# Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems

**Peter Wägemann**
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

**Christian Dietrich**
Leibniz Universität Hannover (LUH), Germany

**Tobias Distler**
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

**Peter Ulbrich**
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

**Wolfgang Schröder-Preikschat**
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

## Abstract

Although internal devices (e.g., memory, timers) and external devices (e.g., transceivers, sensors) significantly contribute to the energy consumption of an embedded real-time system, their impact on the worst-case response energy consumption (WCRE) of tasks is usually not adequately taken into account. Most WCRE analysis techniques, for example, only focus on the processor and therefore do not consider the energy consumption of other hardware units. Apart from that, the typical approach for dealing with devices is to assume that all of them are always activated, which leads to high WCRE overestimations in the general case where a system switches off the devices that are currently not needed in order to minimize energy consumption.

In this paper, we present SysWCEC, an approach that addresses these problems by enabling static WCRE analysis for entire real-time systems, including internal as well as external devices. For this purpose, SysWCEC introduces a novel abstraction, the power-state–transition graph, which contains information about the worst-case energy consumption of all possible execution paths. To construct the graph, SysWCEC decomposes the analyzed real-time system into blocks during which the set of active devices in the system does not change and is consequently able to precisely handle devices being dynamically activated or deactivated.

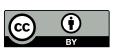
**2012 ACM Subject Classification** Computer systems organization → Real-time systems

**Keywords and phrases** energy-constrained real-time systems, worst-case energy consumption (WCEC), worst-case response energy consumption (WCRE), static whole-system analysis

**Digital Object Identifier** 10.4230/LIPIcs.ECRTS.2018.24

**Supplement Material** ECRTS Artifact Evaluation approved artifact available at `https://dx.doi.org/10.4230/DARTS.4.2.7`, Source code of *SysWCEC* available at `https://gitlab.cs.fau.de/syswcec`

**Acknowledgements** This work is supported by the German Research Foundation (DFG), in part by Research Unit FOR 1508 under grant no. SCHR 603/14-2 (BATS), Research Grant no. SCHR 603/13-1 (Power-Aware Critical Sections), the CRC/TRR 89 Project C1 (Invasive Computing), and the Bavarian Ministry of State for Economics under grant no. 0704/883 25.





30th Euromicro Conference on Real-Time Systems (ECRTS 2018).
Editor: Sebastian Altmeyer; Article No. 24; pp. 24:1–24:25

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We thank Simon Schuster for the insightful discussions about Platin, Niklas Duda and Stefan Erhardt for the support with the energy-measurement setups, and Robert Burlacu for the help with improving the ILP solving.

## 1 Introduction

Energy-constrained real-time systems must not only ensure that tasks meet their timing deadlines, but also that there is enough energy to execute the tasks to completion [70, 72, 73]. Therefore, it is essential for energy-aware schedulers to consider both an upper bound for the execution time of a task as well as its worst-case response energy consumption (WCRE), that is, the maximum amount of energy required by the system to fully execute the task once it has been started [26, 70]. For systems where a task can be interrupted or preempted by other tasks with higher priorities, this means that a task's WCRE covers both the worst-case energy consumption (WCEC) of the task itself as well as the WCECs of all interrupt service routines and tasks that might be executed while the task is running[1].

Obtaining worst-case execution times can be regarded a solved problem for embedded, single-threaded real-time systems [5, 74] with multiple timing-analysis tools being commercially available [2, 53, 66]. Determining upper bounds for energy consumption, on the other hand, is still an open issue for systems in which devices and peripherals contribute to power consumption. Although energy profilers exist that are able to measure the energy consumption of systems including devices [62], so far there is no analyzer that provides reliable upper bounds for an entire system. Existing approaches to determine worst-case energy consumptions so far are usually limited to an analysis of the influence of a system's processor [35, 71]. Unfortunately, this strategy provides only a partial view of the problem, because in many embedded systems the processor is just one of several energy consumers besides internal devices (e.g., memory, timers) and external devices (e.g., peripherals such as WiFi transceivers, analog-to-digital converters, accelerometers, or LEDs). As illustrated in Table 1 by example of the NXP KL46z platform [23, 24] (ARM Cortex-M0+), a typical hardware for a small battery-operated real-time system, these devices in general significantly contribute to the system's overall power consumption. In some cases, for example, transceivers or LEDs, the power consumption of the device even exceeds the power consumption of the processor. Consequently, in order to obtain reliable results, it is crucial to take the impact of devices into account when analyzing a system's energy consumption.

The common approach to prevent WCRE underestimations for systems with devices is to assume that all the devices are active the entire time [43] and to include their combined power consumption into the analysis. Although this technique has the benefit of being sound, it also comes with the major drawback of usually leading to significant overestimations. These overestimations are caused by the fact that in many systems in practice devices and peripherals are disabled most of the time in order to save energy, and only temporarily switched on while their services are actually required, for example, to broadcast a message via a transceiver. As a consequence, WCRE analyses that assume all the devices to be always on often provide energy-consumption estimates that are much higher than the actual WCECs, which possibly leads to systems stopping execution unnecessarily early or to the system's lifetime being greatly underestimated by the pessimism of the analysis.

[1] We use the terms WCEC and WCRE analogous to timing analysis where the worst-case execution time (WCET) refers to a task in isolation and the worst-case response time (WCRT) to the timespan from the start of a task until its completion, including all possible interferences (e.g., preemptions).

**Table 1** Power consumers in energy-constrained systems [23, 25, 67].

| Hardware Unit | Power Consumption [@3.3V] |
|---|---|
| MCU (run mode) | 5.6 mA |
| MCU (low-power run mode) | 0.7 mA |
| MCU (stop mode) | 0.3 mA |
| Accelerometer | 1.7 mA |
| Analog–to-digital converter | 0.4 mA |
| External memory (FRAM) | 0.2 mA |
| LED | 4.6 mA |
| WiFi transceiver | 87.6 mA |

The main reason why existing approaches deal with the impact of devices at a coarse-grained level is that WCRE analysis is inherently difficult in the context of devices that are dynamically switched on and off. Precisely determining the WCRE of a task requires knowledge about the entire system, including the WCECs of interrupt service routines and tasks with higher priorities. Additionally, the (de-)activation of devices, especially the activation of timers for running the CPU at a higher frequency, causes significant latencies that lead to energy-consumption penalties [8, 9]. In the absence of devices, obtaining the necessary WCECs is straightforward as the individual WCECs of all relevant routines and tasks can be analyzed in isolation from each other. However, in systems with devices this is not possible because, as we will show in detail, the WCEC of a task not only depends on the work performed by the task itself but also on the actions (i.e., device activations/deactivations) taken by other tasks, in some cases even tasks with lower priorities.

This paper presents *SysWCEC*, a static analysis approach that addresses the problem of determining WCREs in real-time systems with devices by taking the entire system into account. For this analysis purpose, *SysWCEC* first constructs and then leverages a novel data structure called the *power-state–transition graph*, which contains knowledge about the worst-case energy consumption of the analyzed system for all possible execution paths.

To construct the power-state–transition graph, *SysWCEC* in a first step searches for locations in the system code at which the power state of a device is changed and then logically decomposes the code into blocks of instructions during which the power states of devices remain constant. Next, *SysWCEC* identifies all possible interactions between the discovered blocks and combines this knowledge with additional information about the blocks' power consumptions and worst-case execution times. In the last step, this enables *SysWCEC* to determine all possible states the system might be in while it is running; in addition, for each of these states, this allows *SysWCEC* to compute the maximum amount of energy the system will consume while executing the instruction block associated with the state.

Decomposing the overall system into smaller blocks with constant device power states offers the key benefit of allowing us to perform large parts of the WCRE analysis without having to deal with varying power consumption while still being able to account for dynamic device (de-)activations. Apart from that, the context-sensitive analysis of both synchronous task interactions as well as asynchronous interrupts enables us to individually determine the WCEC of each task even for systems in which a task's WCEC cannot be analyzed in isolation as it might depend on the behavior of other tasks.

The *SysWCEC* approach presented in this paper borrows ideas from previous work on whole-system response-time analysis of fixed-priority real-time systems [19]. However, although lessons learned from timing analysis are helpful for energy-consumption analysis, in

general, it is not possible to directly reuse existing techniques due to substantial differences between both domains. As we will show in this paper, energy-consumption analysis requires a more extensive system analysis that considers tasks of all priorities, addresses device (de-)activation penalties, and tracks power states of devices across all possible system states. Solving these problems forced us to develop a new approach to structuring real-time systems and their devices' power consumption to determine safe and accurate WCREs.

In summary, this paper makes the following contributions: (1) It presents our whole-system approach to WCRE analysis for real-time systems with internal and external devices and provides details on *SysWCEC*'s central data structure: the power-state–transition graph. (2) It gives insights into the open-source *SysWCEC* prototype, which supports the fully-automatic processing of OSEK-compliant (i.e., ECC1 [49]) real-time systems. (3) It discusses our evaluation of two different hardware platforms, which shows that *SysWCEC* is able to significantly reduce WCRE overestimations compared with the approach of assuming all devices to be always active.

## 2 Problem Statement

In this section, we first provide details on *SysWCEC*'s underlying system model and then discuss two open challenges that so far remain with regard to WCRE analysis: (1) precisely accounting for the fact that devices and peripherals in practical systems are dynamically switched on and off, and (2) determining task WCECs that depend on overall system state.

### 2.1 Hard- & Software System Model

*SysWCEC* targets embedded real-time platforms for which energy is a scarce resource. In such systems, the processor usually has a single processing core, a small predictable instruction cache, no data cache, and few pipeline stages [3, 4]. Due to the limited complexity, determining worst-case execution times based on the cycle costs of instructions in isolation is a feasible approach and achieves low overestimations [61]. Typically, the software running on such platforms consists of less than a dozen tasks that have fixed priorities and possibly depend on each other. A task is either synchronously activated by another task or a periodic alarm, or asynchronously activated as the result of a hardware interrupt. Interrupts always preempt the task currently running and can be released with a minimum inter-arrival time $p_i$, that is, there is an upper bound for the frequency with which interrupts are triggered.

Apart from the processor, systems in the targeted domain typically have numerous internal and external devices that significantly contribute to overall power consumption. While simple devices can only assume two different power states (i.e., `on` and `off`), more complex devices may comprise additional power modes, for example, to offer different tradeoffs between performance and power consumption. In each power mode, a device has a (mode-specific) maximum power consumption. Consequently, an upper energy bound $E$ for an interference-free code sequence can be determined based on the worst-case execution time $WCET$ of the code using $E = WCET \cdot P_{max}$, with $P_{max}$ being the total maximum power consumption of all hardware components in their current power modes. How to create a sound worst-case energy model to compute $P_{max}$ is outside the scope of this paper. In general, the necessary information can be obtained from hardware analyses [50] and/or documentation [24].

In the targeted systems, transitions between power modes are initiated by the operating system as the result of a system call invoked by a task or an interrupt service routine; in this paper, we refer to such calls as *device system calls*, or *device syscalls* for short. Once invoked, a device syscall only returns after the requested power-mode switch is complete. All

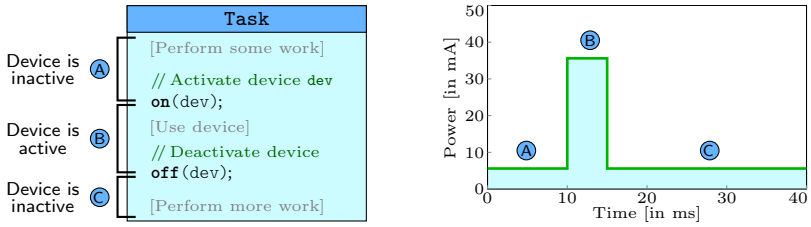



**Figure 1** Effect of a temporarily activated device on power consumption.

dynamic power management is explicitly controlled by the application via device system calls and passes the operating-system kernel. That is, as it is common for embedded platforms, the hardware does not initiate power-mode changes itself. To determine upper bounds for energy consumption for the whole system, the analysis cannot be limited to the user level but requires a whole-system view that includes both user application and the operating system.

## 2.2 Challenge #1: Modeling Temporarily Activated Devices

With devices being a decisive factor in an embedded system's power consumption, energy-constrained systems are usually designed to only keep a device active as long as its services are actually required. While on the one hand, this approach enables such systems to greatly extend their lifetimes, on the other hand, it also complicates static energy-consumption analysis because the power consumption of the system no longer only depends on the instructions executed but instead is also affected by the set of devices currently active. Figure 1 illustrates this problem for a task consisting of three parts: a first part in which the task performs some processing without using any devices (Part Ⓐ), a second portion in which the task temporarily activates and accesses a device (Part Ⓑ), and a third part that continues processing (Part Ⓒ). In the absence of devices, the worst-case energy consumption of a task can be statically determined based on the energy costs of individual instructions [35, 71]. However, for the example task this is not possible because the system calls to activate and deactivate the device, despite consuming only a small amount of energy themselves, significantly change overall power consumption due to modifying system state. Consequently, the energy consumption of the system for Part Ⓑ to a large extent is not a result of the processor executing certain instructions but of the fact that the device is active during this period of time. Our example therefore shows that for systems with devices it is insufficient to limit the energy-consumption analysis to the instructions executed and it explains why techniques that focus on the processor [35, 71] usually underestimate the worst-case energy consumption. Furthermore, many real-time scheduling approaches using DVFS disregard the latencies to switch on the timer devices for running on a higher frequency [8], which can be in the range of one millisecond [9].

In the context of worst-case energy-consumption analysis, the common approach to deal with devices is to prevent underestimations by modeling all devices in a system to be always on [43]. As our example in Figure 1 illustrates, this generally leads to significant overestimations due to assuming an increased power consumption that most of the time (e.g., during Parts Ⓐ and Ⓒ) is much higher than the consumption actually possible in practice.

*Our Approach:* To properly account for the energy consumption of internal and external devices, we identify parts of the system code during which the set of active devices does not change, starting a new part whenever a device is activated or deactivated. Performing our analysis at this granularity level allows us to minimize analysis complexity without losing the ability to model the impact of temporarily activated devices.

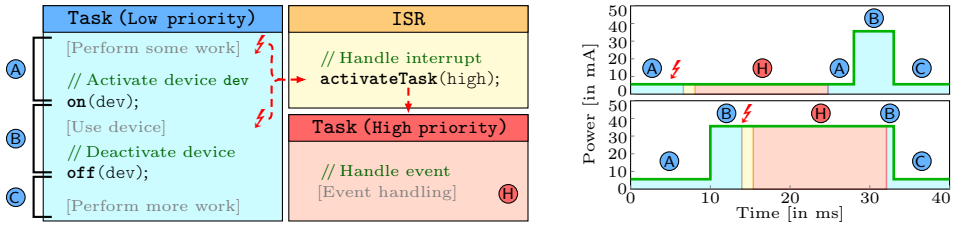



**Figure 2** A task's WCEC may depend on another lower-priority task.

## 2.3 Challenge #2: System-State – dependent Task WCECs

While worst-case energy consumption (WCEC) analysis for systems with devices and peripherals is already challenging for a single task, the problem becomes even more difficult when entire task sets are involved. To illustrate this, we extend the example of Section 2.2: As depicted in Figure 2, we now assume that the task can be interrupted and that, as reaction to an interrupt, the interrupt service routine (ISR) activates a task with a higher priority, which executes a Part Ⓗ. In the following, we focus on discussing the difficulties associated with determining the WCEC of this higher-priority task.

As illustrated by the graphs in Figure 2, the power consumption of the high-priority task depends on the point in time at which the interrupt is triggered. That is, if the interrupt arrives while the system executes Part Ⓑ of the low-priority task, the high-priority task consumes much more power compared to the case in which the interrupt arrives during Part Ⓐ when the device is still inactive. However, in both cases the high-priority task executes exactly the same instructions (i.e., Part Ⓗ), which shows that the WCEC of the task does not only depend on the actions taken by the task itself but also on the state the system is in when the task starts executing, which is a result of previous actions taken by other tasks (i.e., device activations and deactivations). As shown by the example, this may even include actions taken by other tasks with lower priorities.

The fact that in systems with devices the worst-case costs of a task may depend on other tasks constitutes a major difference between timing analysis and energy-consumption analysis: To determine the worst-case execution time of the high-priority task, it is sufficient to analyze the task in isolation. Consequently, to compute an upper bound for the response time of a task (i.e., WCET plus potential interferences), an analysis only needs to consider the task itself as well as all interrupt handlers and tasks that might be executed while the task is running [5, 68]. In contrast, an analysis of worst-case (response) energy consumptions requires a comprehensive analysis of the entire task set, which means that existing timing-analysis approaches cannot be directly applied to analyze energy consumption in systems with devices.

*Our Approach:* Using a context-sensitive analysis that covers both synchronous task activations and asynchronous interrupts, we identify all possible states the analyzed system might reach during execution. By also analyzing the transitions between these states, we are able to determine the set of active devices for each of the states, which consequently allows us to precisely compute the WCECs of individual tasks.

# 3 The SysWCEC Approach

In the following, we present *SysWCEC*, a whole-system analysis approach to worst-case response energy consumption (WCRE). We tackle the challenges mentioned in Section 2 and tighten WCRE estimates by eliminating infeasible combinations of system-wide execution paths and energy states and thus abandon the all-always-on approach for external devices.

## Overview

In a nutshell, *SysWCEC* leverages knowledge about device usage and operating-system semantics for a context-sensitive system-wide control-flow analysis that, in particular, incorporates state-dependent power consumptions. Conceptually, *SysWCEC* is based on the inference of the system's possible dynamic behavior and consequently all states the system might take during execution. By that, we mean the *system state* consisting of (a) active tasks and their priorities, (b) interrupt masks, (c) resource occupancy and ceiling priorities, and (d) power states of external devices. This knowledge allows for fine-grained modeling of extrinsic energy costs that can neither be attributed to individual instructions nor tasks but must be assessed in a system context, which is a fundamental advance of traditional techniques.

To infer the system states and thus disclose all energy-relevant interactions, our whole-system static analysis requires the following three steps, which we briefly outline next before immersing in further details in Sections 3.1 to 3.3.

1. **Abstraction and Decomposition:** In a first step, *SysWCEC* derives control-flow graphs of the entire system from the source code. To keep this step feasible, we take advantage of the fact that only system calls, or syscalls for short, can alter the system state and thus the power state and set of active devices. Consequently, our approach is to decompose the code into coarse-grained blocks that span between syscalls and thus are atomic from a system-state perspective: this coarsening facilitates the subsequent state enumeration and allows *SysWCEC* to perform large parts of the WCRE analysis efficiently without losing precision of dynamic device activations and deactivations.
2. **Power-Aware System-State Enumeration:** In the second step, *SysWCEC* explicitly enumerates all possible block-to-block transitions considering the priorities of tasks, synchronous task activations, and asynchronous interrupts. The result of this symbolic state enumeration is a state graph that incorporates the operating-system semantics and thus the possible dynamic behavior of the system. This, in particular, links the code blocks from the previous step with state-dependent power states and device activities.
3. **ILP Formulation & WCRE Determination:** In the last step, *SysWCEC* determines the worst-case energy consumption of each state-graph node based on the worst-case execution time of the associated code block and the respective power states of active devices. Furthermore, it constructs an integer-linear program (ILP) to eventually derive the WCRE.

## 3.1 Abstraction and Decomposition

Recalling our goal of a fine-grained, state-dependent modeling of energy consumption, we first need a global control-flow graph that, in particular, incorporates inter-task dependencies as well as the operating system. The canonical approach to this would be a full path analysis on a basic-block level. This granularity is, however, too fine and infeasible for the vast number of possible program paths through an entire system [10, 38].

Nevertheless, to determine the WCRE, besides scheduling events, we are only interested in energy-relevant events, that is, spots in the control flow that have the potential to change the power consumption. In other words, we can abstract from sequences of instructions that share a particular power and system state if executed uninterrupted.

We consequently based *SysWCEC*'s analysis on previous work on the concept of *atomic basic blocks* (*ABBs*) [22, 55] to abstract from the code's microstructure and decompose the system. An ABB is a control-flow superstructure that subsumes one or more basic blocks and conceptually spans between syscalls. Each ABB has exactly one entry and one exit

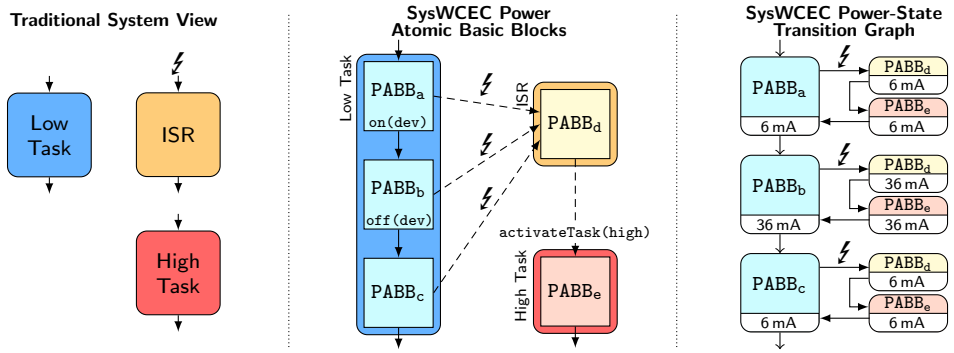



**Figure 3** Illustration of *SysWCEC*'s first two analysis steps (decomposition & state enumeration).

block, which typically is the delimiting syscalls, forming a single-entry single-exit region. As long as the result complies with this rule, ABBs may be split arbitrarily for optimization reasons. These construction rules imply that an ABB executes *atomically* from a scheduling perspective. Still, there is no correlation between ABBs and power states.

Building on this foundation, for WCRE analysis we therefore developed the concept of *power atomic basic blocks* (*PABBs*) with the additional property that the set of active devices and their power states does not change within a block. With the operating system being the governor of power states and devices, this boils down to an extended analysis and decomposition of the implementation: any device reconfiguration is considered as a dedicated syscall, a device syscall (see Section 2.1). Consequently, a PABB is atomically executed from both the scheduling and power perspective. In the resulting PABB graph (i.e., coarse-grained global control-flow graph), changes in the system state (i.e., operating-system and power states) are possible only at the edges between PABBs. Note that the PABB graph covers the entire system implementation and all machine instructions. By that, *SysWCEC* inherently considers overheads (e.g., context switch, syscall, and scheduling costs), which are often neglected in real-time scheduling approaches [15].

Figure 3 illustrates the decomposition into PABBs using the example system from Section 2.3. Following the construction rules, the low-priority task is split by the device syscalls (`on`/`off`) into three PABBs, which account for the actual power states and the utilization of the external device. Consequently, only $\texttt{PABB}_\texttt{b}$ is modeled with active power state, while the computation in $\texttt{PABB}_\texttt{a}$ and $\texttt{PABB}_\texttt{c}$ is assigned the correct inactive power state. Here, the state modification is associated with the edges between the three PABBs. Similarly, the effect of scheduling-related syscalls is handled as inter-task constraints between PABBs. For example, the activation of the high-priority task from the ISR ($\texttt{PABB}_\texttt{d}$ to $\texttt{PABB}_\texttt{e}$). We further discuss the handling of asynchronous interrupts in Section 3.2.

Overall the decomposition into PABBs on its own is already a significant improvement over the all-always-on assumption. Our approach allows for an independent analysis of implementation artifacts and states, which has three main advantages that highly benefit the subsequent steps: First, it substantially reduces analysis complexity and allows *SysWCEC* to examine an entire system by identifying all possible states, without the need of enumerating all possible program paths. Second, the fact that the power consumption does not change within a PABB greatly facilitates the problem of determining upper bounds for the block's energy consumption. Third, the single-entry single-exit property allows the reuse of previously developed whole-system and timing analysis techniques [17, 18, 19].

## 3.2 Power-Aware System-State Enumeration

So far, the PABB graph only captures a static view of the system structure as well as the interactions between application, operating system, and external devices. Therefore, *SysWCEC* leverages the PABB graph in a second analysis step to deduce the dynamic system behavior by an explicit enumeration of all feasible system states and all transitions between them. A key aspect of this step is to further enrich the analysis by a model of operating system behavior (i.e., fixed-priority scheduling and resource protocols), the system configuration (e.g., task priorities, minimal inter-arrival times, and deadlines), and an energy cost model of the external devices. The resulting power-state–transition graph (PSTG) ultimately exposes the aspired context-aware global execution paths, including synchronous and asynchronous preemptions (i.e., task switches and interrupts). Thereby, we are subsequently able to identify and eliminate infeasible combinations of system-wide execution paths and power states and thus further refine the input for the final step in Section 3.3. Overall, the PSTG holds all relevant information to safely formulate an ILP, whose solution yields the WCRE. In the following, we detail the elements of the PSTG as well as its construction and show how to incorporate the operating-system semantics and the energy costs of devices.

### Basic Principle and Operating-System Semantics

We begin our elaboration of the PSTG with its underlying principles and the operating-system–aware identification of possible execution paths. The basic construction rule is, as mentioned, to enumerate all possible system states and all transitions between them. A system state node is defined to hold the following information: (a) operating-system parameters, including the set of tasks with their current status (i.e., ready, running, suspended), priority, acquired resources, and resumption point as well as the ceiling priority. (b) Interrupt-related information including their status (i.e., enabled, pending, acknowledged). Finally, each state comprises (c) exactly one PABB and thread of execution, accordingly.

The construction algorithm starts with a dedicated entry state that is set up by the boot code. From this initial state, the application logic, which is obtained from the PABB graph, is simulated on a model of the operating system. At this point, the system configuration comes into play, which is used to instantiate the model to fit the concrete implementation. Subsequently, all reachable states are enumerated while the operating-system scheduling semantics are employed to discover inter-thread transitions. For example, when multiple tasks are runnable, the algorithm selects the task with the highest priority, and the follow-up state node references the task's entry PABB. Reconsidering the example system and its PSTG (see right part of Figure 3), the only successor of the interrupt is the runnable high-priority task. A transition to the low priority task is not possible in the fixed-priority scheduling model within this context-sensitive PSTG node. Moreover, tasks do not necessarily have their configured static priority due to shared resources and the employed priority-ceiling protocol [7] with its priority inheritance. Thus, tasks can have a dynamic priority, which is context-sensitively recorded in the task parameters of each PSTG node. Consequently, a PABB can occur multiple times in the PSTG with varying system states.

### Handling Interrupts

Although the scheduler treats PABBs as atomic units, asynchronous interrupts can be released within a PABB's execution. At runtime, the interrupt could occur after every instruction and thus multiple times during the PABB execution. To handle such asynchronous preemptions, the PSTG construction algorithm inserts transitions from interruptible PSTG nodes to the

entry functions of the ISRs (see the transition from lower-priority task to ISR in Figure 3). On the PSTG level, a single interrupt transition facilitates the state enumeration and is sufficient to enable compliance with the construction rules. We show in Section 3.3.2 how to bound the actual number of occurring interrupts in the final ILP formulation with help of the interrupt's minimum inter-arrival time and its response time from entry to return [19].

### Power States and Energy-Consumption Costs

Finally, the PSTG's most distinctive feature comes into play: the inference of the current power state and the set of active devices. The associated power consumptions are taken from the given energy cost model. This combined information is crucial for modeling the WCEC of the individual system states during the ILP construction (see Section 3.3).

The power states are determined as part of the state enumeration: the construction algorithms memorizes the last power state when following transitions and updates the state (i.e., set of active devices) whenever it encounters a device syscall. Consequently, all possible succeeding nodes obtain the updated power state. In the same way, energy penalties (e.g., caused by mode changes) are incorporated at node transitions. Figure 3 illustrates the resulting power states as state-dependent consumption data. In this example, the ISR and the high-priority task are penalized with the additional power consumption caused by the device activation in the low-priority task only if the interrupt occurs within the device's operation period (i.e., within $\texttt{PABB}_\texttt{b}$). Note that, as with scheduling, the PSTG construction only eliminates infeasible states. Still, it contains all feasible combinations of execution and power state. Thus, WCEC estimation is the responsibility of the following ILP step.

Overall, the final PSTG incorporates by construction all possible execution paths of the concrete system under consideration of operating-system locks, scheduling, and interrupts as well as device usage and energy penalties. This modeling approach, like the PABB graph, represents a genuine simplification since it allows for independent handling of the system state and thus does not bloat the following ILP formulation unnecessarily. In Section 3.4, we provide details on our analysis framework and on how the executable system is generated, which behaves identically to the PSTG's analysis model [16].

## 3.3 ILP Formulation & Determining the WCRE

In the following, we describe how *SysWCEC* formulates an ILP to determine the WCRE of the overall system based on the entire PSTG. Analyzing the WCRE of a particular task would require the same steps, but only consider a subgraph of the PSTG that spans from the task's release until its completion. Our approach is based on a sound extension [19] of the well-known and proven implicit path-enumeration technique [42, 52], which we adapt for whole-system worst-case energy-consumption analysis. Once formulated, the ILP can be solved with a mathematical optimizer to eventually compute the WCRE.

### 3.3.1 Integer Linear Program

The main idea behind the ILP produced by *SysWCEC* is to determine for each PSTG node $v$ how often the system executes the node in the worst case and to connect this execution frequency $f(v)$ to the worst-case energy consumption $E(v)$ of the machine code corresponding to the node. For this purpose, we rely on the following objective function to maximize the

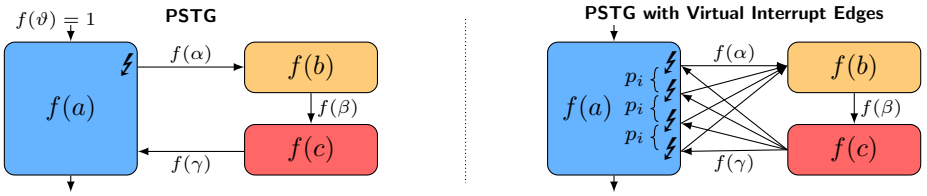



**Figure 4** The ILP formulation derived from the PSTG accounts for the interrupts and potential switches to higher-priority tasks.

flow through the graph:

$$WCRE = max\Big(\underbrace{\Big(\sum_{v \in V} E(v) \cdot f(v)\Big)}_{\text{nodes}} + \underbrace{\Big(\sum_{\varepsilon \in \mathcal{E}} E(\varepsilon) \cdot f(\varepsilon)\Big)}_{\text{edges}}\Big)$$

Apart from nodes, the objective function also considers worst-case energy costs $E(\varepsilon)$ for edges $\varepsilon \in \mathcal{E}$ in the PSTG. This allows us to take energy costs into account that are caused by transitions between different power modes of a device and are a result of the fact that power-mode changes for some devices do not complete instantaneously.

To provide sound results, *SysWCEC* requires $E(v)$ and $E(\varepsilon)$ to be upper bounds of the energy consumptions of the PABB associated to node $v$ and of the power-mode transition represented by edge $\varepsilon$, respectively. However, the *SysWCEC* approach does not make any assumptions on how these worst-case values are obtained which enables the reuse of existing WCEC analysis techniques [35, 50, 71]. One possibility to determine the WCEC for the PABB of a PSTG node $v$, for example, is to multiply the block's worst-case execution time $WCET(v)$ by $P_{max}(v)$, the maximum amount of power the system and its devices consume while the system is in the state represented by the PSTG node $v$; that is, $E(v) = WCET(v) \cdot P_{max}(v)$. As explained in Section 3.2, such knowledge about the power state of the system is part of the information maintained by *SysWCEC* in the PSTG and updated on each system-state transition. We discuss further refinements of this model in Section 6.

In addition to the objective function presented above, the ILP formulated by *SysWCEC* includes a set of constraints to specify dependencies between the execution frequencies $f(v)$ and $f(\varepsilon)$ of nodes and edges: (1) The entry and exit edge of the PSTG are each assigned a frequency of 1. (2) For each node $v$ in the PSTG, the sum $f_{in}(v)$ of the execution frequencies of all incoming edges must be equal to the node's execution frequency $f(v)$ and must match the sum $f_{out}(v)$ of the execution frequencies of all outgoing edges; that is, $\forall v \in V : f(v) = f_{in}(v) = f_{out}(v)$. This constraint preserves the flow through the graph.

### 3.3.2 Handling Interrupts in the ILP

If an interrupt can occur within the execution of a PSTG node's PABB, the graph contains a single corresponding interrupt-transition edge. However, as the interrupt may be triggered more than once, in the ILP *SysWCEC* needs to consider the interrupt multiple times. Figure 4 illustrates this scenario for an example PSTG with three nodes: a node $a$ depicting a low-priority task, a node $b$ representing the asynchronous interrupt, and a node $c$ referring to a high-priority task. With interrupts at most being released with a minimum inter-arrival time $p_i$ (see Section 2.1), there is an upper bound $N \in \mathbb{N}_0$ for the number of interrupts that can occur during the execution of a system. In our example $N = 4$, which also represents the number of times the PABB of node $a$ can be preempted and resumed. To bound $N$ we use

the following inequation based on $T$, the runtime of the system for the execution scenario in which the system achieves its worst-case response energy consumption:

$$p_i \cdot (N-1) \leq T$$

In a nutshell, this constraint expresses the fact that the longer the system runs (i.e., the larger $T$), the more interrupts $N$ may be triggered. In the worst case, the first interrupt is released right at the start of system execution, with another interrupt following every $p_i$. To determine $T$, we combine the worst-case execution times $WCET(v)$ and $WCET(\varepsilon)$ of all PSTG nodes and edges and combine them with the execution frequencies $f(v)$ and $f(\varepsilon)$ of the ILP's objective function presented in Section 3.3.1:

$$T = \Big( \sum_{v \in V} WCET(v) \cdot f(v) \Big) + \Big( \sum_{\varepsilon \in \mathcal{E}} WCET(\varepsilon) \cdot f(\varepsilon) \Big)$$

Relying on the same execution frequencies that are used to compute the WCRE ensures that $T$ actually represents the runtime of the scenario consuming the most energy.

Using the constraint presented above to bound the maximum number of interrupts, a solver is able to determine the WCRE of a system with interrupts, but may provide unnecessarily pessimistic results, as the following example based on Figure 4 shows: With the execution frequency of the graph's entry edge being $f(\vartheta) = 1$, the PABB of node $a$ under any circumstance is only executed once; that is, $f(a) = 1$. However, applying the flow-preserving constraint, without further action, a solver would compute an execution frequency of $f(a) = f_{in}(a) = f(\vartheta) + f(\gamma) = 1 + N = 5$, accounting for the fact that the interrupt may resume up to $N$ times. To prevent such overestimations, we differentiate between synchronous and asynchronous activations, which is knowledge that is already an attribute of the PSTG's transition edges (see Section 3.2). This approach allows us to consider and subtract the number of completed suspend-resume cycles when determining the execution frequency of a node $v$ as follows: $f(v) = f_{in,sync}(v) + f_{in,async}(v) - f_{out,IRQ}(v)$, with $f_{in,sync}(v)$ and $f_{in,async}(v)$ being the execution frequencies of all incoming synchronous and asynchronous edges, respectively, and $f_{out,IRQ}(v)$ representing the execution frequency of all outgoing interrupt edges of the node. For the example system, this optimization reduces the execution frequency of node $a$ to $f(a) = f(\vartheta) + f(\gamma) - f(\alpha) = 1 + N - N = 1$, and as a consequence correctly reflects the actual execution frequency of this node. In a similar way, *SysWCEC* is able to address interrupt preemptions that are not resumed, which can happen if the start and end point of the WCRE analysis are in different tasks. Note that in the example the described optimization only affects the execution frequency of node $a$. The execution frequencies of both other nodes still take the effects of multiple interrupts into account, resulting in frequencies of $f(b) = 0 + f(\alpha) - 0 = N$ and $f(c) = f(\beta) + 0 - 0 = N$.

## 3.4 Implementation

As shown in Figure 5, the *SysWCEC* toolchain relies on two main components: a modified version of the `dOSEK` framework [30] to construct the power-state–transition graph, and the `Platin` analysis toolkit [29, 51] to formulate the integer linear program necessary to determine the WCRE. Both the `dOSEK` system-analysis/-generation framework and the `Platin` timing-analysis toolkit are fundamentally based on the LLVM compiler infrastructure [40].

Provided with the specification of a real-time system and the implementation of tasks and interrupts, `dOSEK` is able to identify all possible system states and to automatically generate an executable and OSEK-compliant (i.e., ECC1 [49]) operating-system implementation. The conformance class ECC1 allows using prioritized, preemptible, self-suspending, and

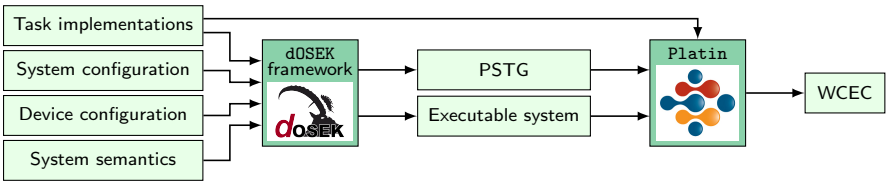



**Figure 5** Workflow of the *SysWCEC* analyzer.

work-preserving tasks. Tasks can wait for specific events and can acquire resources, whereas a stack-based priority-ceiling protocol (PCP) avoids unbounded priority inversion [6]. `dOSEK` is able to perform the entire system-state enumeration considering also the dynamic priorities due to the PCP. For *SysWCEC*, we made extensive enhancements to implement the concepts of analysis, decomposition, and state enumeration described earlier. In particular, we introduced the notion of device syscalls, added means to supply `dOSEK` with information about the maximum power consumption of devices in different modes, and enabled the framework to track the modes of devices across different system-state transitions. As a result of our modifications, `dOSEK` now performs a whole-system state analysis that takes devices into account and puts out the results in the form of the power-state–transition graph.

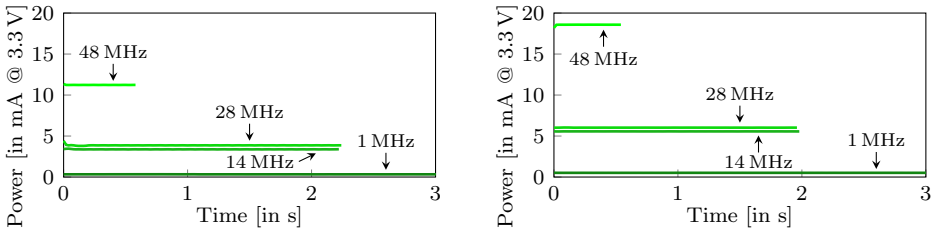
Providing `Platin` with the PSTG and the system implementation generated by `dOSEK`, we can use the toolkit to determine the WCET of PABBs. This allows us to compute the WCEC for each system state by multiplying the WCET of the associated block by the power-mode–specific maximum power consumption of all the devices that are active in the state. Based on this knowledge, `Platin` formulates the ILP for the WCRE bound that is then solved by the mathematical optimizer `Gurobi` [28].

# 4 Evaluation

In this section, we experimentally evaluate the *SysWCEC* approach and its prototype. Our focus in this context does not lie on proving that the WCRE values determined by *SysWCEC* are actually upper bounds for response energy consumption. As discussed in Section 3.3.1, due to relying on proven analysis techniques *SysWCEC* delivers sound results by construction, provided that the worst-case energy model used for the analysis is accurate. Creating energy models with such properties is feasible [50] but outside the scope of this paper, which is why in our evaluation we concentrate on assessing the effectiveness of *SysWCEC* in comparison to existing analysis techniques. To obtain meaningful results, for this purpose we require an energy model that comprises realistic values for the power consumption of different hardware units, including devices and peripherals, which we can then use as input for *SysWCEC*. In Section 4.1 we describe how we compiled the energy model for our experiments. We do not claim this model to contain guaranteed upper power-consumption limits. Nevertheless, due to offering information on the characteristics of real-world hardware components, the model allows us to evaluate *SysWCEC*'s ability to deal with temporarily active devices (Section 4.2), its context-sensitive analysis (Section 4.3), as well as its scalability (Section 4.4).

## 4.1 Energy Model

In order to be able to evaluate *SysWCEC* with realistic power and energy consumption values of devices, peripherals, and processors, our energy model combines knowledge from different sources (e.g., manuals, measurements) and different hardware platforms.




| Phase | Mean | Max. Upper Error | Standard Deviation (abs. / rel.) |
|---|---|---|---|
| CPU, 48 MHz | 11.23 mA | 0.01 mA | 3.7 µA / 0.03 % |
| CPU, 28 MHz | 3.86 mA | 0.06 mA | 14.61 µA / 0.38 % |
| CPU, 14 MHz | 3.38 mA | 0.08 mA | 12.08 µA / 0.36 % |
| CPU, 1 MHz | 0.33 mA | 0.01 mA | 10.19 µA / 3.09 % |
| Memory, 48 MHz | 18.58 mA | 4.32 µA | 3.24 µA / 0.02 % |
| Memory, 28 MHz | 6.03 mA | 0.02 mA | 3.20 µA / 0.05 % |
| Memory, 14 MHz | 5.57 mA | 3.69 µA | 1.93 µA / 0.03 % |
| Memory, 1 MHz | 0.51 mA | 2.71 µA | 1.04 µA / 0.20 % |

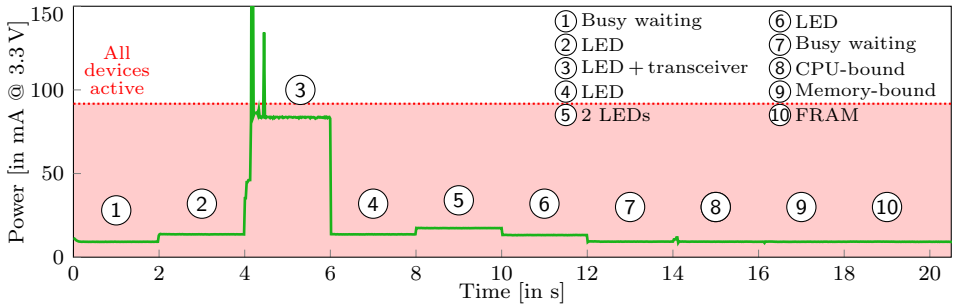
**Figure 6** ARM Cortex-M4: Traces for CPU-bound (left) and memory-bound benchmarks (right).

## Devices and Peripherals

Our first platform is an NXP FRDM KL46z evaluation board [23, 24] that features an ARM Cortex-M0+ core [4] with 256 KB of flash memory, 32 KB of SRAM, and a small cache (i.e., 4-way, 4-set program flash memory cache with a size of 64 B). We set up the evaluation board to run the execution pipeline at 48 MHz, the bus speed is 24 MHz. Apart from the processor, the board comprises a rich set of different devices including two LEDs, an accelerometer, a magnetometer, and an analog-to-digital converter. In addition, we attached an ESP8266 Wi-Fi module [67] as transceiver and an external ferroelectric RAM (FRAM) chip [25] as non-volatile storage. For most devices and peripherals, detailed documentation on the maximum power consumption is available (e.g., LEDs [46], accelerometer [48], magnetometer [47], analog-to-digital converter [24]). In all other cases, we obtain realistic power-consumption values by measurement relying on the source-measure unit Keithley 2612 [36], which is able to measure minimum currents of 100 fA and minimum voltages down to 100 nV at a temporal resolution of up to 20 µs. Using this source-measure unit circumvents the problem of potentially noisy power supplies and the problem of influencing the system under test with shunt-based measurement setups; both are known problems in the context of benchmarking low-power applications [21]. The results for our first platform are presented in Table 1.

## Processor Power Modes

With our first platform's processor only offering a few power modes, we use a second platform with a more complex processor (i.e., an EFM32 Giant Gecko evaluation board [63, 65] with an ARM Cortex-M4) to examine the effects of different processor power modes. For measurements, in this case we rely on the board's integrated current-measurement circuitry, which is able to quantify currents from 0.1 µA to 50 mA and allows us to measure the power consumption of the microcontroller and correlate it to the code executed. As programs, we select a CPU-bound benchmark performing a prime-number calculation and a memory-bound benchmark repeatedly copying data, because these two categories represent the two ends




| Phase | Mean | Max. Upper Error | Standard Deviation (abs. / rel.) |
|---|---|---|---|
| (1) Busy waiting | 9.21 mA | 0.09 mA | 0.04 mA / 0.41 % |
| (3) LED + transceiver | 87.57 mA | 220.84 mA | 33.45 mA / 38.19 % |
| (5) 2 LEDs | 17.38 mA | 0.12 mA | 0.04 mA / 0.23 % |
| (8) CPU-bound | 11.25 mA | 0.67 mA | 0.59 mA / 5.25 % |
| (9) Memory-bound | 9.13 mA | 0.28 mA | 0.26 mA / 2.82 % |
| (10) FRAM device | 9.3 mA | 0.09 mA | 0.03 mA / 0.37 % |

**Figure 7** Measurement results for phases with different sets of active devices.

of the spectrum with regard to power consumption [12, 75]. Figure 6 shows the results and illustrates the impact of different processor power modes on execution time and power consumption. *SysWCEC* addresses this issue by differentiating processor power modes when analyzing a system, thereby modeling the processor in a conceptually similar way to devices.

### Mode-Change Latencies

Hardware units not only consume energy while running in a certain power mode but also during the switch from one power mode to another. *SysWCEC* takes this fact into account by considering mode-change latencies and consequently attributing additional energy costs to power-mode switches. For most devices and peripherals of our two evaluation platforms, we found the associated mode-change overhead to be negligible, which is why in the following we focus on measurement results for the processor of our second platform. On the ARM Cortex-M4, switching from 28 MHz to 48 MHz, for example, takes 396 µs and comes with an energy overhead of 8.71 µJ. For comparison, the CPU-bound benchmark computing a 4-digit prime number on the same platform at 28 MHz requires 2 ms and 25.5 µJ. This shows that power-mode changes can have a significant impact on response time and energy consumption, although many energy-aware real-time scheduling approaches do not consider such overheads [8]. *SysWCEC*, on the other hand, includes time and energy costs for power-mode switches when analyzing worst-case (response) energy consumption.

## 4.2 WCRE Analysis for Temporarily Activated Devices

Using the energy model obtained in Section 4.1, in our first experiment we evaluate *SysWCEC* in the context of a system in which the set of active devices and peripherals constantly changes, as it is the case in a practical system that only activates devices temporarily in order to minimize energy consumption (see Section 2.2). The experiment runs on the Cortex-M0+ platform and, as shown in Figure 7, consists of phases with different sets of active devices, which results in varying overall power consumption. In each phase, we first execute a specific

**Table 2** WCRE-estimate comparison between the all-always-on approach and *SysWCEC*.

| Benchmark | WCRE All-Always-On | WCRE *SysWCEC* | Improvement |
|---|---|---|---|
| #1 Transceiver (w/o resource) | 4,389.10 µJ | 3,786.32 µJ | 13.73 % |
| #2 Transceiver (w/ resource) | 4,473.91 µJ | 818.66 µJ | 81.70 % |
| #3 Synchronous activation | 2,266.28 µJ | 1,236.15 µJ | 45.45 % |
| #4 Asynchronous IRQ | 399.88 µJ | 335.41 µJ | 16.12 % |

event (e.g., a device activation) or a specific job (e.g., a computation) and subsequently delay execution for 2 ms. When the system activates the transceiver at the beginning of Phase ③, our results show power-consumption spikes that last for a short period of time. The spikes are an artifact of our current prototype hardware and in a practical system can be prevented by additional hardware circuitry [32]. In general, the power consumption within each phase behaves almost linear, as also confirmed by the small standard deviations (see Figure 7).

The measurements on the hardware platform for the Phases ② to ⑤ illustrate that activating and deactivating devices has a considerable impact on the power consumption of the whole system, especially in comparison to Phase ① when all devices are still switched off. This observation confirms that for an analysis of a system's (worst-case) energy consumption it is not sufficient to only take the processor into account, but crucial to consider all power consumers in the system. Although the power consumption of the processor varies depending on whether it executes a CPU-bound job (e.g., a prime-number calculation in Phase ⑧) or a memory-bound job (e.g., copy operations in Phase ⑨), the overall impact of the work performed by the processor is comparatively small.

Using the common approach to determine the WCRE for a system with devices, that is, to assume that all devices are always on (see Section 2.2), for the evaluated scenario results in a significant overestimation, as indicated by the red area in Figure 7. In contrast, by decomposing the system into parts during which devices do not change power modes, *SysWCEC* is able to provide much lower bounds, for example, due to being aware that the transceiver is only operating in Phase ③ and definitely remains inactive the rest of the time. For this experiment, the *SysWCEC* approach leads to a WCRE of 398.53 mJ, which is 79.25 % lower than the value determined by the all-always-on approach, representing the difference between the area under the green curve and the red area in Figure 7.

## 4.3 Exploiting Context-Sensitive Knowledge

In our next experiments, we focus on systems with multiple tasks and compare the WCRE values provided by *SysWCEC* with the WCRE values determined with the all-always-on approach. To obtain representative energy-consumption values, we combine the target platform, in this case the PATMOS research processor [58, 59], with our energy model. We configure the processor to run at 1 MHz and a static power consumption (i.e., all devices deactivated) of 10mA. As workload, we select a total of four benchmarks with different characteristics to be able to evaluate a wide spectrum of scenarios. Table 2 presents and compares the WCRE values determined by both methods evaluated. To compute the WCRE estimate for the all-always-on approach, we multiply the exact worst-case response time of the system by the amount of power the system consumes when all devices are switched on. In the following, we discuss the results of each benchmark in detail.

### Benchmark #1 – Transceiver Benchmark

The structure of the first benchmark resembles the example in Figure 2: A low-priority task activates a device, in this case a WiFi transceiver, for example, to log the system status. During the execution of the task, an interrupt preempts the task and activates another higher-priority task, which is then dispatched after the interrupt service routine has terminated. For the worst-case scenario of the interrupt occurring while the transceiver is active, the low-priority task spends about 8 % of its overall response time before activating the device (i.e., Part Ⓐ in Figure 2), about 83 % of the time while the device is active (Part Ⓑ), and the remaining about 9 % of the time after having switched off the device (Part Ⓒ). Our results in Table 2 show that for this benchmark, the WCRE value determined by *SysWCEC* is 13.73 % lower than the WCRE value obtained with the all-always-on approach. This improvement is possible because due to its context-sensitive analysis of the system *SysWCEC* knows that it is impossible for the transceiver to be active during Part Ⓐ and Part Ⓒ of the low-priority task, which in combination represent about 17 % of the task's worst-case response time.

### Benchmark #2 – Transceiver Benchmark with Resource

For our second benchmark, we modify the first benchmark and introduce a shared resource between the low-priority task and the high-priority task. In OSEK, shared resources are typically used to coordinate different tasks and can only be acquired by at most one task at a time. In our benchmark, the low-priority task acquires the resource right before activating the transceiver and releases it immediately after deactivating the transceiver. Applying the stack-based priority ceiling protocol [6], which is mandated by the OSEK standard to solve the problem of unbounded priority inversion, when the interrupt occurs while the low-priority task holds the resource, the execution of the high-priority task is deferred until the resource has been released. As a consequence, the high-priority task no longer has an influence on the transceiver's active time, independent of when the interrupt is actually triggered. In contrast to the all-always-on approach, *SysWCEC* is able to exploit this knowledge and consequently determines a 81.70 % lower WCRE value, accounting for the fact that even in the worst case the transceiver is only active during a small part of the low-priority task's response time.

### Benchmark #3 – Synchronous Task Activation

In the next benchmark, a low-priority task synchronously activates two tasks with higher priorities, one prior to switching on a transceiver and the other one afterwards. As both high-priority tasks take the same time to run and due to their execution times dominating the response time of the low-priority task, the transceiver is activated about half way into the experiment. This leads to *SysWCEC* being able to provide a WCRE estimate that is 45.45 % lower than the all-always-on value.

### Benchmark #4 – Asynchronous Events

The fourth benchmark consists of a task activating a transceiver and an interrupt service routine deactivating it again. Such a setting represents a textbook example of why context-sensitive WCRE analysis is conceptually different from worst-case response time analysis: To determine the worst-case response time, an analysis must consider the scenario in which the interrupt occurs as often as its minimum inter-arrival time allows; in this case, this results in a response time of 1,377 cycles. In contrast, WCRE analysis must focus on the scenario with the highest energy consumption, which for this benchmark is the interrupt being triggered

**Table 3** Analysis runtimes and statistics for different benchmarks.

| Benchmark | State Enumeration | States | Transitions | ILP Solving |
|---|---|---|---|---|
| Transceiver (w/o resource) | 54.05 ms | 63 | 71 | 534 ms |
| Transceiver (w/ resource) | 69.23 ms | 84 | 96 | 869 ms |
| Synchronous activation | 40.44 ms | 19 | 19 | 308 ms |
| Asynchronous IRQ | 42.88 ms | 35 | 40 | 136 ms |
| Multiple devices (2 tasks) | 73.63 ms | 119 | 135 | 0.91 s |
| Multiple devices (3 tasks) | 535.22 ms | 1,356 | 1,580 | 10.41 s |
| Multiple devices (4 tasks) | 1.4 s | 6,231 | 7,359 | 54.17 s |
| Multiple devices (5 tasks) | 2.62 s | 39,711 | 47,215 | 33.17 min |

only once: at the very end of the task's computation; for this scenario, the response time is only 1,155 cycles. *SysWCEC* correctly identifies this scenario and determines a 16.12 % lower WCRE value than the all-always-on approach.

## 4.4 Scalability

In our last experiment, we evaluate the performance and scalability of the *SysWCEC* approach, thereby focusing on the two steps that contribute to the overall analysis runtime: the symbolic state enumeration performed by `dOSEK` and the solving of the ILP (see Section 3.3). All analyses run on a server (Intel Xeon E5, 80 cores, 132 GB RAM) and use Gurobi 7.5 for ILP solving. To reduce durations for ILP solving, we explored parameter-tuning strategies [28] and carried out optimizations in Gurobi, which eventually determines optimal bounds.

Apart from the results of the four benchmarks introduced in Section 4.3, we also present measurements gained from four additional benchmarks that use multiple devices and all share the following general structure: All of these benchmarks consist of a low-priority task whose WCRE is to be determined. Apart from this task, the benchmarks comprise a set of additional tasks that each possess a unique higher priority and are activated through dedicated asynchronous interrupt service routines; the minimum inter-arrival time between interrupts is 100 ms. All tasks in the system are assigned different devices. During execution, a task first switches on its device, then performs a computation, deactivates the device again, and finally terminates. To evaluate the impact of system complexity on analysis runtime, we rely on four different benchmarks whose task-set sizes range between 2 and 5. Note that due to the interfering interrupts and number of tasks, from a system-level perspective our multi-device benchmark comprising 5 tasks plus 4 task-activating interrupts has a comparable complexity as the real-world real-time benchmark DEBIE [31].

Table 3 compares the execution times for the two evaluated analysis steps and also presents the number of system states and transitions identified by *SysWCEC* for each benchmark. The results show that in general solving the integer linear program takes significantly more time than enumerating all system states and that the runtime of both analysis steps increases with the number of possible system states. In Section 6.2, we discuss how to further improve the performance and scalability of *SysWCEC* and its exponential state growth with increasingly complex systems. Nevertheless, even for systems with high complexity such as the multi-device benchmark with 5 tasks, preemptions through interrupts, and dependencies between tasks and interrupts, leading to 39,711 different system states and 47,215 transitions, *SysWCEC* can complete the entire analysis in around half an hour.

## 5 Related Work

The development of *SysWCEC* benefited from lessons learned in our previous work on system-wide timing analysis for fixed-priority real-time systems [19]. However, due to the substantial differences in objectives and requirements between timing analysis and energy-consumption analysis, we were not able to directly apply our existing analysis techniques to the problem of determining WCREs. Instead, it became necessary to develop a new approach, *SysWCEC*, that solves the specific problems associated with energy-consumption analysis. To the best of our knowledge, *SysWCEC* is the first work that enables determining WCEC bounds in peripheral-driven real-time systems that execute on various low-power modes.

Existing WCEC-analysis techniques [27, 35, 71] only consider the analysis of a single thread with a fixed set of active peripherals. In analogy to WCET tools, these analyzers follow the common approach of carrying out a hardware-independent path analysis that is combined with a hardware-dependent cost analysis. In contrast to these existing techniques, *SysWCEC* is able to analyze a whole embedded system with all attached or integrated power consumers. In addition to the awareness of all power-consuming devices and peripherals, the *SysWCEC* approach precisely addresses all non-hierarchical program flows such as synchronous task activations and asynchronous interrupts.

The integration of transceivers is common in the area of wireless sensor networks to estimate the overall lifetime of nodes [20, 39, 43]. Such lifetime-estimation tools are already featured by integrated development environments for battery-constrained devices [64]. Their basic principle is to multiply the maximum drawn power by the fixed duty cycle of periodically receiving/transmitting applications. In distinction to these approaches, *SysWCEC* is capable of modeling fixed-priority sporadic task sets with real-time constraints. An interleaved timing analysis determines the duration of activated devices and these costs are integrated into an overarching ILP formulation, which yields WCRE bounds.

Schneider pointed out that it is impossible to analyze timing constraints of applications without considering the semantics of the operating system and vice versa [56]. To solve this problem, he proposed to integrate fixed-priority scheduling semantics into timing analysis [57]. Following the idea of whole-system analysis, *SysWCEC* provides means to integrate multiple devices to determine WCRE bounds between two arbitrary program points in the system. The integration of operating-system standards [1, 49] into commercial, static analysis tools, such as Astrée, indicates the relevance of system semantics in analyzers [44].

In the context of real-time scheduling, a system's power consumption is often determined based on the frequency-aware power model [14, 33, 77, 78], which is then exploited by DVFS. This power model assumes that the system's dynamic power consumption only depends on the processor's frequency. However, when considering systems that use devices, such as sensors, this model is no longer applicable. *SysWCEC* addresses the integration of devices with knowledge of global system paths and their set of active devices.

Furthermore, many scheduling approaches using DVFS to minimize energy consumption while still guaranteeing timeliness neglect the overheads to switch the processor frequency [8]. As we found out for an ARM Cortex-M4 (EFM32), the latency to switch the frequency can be up to 396 µs. Rusu et al. and Zhang et al. discovered even greater latencies for the PowerPC 405LP [54] and a digital signal processor [76]. Since *SysWCEC*'s central data structure, the PSTG, contains all executed instructions and all possible paths in the whole system, it is inherently aware of these transition overheads.

In addition to the DVFS power model, a huge body of related work exists on energy-cost models for the processor's microarchitecture [13, 37, 41, 45, 50, 60, 61, 69]. However,

considering the relations of power consumers in energy-constrained systems, processing cores only take a minor portion of the whole-system consumption. *SysWCEC* focuses on precisely integrating these consumers and determines WCRE bounds, but can profit from these advances on more fine-grained energy-consumption models.

# 6 Discussion & Future Work

In this section, we first discuss both improvements of our energy-consumption model and the scalability of the approach, and then outline our future work in the context of *SysWCEC*.

## 6.1 Improving Energy-Consumption Hardware Model

*SysWCEC* is a sound formulation of the path-analysis problem of whole-system WCRE analyses. In the current implementation, we use the maximum power consumption of a PSTG node and multiply it by its WCET to obtain an upper bound for the node's energy consumption. Our measurements in Section 4.2 show only small variations in power consumption within a power mode (i.e., around 5 %). This is especially true when these minor variations are put into relation with the orders of magnitude the whole system's power consumption varies when switching between power states. Consequently, we expect our method for determining a node's WCEC to result in only small overestimations. However, if necessary this model could be further improved with knowledge of the platform's microarchitecture [13, 37, 41, 45, 50, 60, 61, 69].

## 6.2 Further Improving Scalability

Although already providing reasonable analysis durations of around half an hour for larger systems (see Section 4.4), the scalability of the approach can be further enhanced. In particular, there are three directions for optimizing *SysWCEC* and mitigating the problem of exponential state growth: (1) constructing a smaller PSTG that also reduces the ILP size, (2) speeding up PSTG construction itself, and (3) improving ILP-solving times. First, it is possible to group several PSTG nodes together into a larger super structure [18]. Although this approach might sacrifice precision and lead to higher overestimations, it potentially enables smaller ILP formulations and thus shorter solving times. Second, `dOSEK`'s current state enumeration is implemented using a single thread. Consequently, the framework would greatly benefit from parallel symbolic state enumeration [11] and the usage of multi-core platforms. Third, although we applied heuristics to speed up ILP solving [28], the process can be further enhanced by providing upper bounds for variables by exploiting traditional WCRT analyses [5]. Thereby, search spaces can be narrowed and thus solving times reduced.

## 6.3 Trading Timeliness for Energy Consumption

Trading worst-case energy consumption for worst-case execution time and vice versa is an upcoming research area [26, 34]. In the evaluation, Benchmark #2 demonstrated that blocking the execution of a higher-priority task can be beneficial if it prevents the preemption of a lower-priority task that has previously activated a device. This blocking leads to a prolonged WCRT of the higher-priority task, but – depending on the deadlines, duration of the critical section, and power consumptions – the overall benefit can be optimized. With *SysWCEC*, we provide a framework that is capable of analyzing both WCRE and WCRT bounds along critical program paths of entire systems. This comprehensive framework is beneficial to find optimal solutions for peripheral-driven real-time systems in the time-vs-energy trade-off.

## 7 Conclusion

Power consumption in energy-constrained real-time systems depends not only on the processing unit but also on peripherals. Although these devices are often dynamically (de-)activated, current WCEC analyses do not take this fine-grained structure into account and therefore yield overestimations. Furthermore, with the (de-)activation of devices, the energy consumption of a single task can influence any other task (also tasks of higher priority).

The *SysWCEC* analyzer processes OSEK-compliant (i.e., ECC1) real-time systems and determines WCRE bounds. For this, we present the power-state–transition graph, a device and operating-system–aware data structure. Using this representation, we are able to enumerate all possible system states of fixed-priority real-time systems using multiple devices. This knowledge allows formulating an ILP, whose solution eventually yields the WCRE.

**Source code of *SysWCEC*: `https://gitlab.cs.fau.de/syswcec`**

## References


1 AEEC. Avionics application software standard interface (ARINC specification 653-1), 2003.

2 aiT worst-case execution time analyzers. `http://www.absint.com/ait/`.

3 ARM Limited. Cortex-M4 technical reference manual, 2010.

4 ARM Limited. Cortex-M0+ technical reference manual, 2012.

5 N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

6 T. P. Baker. A stack-based resource allocation policy for realtime processes. In *Proc. of the 11th Real-Time Systems Symp. (RTSS '90)*, pages 191–200, 1990.

7 T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

8 M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. on Embedded Computing Systems (ACM TECS)*, 15(1):7, 2016.

9 L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, 2000.

10 A. Biere, J. Knoop, L. Kovács, and J. Zwirchmayr. The auspicious couple: Symbolic execution and WCET analysis. In *Proc. of the 13th Int'l Work. on Worst-Case Execution Time Analysis (WCET '13)*, pages 53–63, 2013.

11 S. Bucur, V. Ureche, C. Zamfir, and G. Candea. Parallel symbolic execution for automated real-world software testing. In *Proc. of the ACM SIGOPS European Conf. on Computer Systems (EuroSys '11)*, pages 183–197, 2011.

12 A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proc. of the USENIX Annual Technical Conf. (ATC '10)*, pages 1–14, 2010.

13 N. Chang, K. Kim, and H. G. Lee. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In *Proc. of the 2000 Int'l Symp. on Low Power Electronics and Design (ISLPED '00)*, pages 185–190, 2000.

14 J.-J. Chen and C.-F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *Proc. of the 13th Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA '07)*, pages 28–38, 2007.

**15** R. I. Davis. On the evaluation of schedulability tests for real-time scheduling algorithms. In *Proc. of the Work. on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS '16)*, 2016.

**16** H.-P. Deifel, C. Dietrich, M. Göttlinger, D. Lohmann, S. Milius, and L. Schröder. Automatic verification of application-tailored OSEK kernels. In *Proc. of the 17th Conf. on Formal Methods in Computer-Aided Design (FMCAD '17)*, pages 1–8, 2017.

**17** C. Dietrich, M. Hoffmann, and D. Lohmann. Cross-kernel control-flow-graph analysis for event-driven real-time systems. In *Proc. of the Conf. on Languages, Compilers and Tools for Embedded Systems (LCTES '15)*, pages 6:1–6:10, 2015.

**18** C. Dietrich, M. Hoffmann, and D. Lohmann. Global optimization of fixed-priority real-time systems by RTOS-aware control-flow analysis. *ACM Trans. on Embedded Computing Systems (ACM TECS)*, 16:35:1–35:25, 2017.

**19** C. Dietrich, P. Wägemann, P. Ulbrich, and D. Lohmann. SysWCET: Whole-system response-time analysis for fixed-priority real-time systems. In *Proc. of the 23nd Real-Time and Embedded Technology and Applications Symp. (RTAS '17)*, pages 37–48, 2017.

**20** W. Dron, S. Duquennoy, T. Voigt, K. Hachicha, and P. Garda. An emulation-based method for lifetime estimation of wireless sensor networks. In *Proc. of the Int'l Conf. on Distributed Computing in Sensor Systems (DCOSS '14)*, pages 241–248, 2014.

**21** EEMBC ULPMark FAQs. `http://www.eembc.org/ulpbench/faq.php`.

**22** F. Franzmann, T. Klaus, P. Ulbrich, P. Deinhardt, B. Steffes, F. Scheler, and W. Schröder-Preikschat. From intent to effect: Tool-based generation of time-triggered real-time systems on multi-core processors. In *Proc. of the 19th Int'l Symp. on Real-Time Distributed Computing (ISORC '16)*, pages 134–141, 2016.

**23** Freescale Semiconductor, Inc. *KL46 Sub-Family Reference Manual*, 2013.

**24** Freescale Semiconductor, Inc. *Kinetis KL46 Sub-Family*, 2014.

**25** Fujitsu Semiconductor. *FRAM MB85RC256V*, 2013.

**26** R. Gran, J. Segarra, C. Rodríguez, L. C. Aparicio, and V. Viñals. Optimizing a combined WCET-WCEC problem in instruction fetching for real-time systems. *Journal of Systems Architecture*, 59(9):667–678, 2013.

**27** N. Grech, K. Georgiou, J. Pallister, S. Kerrison, J. Morse, and K. Eder. Static analysis of energy consumption for LLVM IR programs. In *Proc. of the 18th Int'l Work. on Software and Compilers for Embedded Systems (SCOPES 2015)*, pages 12–21. ACM, 2015.

**28** Gurobi Optimization, Inc. Gurobi optimizer reference manual. `www.gurobi.com`, 2017.

**29** S. Hepp, B. Huber, D. Prokesch, and P. Puschner. The platin tool kit - the T-CREST approach for compiler and WCET integration. In *Proc. of the 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung (KPS '15)*, pages 277–292, 2015.

**30** M. Hoffmann, F. Lukas, C. Dietrich, and D. Lohmann. dOSEK: The design and implementation of a dependability-oriented static embedded kernel. In *Proc. of the 21st Real-Time and Embedded Technology and Applications Symp. (RTAS '15)*, pages 259–270, 2015.

**31** N. Holsti, T. Langbacka, and S. Saarinen. Using a worst-case execution time tool for real-time verification of the DEBIE software. In *Proc. of the Data Systems in Aerospace Conf. (DASIA '00)*, pages 1–6, 2000.

**32** O. Hruška. Esp8266 killing itself? `ondrovo.com/a/20170205-esp-self-destruct/`, 2017.

**33** P. Huang, Pratyush Kumar, G. Giannopoulou, and L. Thiele. Energy efficient DVFS scheduling for mixed-criticality systems. In *Proc. of the Int'l Conf. on Embedded Software (EMSOFT '14)*, pages 1–10, 2014.

**34** C. Imes, DHK Kim, M. Maggio, and H. Hoffmann. POET: A portable approach to minimizing energy under soft real-time constraints. In *Proc. of the 21th Real-Time and Embedded Technology and Applications Symp. (RTAS '15)*, pages 75–86, 2015.

**35** R. Jayaseelan, T. Mitra, and X. Li. Estimating the worst-case energy consumption of embedded software. In *Proc. of the 12th Real-Time and Embedded Technology and Applications Symp. (RTAS '06)*, pages 81–90, 2006.

**36** Keithley Instruments Inc. *Models 2611B, 2612B and 2614B, System SourceMeter*, 2013.

**37** S. Kerrison and K. Eder. Energy modeling of software for a hardware multithreaded embedded microprocessor. *ACM Trans. on Embedded Computing Systems (ACM TECS)*, 14(3):56, 2015.

**38** J. Knoop, L. Kovács, and J. Zwirchmayr. WCET squeezing: On-demand feasibility refinement for proven precise WCET-bounds. In *Proc. of the 21st Int'l Conf. on Real-Time Networks and Systems (RTNS '13)*, pages 161–170, 2013.

**39** A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Haneveld, T. Parker, O. Visser, H. Lichte, and S. Valentin. Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In *Proc. of the 1st Int'l Conf. on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (SimuTools '08)*, pages 1–10, 2008.

**40** C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. of the Int'l Symp. on Code Generation and Optimization (CGO '04)*, pages 75–86, 2004.

**41** S. Lee, A. Ermedahl, S. L. Min, and N. Chang. An accurate instruction-level energy consumption model for embedded RISC processors. *SIGPLAN Notices*, 36(8):1–10, 2001.

**42** Y.-T. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *Proc. of the 32nd Annual Design Automation Conf. (DAC '95)*, pages 456–461, 1995.

**43** Y. Liu, W. Zhang, and K. Akkaya. Static worst-case energy and lifetime estimation of wireless sensor networks. In *Proc. of the 28th Int'l Performance Computing and Communications Conf. (IPCCC '09)*, pages 17–24, 2009.

**44** A. Miné, L. Mauborgne, X. Rival, J. Feret, P. Cousot, D. Kästner, S. Wilhelm, and C. Ferdinand. Taking static analysis to the next level: Proving the absence of run-time errors and data races with Astrée. In *Proc. of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, pages 570–579, 2016.

**45** S. Nikolaidis, N. Kavvadias, P. Neofotistos, K. Kosmatopoulos, T. Laopoulos, and L. Bisdounis. Instrumentation set-up for instruction level power modeling. In *Integrated Circuit Design*, pages 71–80, 2002.

**46** NXP Semiconductors. *Manufacturer BOM Report*, 2013.

**47** NXP Semiconductors. *Xtrinsic MAG3110 Three-Axis, Digital Magnetometer*, 2013.

**48** NXP Semiconductors. *MMA8451Q, 3-axis, 14-bit/8-bit digital accelerometer*, 2017.

**49** OSEK/VDX Group. Operating system specification 2.2.3. Technical report, OSEK/VDX Group, February 2005.

**50** J. Pallister, S. Kerrison, J. Morse, and K. Eder. Data dependent energy modelling: A worst case perspective. *Computing Research Repository, arXiv*, 2015.

**51** P. Puschner, D. Prokesch, B. Huber, J. Knoop, S. Hepp, and G. Gebhard. The T-CREST approach of compiler and WCET-analysis integration. In *Proc. of the 9th Work. on Software Technologies for Future Embedded and Ubiquitious Systems (SEUS '13)*, pages 33–40, 2013.

**52** P. Puschner and A. Schedl. Computing maximum task execution times: A graph-based approach. *Real-Time Systems*, 13:67–91, 1997.

**53** RapiTime - worst-case execution time (WCET) analysis for critical systems. `http://www.rapitasystems.com/products/RapiTime`.

**54** C. Rusu, R. Xu, R. Melhem, and D. Mossé. Energy-efficient policies for request-driven soft real-time systems. In *Proc. of the 16th Euromicro Conf. on Real-Time Systems (ECRTS '04)*, pages 175–183, 2004.

**55** F. Scheler and W. Schröder-Preikschat. The real-time systems compiler: Migrating event-triggered systems to time-triggered systems. *Software: Practice and Experience*, 41(12):1491–1515, 2011.

**56** J. Schneider. Why you can't analyze RTOSs without considering applications and vice versa. In *Proc. of the 2nd Int'l Work. on Worst-Case Execution Time Analysis (WCET '02)*, pages 79–84, 2002.

**57** J. Schneider. *Combined schedulability and WCET analysis for real-time operating systems*. Shaker, 2003.

**58** M. Schoeberl, F. Brandner, S. Hepp, W. Puffitsch, and D. Prokesch. Patmos reference handbook. Technical report, Technical University of Denmark, 2014.

**59** M. Schoeberl et al. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61:449–471, 2015.

**60** Y. S. Shao and D. Brooks. Energy characterization and instruction-level energy model of Intel's Xeon Phi processor. In *Proc. of the Int'l Symp. on Low Power Electronics and Design (ISLPED '13)*, pages 389–394, 2013.

**61** V. Sieh, R. Burlacu, T. Hönig, H. Janker, P. Raffeck, P. Wägemann, and W. Schröder-Preikschat. An end-to-end toolchain: From automated cost modeling to static WCET and WCEC analysis. In *Proc. of the 20th Int'l Symp. on Real-Time Distributed Computing (ISORC '17)*, pages 1–10, 2017.

**62** Silicon Laboratories, Inc. *Energy Debugging Tools for Embedded Applications*.

**63** Silicon Laboratories, Inc. *User Manual Starter Kit EFM32GG-STK3700*, 2013.

**64** Silicon Laboratories, Inc. *AN0822: Simplicity Studio™User's Guide*, 2016.

**65** Silicon Laboratories, Inc. *EFM32GG Reference Manual*, 2016.

**66** Symtavision tools: SymTA/S. `http://www.symtavision.com/symtas.html`.

**67** Espressif Systems. *ESP8266 Technical Reference*, 2017.

**68** L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. of the Int'l Symp. on Circuits and Systems (ISCAS '00)*, volume 4, pages 101–104, 2000.

**69** V. Tiwari and M. T.-C. Lee. Power analysis of a 32-bit embedded microcontroller. *VLSI Design*, 7(3):225–242, 1998.

**70** M. Völp, M. Hähnel, and A. Lackorzynski. Has energy surpassed timeliness? – scheduling energy-constrained mixed-criticality systems. In *Proc. of the 20th Real-Time and Embedded Technology and Applications Symp. (RTAS '14)*, pages 275–284, 2014.

**71** P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat. Worst-case energy consumption analysis for energy-constrained embedded systems. In *Proc. of the 27th Euromicro Conf. on Real-Time Systems (ECRTS '15)*, pages 105–114, 2015.

**72** P. Wägemann, T. Distler, H. Janker, P. Raffeck, and V. Sieh. A kernel for energy-neutral real-time systems with mixed criticalities. In *Proc. of the 22nd Real-Time and Embedded Technology and Applications Symp. (RTAS '16)*, pages 25–36, 2016.

**73** P. Wägemann, T. Distler, H. Janker, P. Raffeck, V. Sieh, and W. Schröder-Preikschat. Operating energy-neutral real-time systems. *ACM Trans. on Embedded Computing Systems (ACM TECS)*, pages 11:1–11:25, 2017.

**74** R. Wilhelm et al. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. on Embedded Computing Systems (ACM TECS)*, 7(3):1–53, 2008.

**75** C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. AppScope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Proc. of the USENIX Annual Technical Conf. (ATC '12)*, pages 1–14, 2012.

**76** G. Zeng, T. Yokoyama, H. Tomiyama, and H. Takada. Practical energy-aware scheduling for real-time multiprocessor systems. In *Proc. of the 15th Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA '09)*, pages 383–392, 2009.

**77** D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the Int'l Conf. on Computer Aided Design (ICCAD '04)*, pages 35–40, 2004.

**78** Y. Zhu and F. Mueller. Feedback EDF scheduling exploiting dynamic voltage scaling. In *Proc. of the 10th Real-Time and Embedded Technology and Applications Symp. (RTAS '04)*, pages 84–93, 2004.