and a non–worst-case path (right).

3.1.2 Weaving Worst-Case Paths by Overweighting Branches. Since GenEE avoids unnecessarily trivial benchmarks, the generated benchmarks have multiple paths due to input-dependent branches in the code as illustrated in Figure 2. During the generation process, we guarantee that in any possible scenario of the target's microarchitectural state (i.e., caching and pipelining behavior), *the worst-case path leads to both the worst-case execution time (WCET) and the WCEC*. We show later that we can safely activate devices along this dedicated worst-case path (see Section 3.1.3). The main concept to guarantee this worst-case path (even when, for example, only cache misses occur along a non–worst-case path) is *branch overweighting*: Reconsidering a representative of our system model, the ARM Cortex-M4 [10], we identified that, in any case, seven instructions compensate for one (memory) instruction with worst-case pipelining and caching behavior. Thus, we can define a hardware-dependent overweighting factor for the number of generated instructions for each branch of $O_{M4} = 8$ to guarantee that a generated worst-case path leads to the benchmark's WCET. Considering the example in Figure 2, GenEE ensures that the left branch executes O times more instructions than the right (non–worst-case) branch. The overweighting factor O for branches needs to be determined for each target hardware platform independently. Along this dedicated worst-case path with the WCET, we can now safely activate devices in a hierarchical way, as described in the following.

3.1.3 Activating Devices along Worst-Case Paths. GenEE has numerous patterns for arithmetic operations, branches, different shapes of (nested) loops, function calls, and other challenging patterns for static analyzers (e.g., dead-code patterns). For further details on the available patterns and the benchmark-weaving algorithm, we refer to prior work on benchmark generation for timing analysis [24]. Subsequently, we focus on the energy-aware patterns for

(de-)activating devices. Figure 2 illustrates the generation process using an exemplary excerpt with several code patterns: The `then` branch of the top-level `if-else` branch pattern is chosen to be the worst-case path, while the `else` branch is designed to be the non–worst-case path and thereby consumes less time and energy. As a result, the `else` branch does only contain a few instructions and does not contain interactions with devices, while the `then` branch contains many instructions and interactions with devices: The `then` branch contains a device-toggle pattern that enables a single device, executes additional code (e.g., a loop with arithmetic and store instructions), and subsequently disables the device again. The enclosing enable and disable operations guarantee that every enabled device is disabled again, as both the loop and the arithmetic/store instructions resemble structured control flow. In the current approach, GenEE has two assumptions on device activation in order to guarantee having the actual WCEC along the worst-case path: First, device (de-)activations are hierarchical. That is, activated devices along a program path are deactivated hierarchically along the same path. Second, GenEE's weaving algorithm inserts device activations *only along the worst-case path*. Consequently, by design, the worst-case path executes all its phases with the maximum possible power along this program path, which results in the worst-case path – the path that leads to the WCET – also leads to the benchmark's actual WCEC.

3.1.4 Determining the Actual WCEC. At the end of the generation process, GenEE provides both the benchmark's binary and, by design, the input value triggering the WCEC. Thus, we concretely execute the benchmark with this input on the target hardware platform while measuring the energy consumption with an accurate measurement unit. This measurement-based approach follows the rationale that the most-accurate energy-consumption model is the hardware itself. Additional to the execution with the worst-case input value, we execute the benchmark with (randomly selected) inputs to obtain the benchmark's energy-consumption profile.

3.2 Solution #2: Configurable & Resilient Complexity

To provide a way to configure the complexity and composition of the generated benchmark, GenEE allows the user to specify both a list of patterns to be used, as well as a measure for the number of generated assembly instructions along the worst-case path. By increasing or decreasing the number of generated instructions, the benchmarks can be adapted to various limitations of the hardware platform, such as small program memory sizes. The default configuration of GenEE generates benchmarks with input-dependent control flow (see Section 3.1) and uses all available devices, resulting in a high variety in the benchmark's dynamic power and energy consumption. This default can be adapted to particular hardware features, such as the availability of a floating-point unit or external devices by modifying the list of patterns.

In order to support a variety of different configurable devices, GenEE uses system calls provided by an underlying operating system for switching on and off devices. System calls not only enable easy integration of new devices by using external libraries, but also provide the possibility of running GenEE's benchmarks in an unprivileged execution mode by switching to a privileged mode for interacting with the device.

Implementation. The GenEE benchmark generator is implemented as a tool for the LLVM compiler infrastructure [13] and operates on LLVM's intermediate representation (LLVM IR), a platform-independent representation closely related to machine code. The reason for choosing this abstraction level in our implementation is to tackle the challenge of compiler optimizations, compared to using source-code benchmarks written in a high-level programming language: Source-code benchmarks face the problem of being decisively changed by compiler optimizations (e.g., function inlining, constant folding, loop unrolling). The influences of these compilation steps on the final evaluation results can be significant [18, 27] and potentially cause misleading conclusions. Since GenEE generates code on a low-level representation and, additionally, uses code patterns that already resemble optimized code, the generated benchmarks are less affected by compiler optimizations, as we have shown in a prior evaluation on the generator's resilience against compiler optimizations [24].

GenEE's implementation with awareness of devices and energy consumption is based on our previous work on worst-case execution-time analysis [24], but as we show in the evaluation of GenEE (see Section 4), energy consumption and execution time do not have a linear relationship in device-driven CPSs. For GenEE, we modified our framework and focused on comprehensive WCEC-analyzer evaluations of the whole CPS, which includes the energy consumed by both the CPU and its devices. To provide maximum flexibility for future research and modifications, we released GenEE's source code under an open-source license[2].

4 EVALUATION

In this Section, we evaluate the impact of switching on/off devices on the energy-consumption profile of the benchmarks generated by GenEE, validate our approach presented in Section 3, and substantiate the need for benchmarks dedicated to device-driven CPSs.

4.1 Experimental Setup

Our evaluations are conducted using the Infineon XMC4500 development board with an ARM Cortex-M4 processor [10]. The 32-bit processor uses a 3-stage pipeline, contains 1024 KiB on-chip flash memory with 4 KiB instruction cache (2-way set associative, LRU cache-replacement policy), and runs at a frequency of 30 MHz.

As consumer load, we connect four 56 Ω resistors to different GPIO pins, acting as devices with constant power consumption. To reduce the load on the GPIO pins, we power these devices at 3.3 V using a driver circuit based on transistors. Following Ohm's law, the consumption of the devices is approximately 59 mA, which is in the same operation range as external WiFi transceivers [21].

For energy measurement, we measure the voltage drop over a 0.51 Ω shunt resistor and additionally over the whole system using a Tektronix MSO4034 oscilloscope with a sampling rate of 2.5 GS/s. The measurement and evaluation process is based on PyVISA[3] and integrated into our automated evaluation framework Aladdin [6].

[2] GenEE's source code is publicly available at https://gitlab.cs.fau.de/gene
[3] https://github.com/pyvisa/pyvisa

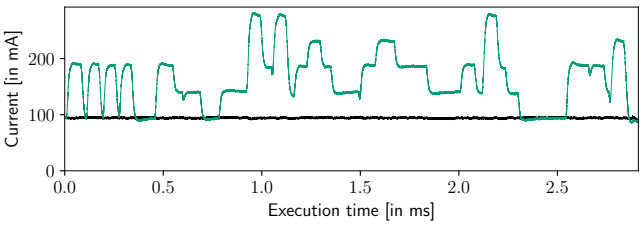

Figure 3: Current consumption over time for the worst-case path leading to the actual WCEC with (green, above) and without (black, below) devices connected to the system

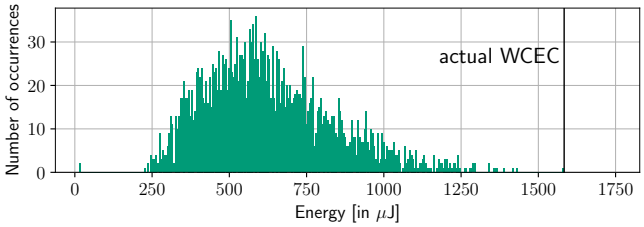

Figure 4: Energy-consumption profile of 2 500 input values and actual worst-case energy consumption (WCEC)

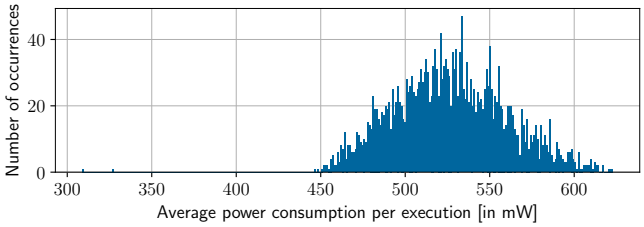

Figure 5: Power-consumption profile of 2 500 input values

The following evaluations are based on one benchmark using GenEE's default parameter set and a complexity measure of 25 000 (i.e., estimate for assembly instructions along the worst-case path), which produces a benchmark in the range of milliseconds.

4.2 Assessing the Effects of Devices

As a first step, we evaluate the effects of enabling and disabling devices on the current flowing through the whole CPS. We expect to see variations of the current over time, depending on whether and how many devices are enabled. To rule out the presence of other effects, we repeat the measurement for the same benchmark while having all devices disconnected.

For this evaluation, we measure and analyze the current over time for the execution of the worst-case path, which results in the WCEC. Figure 3 shows the result of this measurement: The nearly constant black plot (below the green plot with high variability) illustrates the current flowing through the whole system while having the devices disconnected. Without devices, the system shows a current of 95 mA, irrespective of the currently executed instructions. The green plot, above the black one, shows the current observed during the execution of the same execution path through the same benchmark, but with devices connected: Over time, the current varies in the range from 84 mA to 282 mA, rises once a device is enabled, and falls when it is disabled. Subsequent increases of the current (up to a maximum of around 282 mA) demonstrate GenEE's ability to not only enable one device at a time but also several devices in a hierarchical manner. To summarize, GenEE's benchmarks activate devices on a fine-grained level (i.e., in the range of ten microseconds).

4.3 Validating the Generated WCEC

In the second step, we conduct an evaluation to verify that the WCEC obtained during GenEE's generation process is not exceeded by any execution and thereby show the validity of our approach. For this purpose, we generate a benchmark handling four external devices, determine its WCEC, and measure the energy consumption of 2 500 different input values. During the six hours the evaluation takes, we compare each measured energy consumption against the actual WCEC and detect no larger energy consumption. The validation that the WCET is not exceeded along the worst-case path [24] and the fact that devices are only activated along the worst-case path underlines that GenEE is correct by construction and that the generated benchmark's actual WCEC is not exceeded.

Figure 4 shows the energy-consumption profile along with the determined actual WCEC (1 584 µJ). The profile shows a wide range of different values for the energy consumption of the generated benchmark, ranging from 15 µJ to 1 584 µJ. This variety indicates a multitude of different execution paths with varying device interactions. The WCEC determined during generation marks the upper limit of the measured energy consumptions, underlining that the WCEC obtained from the generator is valid and thereby confirms the applicability of our approach.

4.4 Energy vs. Time

Our last evaluation is dedicated to illustrating the need for a benchmark generator for the evaluation of energy-analysis tools by demonstrating that there is no linear relationship between execution time and energy consumption for device-driven CPSs. For this purpose, we determine the average power consumption (i.e., energy consumption divided by execution time) for each of the 2 500 executions shown in Figure 4. Figure 5 shows a histogram of average power consumptions for the benchmark used during the whole evaluation: The values range from a minimum average power consumption of 309 mW to a maximum of 623 mW, indicating a variation of the power consumption depending on the execution path taken. Static WCEC tools need to capture this varying power consumption due to devices to determine precise WCEC bounds.

To summarize the evaluation results, GenEE generates benchmarks with a known actual WCEC enabling comprehensive evaluations of WCEC analyzers. The generated benchmarks are input-dependent and exhibit high variability in dynamic power as well as overall energy consumption. By generating benchmarks close to machine-code level, we achieve fine-grained device activations.

5 RELATED & FUTURE WORK

To the best of our knowledge, GenEE is the first benchmark generator for the evaluation of static analysis approaches for energy-constrained CPSs, where knowledge about the actual WCEC is necessary. Several benchmark suites exist for (energy-constrained)

cyber-physical systems [1, 7, 9, 16], however, determining the actual WCEC from existing source code is not possible [12, 19], and we argue that the only solution to this problem is a generative approach, such as GenEE.

Although our evaluation shows that GenEE produces benchmarks with configurable complexity and known WCEC, there are several directions for future work: GenEE currently generates single-threaded benchmarks that involve switching on/off devices. However, even small CPSs involve several tasks, for example, scheduled with a fixed-priority scheme. We already have an extension in our framework for tasksets [5], but the energy-aware integration of external interrupts from devices is part of future work.

The original focus of GenEE is benchmarking WCEC analyzers. Nevertheless, the framework is also useful for benchmarking the *performance* of embedded processing nodes and benchmarking *communication protocols* between nodes. It is possible to generate phases that stress specific components of the microcontroller, for example, phases with CPU-bound or memory-bound behavior, which are known to mark peaks in power consumption [2, 28]. Furthermore, since GenEE already has the possibility to generate system calls that switch on/off devices, we plan to extend the framework to enter and leave low-power modes of the microcontroller generically. That way, we can achieve a behavior similar to ULPMark™ from EEMBC® [4]. However, in contrast to ULPMark™, GenEE is available free of charge and does not involve any restrictive licenses. Moreover, the structure of ULPMark™'s source-code benchmarks can be decisively changed by compiler optimizations. As a consequence, the energy-consumption results when executing the same benchmark program with the same compiler but different degrees of optimization can differ by 15 % [18]. In contrast, GenEE generates benchmarks on a low abstraction level close to machine code (i.e., LLVM IR), thereby reducing the influence of compiler optimizations and enabling resilience against compiler optimizations [24].

In addition to benchmarking single nodes in a cyber-physical system, an interesting aspect of future work is the integration of communication protocols connecting multiple nodes. GenEE currently generates benchmarks for a single node with peripherals. But extending GenEE to also generate functions for communication is conceptually possible since switching on/off devices on a fine-grained level is already supported. For this future work, existing communication-oriented CPS benchmarks are of significant importance [1, 3, 4] to achieve realistic evaluation scenarios.

6 CONCLUSION

Benchmarking WCEC analyzers for CPS is a challenging task: The missing configurability of benchmark complexities, its lacking resilience against compiler optimizations, and especially the absence of baselines in existing benchmark suites hinders comprehensive evaluations and thus complicates further research in this direction.

With GenEE, we built a benchmark generator solving these problems. GenEE assembles small code patterns with known properties on a low abstraction level in order to generate a complex benchmark, whose baseline (i.e., the actual WCEC) is known and that exhibits a complex energy-consumption profile. In contrast to commercial low-power benchmark suites for cyber-physical systems being available under restrictive licenses, GenEE is open source to enable long-term research in this direction.

The source code of GenEE is publicly available under the open-source GPLv3 license: https://gitlab.cs.fau.de/gene

Acknowledgments: This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/9-2, no. SCHR 603/13-1, no. SCHR 603/14-2, no. SCHR 603/15-1, and the Bavarian Ministry of State for Economics, Traffic and Technology under the (EU EFRE funds) grant no. 0704/883 25.

REFERENCES

[1] C. A. Boano, S. Duquennoy, A. Förster, O. Gnawali, R. Jacob, H.-S. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G. P. Picco, X. Vilajosana, T. Watteyne, and M. Zimmerling. 2018. IoTBench: Towards a Benchmark for Low-power Wireless Networking. In *Proc. of CPSBench '18.*
[2] A. Carroll and G. Heiser. 2010. An analysis of power consumption in a smartphone. In *Proc. of ATC '10.*
[3] EEMBC®. 2019. IoTMark™-BLE - An EEMBC Benchmark. eembc.org/iotmark/.
[4] EEMBC®. 2019. ULPMark™- An EEMBC Benchmark. eembc.org/ulpmark/.
[5] C. Eichler, T. Distler, P. Ulbrich, P. Wägemann, and W. Schröder-Preikschat. 2018. TASKers: A Whole-System Generator for Benchmarking Real-Time-System Analyses. In *Proc. of WCET '18.*
[6] C. Eichler, P. Wägemann, T. Distler, and W. Schröder-Preikschat. 2017. Demo Abstract: Tooling Support for Benchmarking Timing Analysis. In *Proc. of RTAS '17.*
[7] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener. 2016. TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research. In *Proc. of WCET '16.*
[8] K. Georgiou, S. Kerrison, Z. Chamski, and K. Eder. 2017. Energy Transparency for Deeply Embedded Programs. *ACM TACO* 14 (2017).
[9] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proc. of WWC '01.*
[10] Infineon Technologies AG. 2014. Evaluation Board XMC 4500 Relax Kit & XMC 4500 Relax Lite Kit.
[11] R. Jayaseelan, T. Mitra, and X. Li. 2006. Estimating the Worst-Case Energy Consumption of Embedded Software. In *Proc. of RTAS '06.*
[12] J. Knoop, L. Kovács, and J. Zwirchmayr. 2013. WCET Squeezing: On-demand Feasibility Refinement for Proven Precise WCET-bounds. In *Proc. of RTNS '13.*
[13] C. Lattner and V. Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. of CGO '04.*
[14] B. Lucia and B. Ransford. 2015. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *Proc. of PLDI '15.*
[15] K. Maeng and B. Lucia. 2018. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *Proc. of OSDI '18.*
[16] J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On. 2009. A Benchmark Characterization of the EEMBC Benchmark Suite. *IEEE Micro* 29, 5 (2009).
[17] P. Puschner and A. Schedl. 1997. Computing Maximum Task Execution Times: A Graph-Based Approach. *Real-Time Systems* 13 (1997).
[18] M. Redon. 2017. *Strategies for Choosing the Appropriate Microcontroller when Developing Ultra Low Power Systems.* Technical Report. Analog Devices.
[19] H. G. Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Trans. of the American Mathematical Society* (1953).
[20] V. Sieh, R. Burlacu, T. Hönig, H. Janker, P. Raffeck, P. Wägemann, and W. Schröder-Preikschat. 2017. An End-to-End Toolchain: From Automated Cost Modeling to Static WCET and WCEC Analysis. In *Proc. of ISORC '17.*
[21] Espressif Systems. 2018. ESP8266EX Datasheet Version 6.0.
[22] M. Völp, M. Hähnel, and A. Lackorzynski. 2014. Has Energy Surpassed Timeliness? Scheduling Energy-Constrained Mixed-Criticality Systems. In *Proc. of RTAS '14.*
[23] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat. 2018. Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems. In *Proc. of ECRTS '18.*
[24] P. Wägemann, T. Distler, C. Eichler, and W. Schröder-Preikschat. 2017. Benchmark Generation for Timing Analysis. In *Proc. of RTAS '17.*
[25] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat. 2015. Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. In *Proc. of ECRTS '15.*
[26] P. Wägemann, T. Distler, H. Janker, P. Raffeck, V. Sieh, and W. Schröder-Preikschat. 2017. Operating Energy-Neutral Real-Time Systems. *ACM TECS* 17, 1 (2017).
[27] P. Wägemann, T. Distler, P. Raffeck, and W. Schröder-Preikschat. 2016. Towards Code Metrics for Benchmarking Timing Analysis. In *Proc. of RTSS WiP '16.*
[28] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. 2012. AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring. In *Proc. of ATC '12.*