Appears in: Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '24)
Copenhagen, Denmark, June 24, 2024

WoCA: Avoiding Intermittent Execution in Embedded Systems by Worst-Case Analyses with Device States

Phillip Raffeck
FAU Erlangen-Nürnberg
Erlangen, Germany

Johannes Maier
FAU Erlangen-Nürnberg
Erlangen, Germany

Peter Wägemann
FAU Erlangen-Nürnberg
Erlangen, Germany

Abstract

Embedded systems with intermittent energy supply can revolutionize the Internet of Things, as they are energy self-sufficient due to energy harvesting. Existing intermittent-computing approaches, running directly from non-volatile memory, allow incremental progress of machine-code instructions. However, this progress does not apply to many devices (e.g., transceivers) having transactional (i.e., all-or-nothing) semantics: Power failures during transactions lead to starvation when frequently experiencing failed attempts.

We introduce WoCA, an approach that exploits static, whole-system worst-case analysis for device-driven intermittent computing. With the currently available energy, WoCA enables transactional device uses and guarantees forward progress. WoCA's novel static analysis tracks program-path–sensitive device states and transitions to yield energy bounds. With these bounds, WoCA's runtime decides when to safely execute code between checkpoints. Using WoCA's hardware platform, we validate that WoCA makes more efficient use of available energy compared to worst-case–agnostic approaches, while also giving runtime guarantees.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**.

Keywords: static analysis, intermittent systems, worst-case energy consumption (WCEC), energy constraints, devices

ACM Reference Format:
Phillip Raffeck, Johannes Maier, and Peter Wägemann. 2024. WoCA: Avoiding Intermittent Execution in Embedded Systems by Worst-Case Analyses with Device States. In *Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '24), June 24, 2024, Copenhagen, Denmark*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3652032.3657569

LCTES '24, June 24, 2024, Copenhagen, Denmark

ACM ISBN 979-8-4007-0616-5/24/06
https://doi.org/10.1145/3652032.3657569

1 Introduction

Energy-Self-Sufficient IoT. According to ARM's outlook, the number of their produced Internet-of-Things (IoT) systems is estimated to be one trillion within the next decade [40]. The smallest of these systems have no dedicated power supply because they harvest their energy from the environment (e.g., solar cells). Numerous researchers agreed that among the central challenges for the 2020s are such energy-independent systems [1]. While energy-self-sufficient systems have the benefit of avoiding battery replacement, they come with the central challenge of an unstable power supply. As a consequence of this supply and the limited energy storage, these systems face *intermittent operation* [27].

Intermittent Operation. To provide reliable execution, intermittency approaches have to address *(1) forward progress* and *(2) memory consistency*: Checkpointing the system is essential to resume execution safely after a power failure [4, 28, 49]. Once sufficient energy is harvested, the system restores from non-volatile memory and continues operation until reaching subsequent checkpoints. Re-executing from a checkpoint when data was written to non-volatile memory prior to the power failure can invalidate memory consistency [28]. Existing intermittency approaches avoid periodic checkpointing and ensure memory consistency by means of *just-in-time (JIT) checkpointing* [4, 6, 21, 28]: The hardware support for charge assessment of such approaches releases an exception prior to facing a power loss, which then leads to a checkpoint. While existing approaches guarantee memory consistency, they are not able to ensure that device-driven systems make *forward progress*. Approaches exist to reduce the energy wasted by re-executing code [52]. However, at worst, these approaches continuously waste energy without reaching a subsequent checkpoint, suffering from starvation.

Worst-Case Analysis. Recent work identifies correctness guarantees as a crucial research direction for intermittent systems [3]. We argue that giving runtime guarantees is only possible by capturing the system's dynamic behavior through static analysis [13]. In real-time systems, static analysis techniques for determining the *worst-case execution time (WCET)* [48] are the only means for providing safe execution within time budgets. In contrast, dynamic profiling techniques are incapable of providing runtime guarantees. For energy consumption, researchers have been making progress over the last decades in determining upper bounds for the *worst-case energy consumption (WCEC)* [22, 29, 32,

42, 45–47]. The fundamental methodology for determining WCEC is to describe the possible runtime behavior (i.e., possible execution paths) by means of a formal problem formulation and combine this formulation with a hardware-specific cost model. Solving the formulation then yields resource bounds.

Worst-Case Analysis in Intermittent Systems. The use of WCEC estimates has recently gained attention in intermittent systems: *RockClimb* [12] uses WCETs to compute a safe active time of code regions, which is used at runtime to decide whether a region can be transactionally executed. Yarahmadi et al. [51] use WCECs to guarantee power-failure freedom for executing basic blocks. Having guarantees for the execution of regions, including checkpoints, has three benefits: First, worst-case estimates allow guaranteed power-failure–free execution, avoiding starvation. Second, worst-case awareness reduces the complexity of checkpointing approaches because memory consistency is straightforward to guarantee. Third, avoiding the creation of checkpoints reduces the induced time and energy overheads. However, no approach addressed device-driven systems and their states.

Device-Driven Systems with Interrupts. Real-world systems usually have two properties that significantly complicate their WCET/WCEC analysis: *(peripheral) devices* (e.g., sensors, transceivers) in intermittent systems [6] and the presence of *asynchronous interrupts* (e.g., device-related interrupts). Regarding the whole system's power demand, devices play a major role: For example, on our evaluation platform [15], the CPU itself requires around 20 mW, while our used LoRa [44] radio transceiver requires up to 250 mW. Besides the discrepancy in power demand, the semantics of the transmission differ from the execution of instructions: Under the assumption that CPU instructions execute directly from non-volatile memory, as in *Ratchet* [43], instructions can be executed incrementally. On the contrary, sending a packet inherently has *transactional semantics*. For example, a power outage right before the completed transmission renders the packet useless. Similar considerations also hold when communicating with devices over the commonly used SPI bus: In addition to the energy waste of uncompleted transactions, facing a power failure during SPI communication (e.g., with a memory device) can leave the device in an inconsistent state. In summary, the semantics of certain devices require a transactional execution model, thus requiring the avoidance of intermittency for code regions. Besides these transactional semantics, a further aspect that unavoidably comes with handling devices is the presence of interrupts: The asynchronous nature of interrupts complicates capturing their dynamic behavior within a static analysis. However, for the practical applicability of worst-case estimates, these device-related interrupts must be part of the system's analysis. We argue that devices are often the main driver (i.e., sensing, communication) of (IoT) systems, making them *device-driven*.

WoCA's Contributions. We present WoCA, an approach that exploits worst-case analyses for intermittent systems. We leverage WCEC analysis to avoid intermittent execution of code with regard to complex multi-state devices: Based on the WCEC estimates, we use a runtime system that spans software- and hardware-related aspects for checkpoint management. We show how we integrate peripherals into the static analysis to achieve guaranteed execution of an embedded system that communicates with a transceiver device (i.e., LoRa). Using WCEC analysis in device-driven intermittent systems enables us to achieve our main goals of forward progress and memory consistency. Our approach to avoiding intermittency leverages a WCEC analysis, named *SysWCEC* [45], which is a tool integrated into a compiler. However, this tool is only capable of handling devices with binary states (i.e., on/off). That is, the analysis misses the handling of different device states (and the related transitions) in a program-path–specific way. WoCA's differentiation of *context-sensitive device states* is a requirement for WoCA's device-driven intermittent systems. In summary, this paper makes the following four contributions: (1) *Worst-Case– & Device-Aware Checkpoints*: The WoCA approach is the first to handle devices with worst-case bounds for forward-progress and memory-consistency guarantees in intermittent systems. (2) *Energy-Consumption Model & Static WCEC Analysis*: We show how our energy- and timing-related models of devices are safely integrated into WoCA's static WCEC analysis. Thereby, WoCA is the first approach that introduces the notion of *context-sensitive device states and their related transitions* as part of the analysis. (3) *Open-source Software & Hardware Implementation*: The open-source prototype of WoCA comprises both a *hardware prototype* (i.e., PCB) with accurate energy accounting as well as the hardware-aware *software implementation*. (4) *Evaluation Comparing WoCA with Existing Work*: We conduct evaluations to show that exploiting holistic knowledge of both the software and hardware has advantages over existing approaches.

2 Background & System Model

As WoCA relies on static analysis, our system model has assumptions to statically describe all dynamically possible cases. We consider these assumptions realistic since they hold for our hard- and software implementation (Section 5).

Resource Compositionality. WoCA targets low-power applications on single-core processors. The processor's microarchitectural behavior (i.e., caching, pipelining) is deterministic (i.e., freedom from timing anomalies [18]). As a consequence thereof, the worst-case execution time of two successive tasks ($WCET(\tau_1) + WCET(\tau_2)$), can be safely combined to yield the composed demand. We bypass the necessity of precisely modeling the processors' caching behavior by placing all executed program code and data into the zero-wait–cycle accessible SRAM [15, 16]. Due to the

simplicity of our targeted systems, determining the temporal behavior on a processor-cycle–accurate level is possible.

Power-Related Behavior. The power behavior requires special attention for determining safe energy-consumption estimates, as power and, eventually, energy (i.e., power over time) have several system-wide influences. Determining WCECs –without device awareness– is possible with similar considerations as determining WCETs on a cycle-accurate level. In this context, several works exist for instruction-level energy models for processors [29, 39]. However, most devices differ in their temporal behavior from processors, as they, for example, require power for a distinct amount of time (Δt) when being in the state of transmitting packets. With regard to our targeted transceiver devices and their temporal behavior [33], we model the energy demand of transceiver states with their maximum power multiplied by the respective worst-case time in this state. As a result, the energy demand of such operations can be generally bounded by $WCEC(\Delta t) = \int_{t=0}^{t=\Delta t} P_{max}(t)dt$, where the timespan Δt refers to the WCET of this operation. Refinements of the P_{max} estimates are possible with knowledge of the target application [10]. Regarding P_{max}, WoCA treats the processor itself as a device with state-dependent maximum power demands. In summary, all power-related behavior is software-controlled and thus analyzable by static analysis.

Non-Volatile Memory. For checkpointing across power outages, WoCA requires non-volatile memory (NVM). In contrast to other approaches [12, 28], direct code execution from NVM is not required, as WoCA's guarantees to run checkpoints to completion loosens this NVM-related restriction. Avoiding this restriction is valuable since the availability of embedded hardware platforms directly executing from NVM is severely limited, with the MSP430FR series being the de facto standard for intermittent systems [41].

Devices & Their Penalties. The NVM in WoCA systems is part of the set of devices. We assume that practical IoT systems have at least one transceiver device. Further, our system model includes sensor devices (e.g., temperature, air quality). For all devices, their states (e.g., receive, transmit) are controlled by the system's software, and their respective *context-sensitive* P_{max} estimates are statically known. WoCA's notion of devices regards them not necessarily as "peripheral" since modern system-on-chip platforms directly include numerous components. As a generic abstraction, our notion treats any power-consuming component as a device with different states and respective power demands. Further, switching between these device states requires *context-sensitive time/energy transition penalties*. Each penalty consists of a software-related part (e.g., writing the device's registers) and a hardware-related part (e.g., waiting for the device's state transition). Resource-consumption values of device states and transition penalties are available by documentation or by accurate, measurement-based determination.

Charge for Useful Work & State of Charge. The assumption of transactional device interactions determines a lower bound of the energy E_{min}, which is required to execute useful work with WoCA systems: For example, the observed energy for transmission on our target platform is $E_{min} = 975\,\mu J$. Consequently, the energy storage (e.g., capacitor) requires at least this amount of stored energy E_{min} to execute useful work. Thus, intermittent systems with transactional device semantics require a sufficiently large energy storage to safely execute work. Besides the offline dimensioning, WoCA requires hardware to assess the currently available state of charge. Further, our hardware supports issuing interrupts once a lower/upper energy threshold is reached, which avoids frequently polling the state of charge.

Synchronous Tasks & Asynchronous Interrupts. Our system model contains several tasks (typically less than a dozen in intermittent systems) that are synchronously activated. Besides synchronous tasks ($\tau_1, \tau_2, \ldots, \tau_n$), our model includes asynchronous interrupts ($ISR_1, ISR_2, \ldots, ISR_n$), which are often unavoidable when making use of devices. In order to bound the required time/energy resources for interrupts, WoCA requires a *minimum inter-arrival time* between two subsequent interrupts. In practical systems, determining and enforcing these timing requirements are achieved by structured interrupt activation [34]. All code for the tasks and for the interrupts are available for the static analysis, and the semantics for device accesses are unambiguous.

3 Problem Statement

Running Example. This section details the problems of safe operations in device-driven systems: the problem of transactional device uses (Section 3.1) and the handling of interrupts (Section 3.2). These problems are illustrated with a running example shown in Figure 1, where the task τ_1 activates a transceiver device `tx`, transmits, and waits for the packet to be transmitted, before switching `tx` off.

3.1 The Problem of Device Uses

Dynamic Power Demands. While the processor has a maximum power demand of $P_{max,CPU}$, the transceiver requires substantially more power, leading to $P_{max,CPU,txHigh}$. We refer to these two values as P_{low} and P_{HIGH}. Devices are often power-hungry, which is the reason for their selective activation. The assumption that P_{HIGH} is always consumed and not selectively leads to a safe $WCEC$ bound [12]. However, this approach has vast pessimism: A transceiver with exemplarily $P_{max,txHigh} = 1000\,mW$ for a single, very short active time and a CPU with $P_{max,CPU} = 1\,mW$ would lead to around 1000 x overestimation over a long runtime.

Hardware-Dependent States. We detail the problem of different device states based on the LoRa modulation [44], which we later use in our evaluation: The transmission time of LoRa packets depends on (1) the size of the payload `PL`,

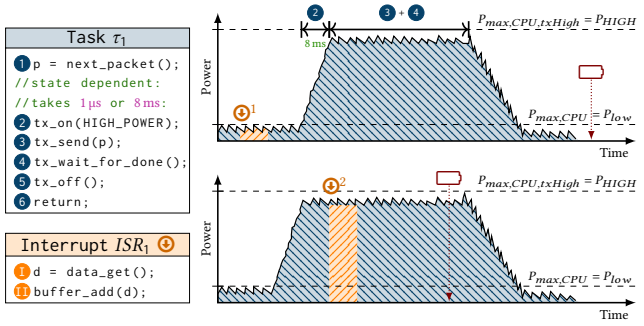

Figure 1. Intermittent systems must account for devices' transactional semantics. A power failure in transactions can lead to inconsistency. Asynchronous interrupts complicate the problem of analyzing the time/energy behavior.

(2) the bandwidth BW, and (3) the spreading factor SF. The following code extends the transmission example of Figure 1 by a configuration of the device:

```
Task(τconfig){ ... set_bandwidth(BW);
                   set_spreading_factor(SF); ... }
Task(τ1) { ... tx_send(PL); ... }
```

When statically analyzing the send operation, the current state of the transceiver device is essential: Both values of BW and SF influence the transmission time and, thereby, the operation's WCEC. Thus, a device-aware WCEC analysis requires these values at each actual call site, that is, the program path's actual context. Since most protocols can change these parameters during runtime [50], these parameters need to be available for analysis in a context-specific way. While the number of transmitted bytes is available at the call site, the other values (2) and (3) originate from prior code paths from Task(τ_{config}). To our knowledge, no WCEC technique has achieved this device-state–aware analysis.

Hardware-Dependent Transition Penalties. Changing the state of a device usually involves a non-negligible penalty in the time and, likewise, in the energy dimension. These penalties can range from few clock cycles (e.g., for writing a register) to significant execution times of hundreds of milliseconds, when major hardware reconfigurations are involved [11, 15, 35]. The device-related penalty itself is subdivided into a software-related part (e.g., access registers) and a hardware-related part, for example, waiting for a timer device to stabilize its phase-locked loop (PLL). Figure 1 illustrates such an activation penalty when transitioning the transceiver device into HIGH_POWER mode in phase 2. Phase 5 deactivates the device, also involving a certain time/energy penalty. Handling penalties becomes more complicated with consideration of the devices' context (i.e., their current state) when accessing the devices for their state transition: On the system's execution path that leads to task τ_1's execution of 2, the device can already have the required reconfiguration for the HIGH_POWER state. In this case, the transition cost of 2 is significantly smaller than actually performing the reconfiguration. Figure 1 shows this device-context–specific reconfiguration (in purple) with the temporal transition penalty either being 1 µs or, in the case of the actual hardware reconfiguration 8 ms. In other words, the software-related cost for accessing the registers remains, but the hardware-related penalty (e.g., PLL locking) is zero. Similar to the device-agnostic assumption of always using the system's maximum power P_{HIGH}, always assuming the maximum penalty causes pessimism, which WoCA avoids.

Transactional Semantics. Besides device states and hardware-dependent transition penalties, a third problem comes from the devices' transactional behavior: As illustrated in the top-right power trace, the power failure (symbol in Figure 1) interrupts the systems after the task τ_1 is completely finished, and the packet p is transmitted. However, when the power-failure interrupt hits the system right before the transmission's end, as in the bottom-right trace, the transmission is incomplete. In a benign case, a simple transmission retry is possible. However, in the worst case, the transceiver remains in an inconsistent state, with regard to its internal memory state. While device interactions in intermittent systems with DMA can circumvent transactions [28], numerous devices have transactional semantics (e.g., use of the SPI bus).

Our Approach for Devices. WoCA solves the problems of dynamic power demands, transition penalties, and transactional behaviors as follows: First, it introduces the notion of *context-dependent device states* and *device-state transitions*. Then, WoCA decomposes the code in a way that allows it to handle arbitrary device de-/activations. The decomposition is the basis for exploring all path-sensitive transition penalties.

3.2 The Problem of Interrupts

As Figure 1 shows, besides the task τ_1, an asynchronous interrupt exists in the system. This interrupt can occur exactly once while the task executes, meaning that the interrupt's minimum-inter arrival time is longer than the task's WCET. The existence of the interrupt requires in-depth considerations to eventually determine both timing and energy bounds. The time-related aspect is well-explored. The execution time of the task is prolonged by the interrupt's WCET, which leads to the worst-case response time (WCRT) of τ_1. The response time from τ_1's start to its return generally includes all interferences by higher-priority tasks or interrupts, which are highest-priority interferences from an analysis point of view. Thus, in our running example, the WCRT is composed by $WCRT(\tau_1) = WCET(\tau_1) + WCET(\text{⬇})$.

However, this additive composition does not hold for bounding τ_1's energy demand: The interrupt can occur in the P_{low} state, as shown with the ⬇¹ in the top-right trace of Figure 1. Alternatively, in the worst case with respect to energy, the interrupt ⬇² occurs in the high-power state P_{HIGH}. This case leads to the energy demand of $WCET(\text{⬇}) \cdot P_{HIGH}$, instead of $WCET(\text{⬇}) \cdot P_{low}$ for the interrupt. The bottom-right

orange area (with the upwards-facing line pattern) is larger than in the upper trace. A safe analysis technique must account for both scenarios to safely avoid the power failure during transactions, as in the bottom-right scenario.

Our Approach for Interrupts. In short, WoCA explores all system-wide program paths while tracking context-sensitive device states and power demands. Thus, WoCA analyzes tasks not only in isolation but with all possible interferences.

4 The WoCA Approach

WoCA is split into an offline/analysis part (see Section 4.1) and an online/runtime part (see Section 4.2).

4.1 WoCA's Offline Analysis

Our offline part for WoCA extends the existing approach *SysWCEC* [45]. We leverage this method for our goal of avoiding intermittency in device-driven systems with transceiver devices [20], which comes with the problem of device-state–dependent transition penalties. In the following, we present all components that enable us to determine resource bounds, being the foundation to guarantee transactional device uses.

Maximum-Flow Problems. The core principle of the worst-case analysis is translating the problem of finding the WCEC to a *maximum-flow problem* formulation. Such formulations are straightforward to solve with common mathematical solvers [5, 17]. For WCET analysis, this strategy is employed by the well-explored *implicit path enumeration technique (IPET)* [26, 31]. For WCET, the IPET determines how often the system executes a basic block (in the control-flow graph) in the WCET case. For WoCA, this technique determines how often the system executes in the WCEC case. Both techniques work on a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i : 0 \leq i < |\mathcal{V}|\}$ is the set of all nodes, and $\mathcal{E} = \{\varepsilon_i : 0 \leq i < |\mathcal{E}|\}$ describes all edges in the graph. The variable f is the execution frequency of the edge (transition) or node (state). The objective function determines the WCEC for an analyzed task, specifically $WCEC(\mathcal{G}) =$

$$max\Big(\underbrace{\Big(\sum_{v\in\mathcal{V}} WCEC(v)\cdot f(v)\Big)}_{\text{states}} + \underbrace{\Big(\sum_{\varepsilon\in\mathcal{E}} WCEC(\varepsilon)\cdot f(\varepsilon)\Big)}_{\text{transitions}}\Big)$$

In order to make the maximum-flow technique applicable to our approach of analyzing device-driven systems and device transitions, WoCA must decompose the considered systems according to this graph representation. The example system from the problem statement now serves to illustrate WoCA's decomposition, as shown in Figure 2.

System States v_i. The central part of the system's decomposition is the notion of the single states v. In order to accurately compute the worst-case energy consumption value for the state $WCEC(v_i)$, the state requires several essential components, specifically, (1) the *code* that is executed in this state (see {❶}), (2) the *power* (progress) for this state (see P_{max}), (3) the associated *state of all devices* in the system (see $\{d_i\}$). As a consequence, when one of these components changes, this marks the *terminator* for the respective state v. Figure 2 illustrates these terminators of the system graph with the ⋊ symbol in the left part. For example, the call to `tx_on(HIGH_POWER)` initiates a change of a device state in ❷ and, thus, terminates the prior system state ❶ where the device was in the `OFF` state. This decomposition is depicted in the middle part of Figure 2. The implementation of such a system-state analysis requires a mapping between function calls and respective device states along with the transition costs (see implementation details in Section 5.3). As the right part of Figure 2 outlines, multiple device graphs $\mathcal{D}_i$ may exist since the system state v_i stores a *set* of associated device states. However, for readability, the running example further uses one device graph of an example transceiver $\mathcal{D}_{\texttt{transceiver}}$ with its states $d_i \in \mathcal{D}_{\texttt{transceiver}}$.

Device States d_i. The device's graph of states and their transitions advances the state of the art in WCEC analysis. It serves as an input for WoCA's analysis. Essentially, WoCA supports devices that behave as an automaton, which holds true for common low-power transceivers of embedded systems [7, 9]. Crucial for the WCEC analysis is the precise and context- or path-sensitive handling of transition costs. Depending on the prior state, transitioning to the `ON` state either requires 1 µs or 8 ms. The system-state exploration maintains the prevailing device states for determining accurate transition costs, detailed as follows.

Exploration of System States. The decomposed code (left part in Figure 2) serves as input for the exploration of system-wide program paths under consideration of device states and potential interrupt occurrences. The exploration starts with the initial system state, after system reset, and keeps track of all state changes. Considering the running example, the interrupt might occur in the state v_1 (containing the code of ❶). Consequently, the transition to the interrupt state v_6 and back to v_1 exists in $\mathcal{G}$. The device state (d_1) and the state's maximum power (P_{low}) are propagated along the transition ε_{16} and back over ε_{61}. To differentiate transitions due to device changes and interrupts, we use the superscript ^{d} in the transition ε^d to denote a *d*evice-related system-state change. Such device-induced transitions ε^d from $\mathcal{D}$ are copied to normal transitions ε in $\mathcal{G}$ during the state exploration. For example, with the transition ε^d_{12}, the device state changes to the next state v_2 from d_1 to d_2 (i.e., `ON`). The time/energy cost for this transition stems from the device graph, and the state exploration stores this cost for the system-state graph. The exploration terminates once all system states are visited, which happens within a few minutes, given the fact that embedded systems have a manageable amount of system states. The final $\mathcal{G}$ is one essential component for formulating the maximum-flow problem together with the WCEC values.

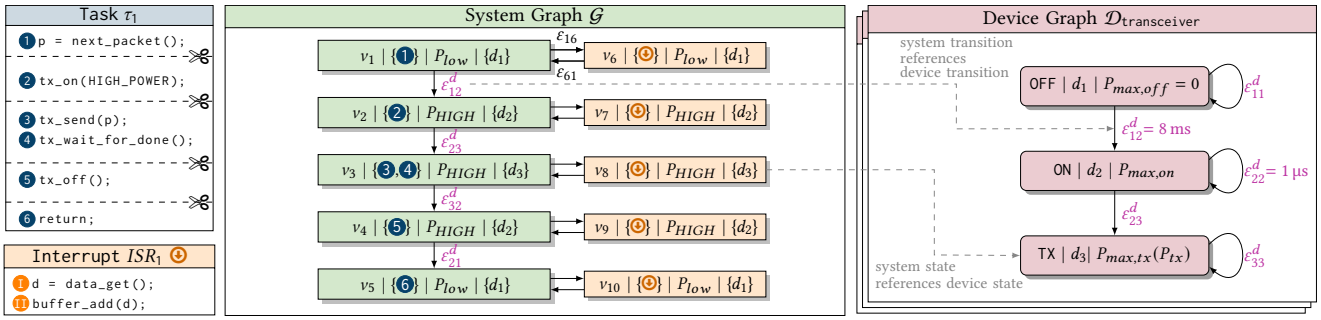

Figure 2. WoCA's analysis decomposes the system into states with device-state awareness, which is the basis for exploring the system's dynamic behavior. The device-state–aware differentiation of transitions, for example, $\varepsilon^d_{12} = 8$ ms and $\varepsilon^d_{22} = 1$ µs highlighted in purple when entering state d_2, enables WoCA to determine accurate bounds in a context-sensitive way.

Determining $WCEC(v)$ ***&*** $WCEC(\varepsilon)$. For the $WCEC$ of nodes, two main approaches exist: instruction-level energy-consumption analysis [22, 24, 29, 39] and modeling the WCEC indirectly by means of the WCET multiplied by the respective maximum power P_{max} [12, 45]. The latter approach is favorable for WoCA since the P_{max} value is tracked along the system-wide program paths. Determining this value requires knowledge of all active components: WoCA uses the device information for each state, including the CPU device, to compute a P_{max} bound since any active device contributes to this system-wide P_{max} value. A WCET analysis of the node then yields the temporal value for the $WCEC(v)$ equation. For this WCET analysis, WoCA further considers device-related worst-case times, such as the transmission duration of transceivers based on the packet length. Often, these estimates are documented in the device's manual [20] or can be found in literature, for example, for LoRa [33]. In case of lacking documentation, measurements serve as a resort for determining the transition costs $WCEC(\varepsilon)$.

Flow Constraints & Bounding Interrupts. Giving the objective formulation listed above to a mathematical solver would result in unboundedness. WoCA maps branches in the explored graph $\mathcal{G}$ to respective flow constraints for the WCEC value. As depicted in Figure 2, possible interrupts are part of the graph. To precisely bound the number of occurring interrupts, the analysis differentiates between device-related and interrupt-related transitions. Further, it adds timing-related constraints of the worst-case response time to the problem formulation. Along with the notion of each interrupt's minimum inter-arrival time and the interrupt transitions, the formulation eventually yields a sound interrupt-occurrence bound, which is reflected in the final WCEC estimate of the analyzed task.

4.2 WoCA's Online Runtime

Software Support for Task Execution. In contrast to sophisticated JIT-checkpointing techniques [4, 8, 21, 28], WoCA shifts its main complexity and engineering effort to the system's design time. As the major benefit of this strategy with worst-case awareness, the software-related runtime of WoCA is comparably straightforward. When omitting our energy/time overhead adjustments for handling the state-of-charge assessment itself, the code for safely executing a transactional code section in WoCA reads as follows:

```
Task τi = get_next_task();
if(WCEC(τi) < get_state_of_charge()) {
  τi();
}
```

Support for Task Execution. Other than the (1) system-wide WCEC estimates of tasks and (2) the checkpointing infrastructure, WoCA requires (3) hardware support to eventually decide on the code's transactional execution. This requirement of accurate state-of-charge assessment likewise holds for any JIT-checkpointing technique. Typically, a capacitor is charged with the harvested energy, and the voltage drop over the capacitor serves to determine the state of charge with an ADC. Having additional support, with a DAC connected along with the capacitor to a comparator, can serve as a lightweight signaling mechanism, also for waking the system up once sufficient energy is available to safely execute transactional workloads. Since WoCA only begins its execution when ensuring the completion of transactional operations, a minimum capacitor size for the stored energy is required respectively (e.g., $E_{min} = 975$ µJ for transmitting a LoRa packet). WoCA's awareness of the WCEC for executing the checkpoint itself eventually guarantees consistency.

5 Implementation

Our prototype implementation of the WoCA approach targets the ESP32-C3 [15], a RISC-V system-on-chip, which we integrated into custom PCBs (Section 5.1). Section 5.2 shows how we derive the energy model for the static analysis presented in Section 5.3. Finally, we detail device-context–dependent timing bounds in Section 5.4.

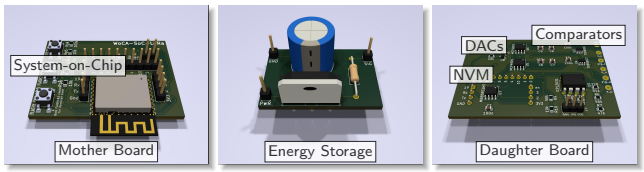

Figure 3. WoCA's circuit boards feature components for state-of-charge assessment and non-volatile memory

5.1 WoCA's Printed Circuit Boards (PCBs)

For WoCA, we developed a stack of connected PCBs that feature all requirements to execute applications under intermittent power supply safely. Figure 3 gives an overview of these components. The mother board supports the ESP32-C3 and the LoRa transceiver (bottom side of PCB). Pin headers forward signals to the daughter board, which features the necessary energy-management tooling. It allows assessing the state of charge in a polling-based way with resistors and features two DAC-comparator circuits to detect lower and upper voltage thresholds, providing a lightweight setup for receiving energy-threshold–related interrupts. A non-volatile–memory chip serves to store the system's checkpoints. Another board has the main capacitor for supplying all components of the system and a MOSFET to disconnect the power source from the capacitor for evaluation purposes.

5.2 Energy-Model Implementation

The basis of safe intermittent execution through guarantees is an accurate energy model, which is used by the analysis. Such a model has to include the consumption of all used peripheral devices. For devices without manufacturer data on the resource consumption, we resort to measurement-based worst-case modeling. We employ a JouleScope JS220 [23] energy-measurement unit, which allows the simultaneous recording of a trigger signal to identify regions of interest in the current/voltage signals. The maximum of all observed values serves as the maximum value for the energy model.

For our current prototype implementation of WoCA, we use two devices: the built-in temperature sensor of the ESP32-C3 and an RFM98 LoRa transceiver [20] connected via SPI. We evaluated the power demand of both devices in various operational modes at a steady input voltage of 3.5 V. The baseline for the energy model is the current drawn by the CPU of the ESP32-C3 at a frequency of 160 MHz. We measured the power demand of a temperature sensor for being on and during a continuous data read-out. The LoRa transceiver serves as an example of a complex device, as it has multiple operational states and device-state–specific energy demands, as previously outlined with the (context-sensitive) spreading factor and bandwidth in Section 3.1. Further, it features automatic interrupt-signaled mode changes, e.g., after sending. WoCA precisely captured this behavior in the analysis to derive sound and accurate resource-consumption bounds.

5.3 Analysis Implementation

For our static analysis, we leverage the *SysWCEC* tool [45] of the *Platin* analysis toolkit [19, 30] extended by a microarchitecture backend for the RISC-V architecture. These tools are integrated into the LLVM compiler framework [25]. While our prototype is partly based on the ESP IoT Development Framework (ESP-IDF), we implemented custom drivers for our peripherals, including GPIO and SPI, to enhance the interaction with the analysis tool.

Each device state is represented as an individual device D_i, used for the system-wide state-graph representation $\mathcal{G}$. On top of that, the LoRa transceiver has internal states, which do not directly translate to a different power state but influence the timing or energy behavior of future operations, which again underlines the necessity of WoCA's device-context sensitivity. We track these internal parameters in addition to the externally observable state for each system-graph node. All device operations are mapped to individual functions. The effects of these functions on the internal state are either derived from the function arguments or explicitly stated via annotations by the programmer [37]. Our current prototype relies on function arguments. Each function performs at most one change in device power consumption. The exception from this rule are functions that change to one state during their whole execution and return to the previous state upon return, for example, functions reading out sensor data.

For the automatic mode change after sending, the functionality is split into two functions: `start_tx` and `wait_tx`, for starting the transmission and waiting for the interrupt signaling the transmission's completion, which marks the change back to the standby mode. For each `start_tx`, a corresponding `wait_tx` must exist. This way, both programmers and compilers can derive changes in the power demand of devices from the source code in a straightforward manner.

Similar to previous analysis approaches [14, 45], we extended our compiler (clang) to construct a system graph of the given tasks and interrupts, with the enhancement of also considering device-specific contexts of observable and internal device states. The device-graph representation $\mathcal{D}_i$ of each device, which includes the function semantics for device interaction, serves the compiler as a means to identify device-state transitions in the system graph. This allows WoCA to capture the influence of devices on timing and power demand beyond state-of-the-art on/off semantics [45].

5.4 Device-Related Timing Bounds

A typical challenge in static analyses is loop bounding, usually resolved by user-provided annotations. For device interactions, such bounds may depend on hardware timings rather than program control flow. Examples include the stabilization of clock sources, transmission delays for devices connected via communication busses (e.g., SPI), or the actual duration of hardware operations, depending on the current

device configuration. Those delays can be bounded by knowledge about the hardware involved. As an example, we again consider a LoRa transmission. The maximum time on air is calculated with knowledge about the active LoRa configuration and the packet length. In WoCA, this is used to specify a bound for the loop waiting for the completion of the transmission. To ease the handling of such device-related timings, we introduce annotations for bounds in physical time (e.g., in microseconds) rather than program flow (e.g., loop-iteration bounds). Together with the processor frequency and maximum power demand, the analysis infers upper bounds from such annotations. For more complex operations with dependencies on the device state, such as the LoRa transmission, WoCA specifies the bound as a formula to allow the analysis to calculate an accurate bound for the current device context.

6 Evaluation

We describe our setup in Section 6.1. Subsequently, we present evaluations of our analysis bounds (Section 6.2), the effects of WoCA on the runtime behavior under continuous (Section 6.3), and intermittent power supply (Section 6.4) with additional focus on WoCA's starvation freedom.

6.1 Evaluation Setup

As described in Section 5.1, our evaluation target is a custom PCB equipped with an ESP32-C3 [15, 16] and a LoRa transceiver [20], complemented by a PCB for managing state-of-charge and providing NVM. We use three micro-benchmarks for our evaluations: one computational which performs the bubble sort algorithm (`bsort`), one using a device with minor complexity, namely a temperature sensor (`temp`), and one interacting with a more complex device, the LoRa transceiver, sending out one byte at the lowest output power mode (`send`). Besides micro-benchmarks, we employ one multi-device application comprising sensing, computing, and actuating phases (`sca`). This application serves as a generic case study for many (intermittently-powered) embedded systems: The sensing performs 100 readouts of the temperature sensor, the computing builds a checksum and encrypts the data, and the actuation sends out the data via LoRa in the *lowest* output power mode. The sense phase takes around 0.35 ms (0.8 %), computing 4.97 ms (11.4 %), and actuate 38.49 ms (87.8 %). Besides this (1) standard `sca`, we use two variants with interrupts: We evaluate a (2) high-frequency interrupt scenario (`sca-isr-hf`) with a minimum inter-arrival time of 5 µs and a load of around 300 ns, and (3) a low-frequency scenario (`sca-isr-lf`) with a minimum inter-arrival time of 1 ms and 200 µs of load. In both `isr` scenarios, we enforce the worst case and trigger the interrupt at its minimum inter-arrival time.

JIT-Based Implementation. For comparison, we implemented a JIT-checkpointing approach inspired by the state-of-the-art approach *Samoyed* [28]. It does not require any resource-consumption knowledge but annotations where a region with device usage begins and ends. Outside such regions, an interrupt notifies the system of a voltage level that is too low and checkpoints. Before entering a region, the runtime system always takes a checkpoint and disables checkpoints during the region to avoid inconsistencies. The programmer is responsible for placing region boundaries correctly so that semantically atomic device operations are not separated by checkpoints, which could leave devices in an unknown state. Contrary to *Samoyed*, which was implemented for a non-volatile processor, our JIT-based approach has to checkpoint global data, the stack, and the processor registers. Subsequently, we refer to this approach as *JIT-based*.

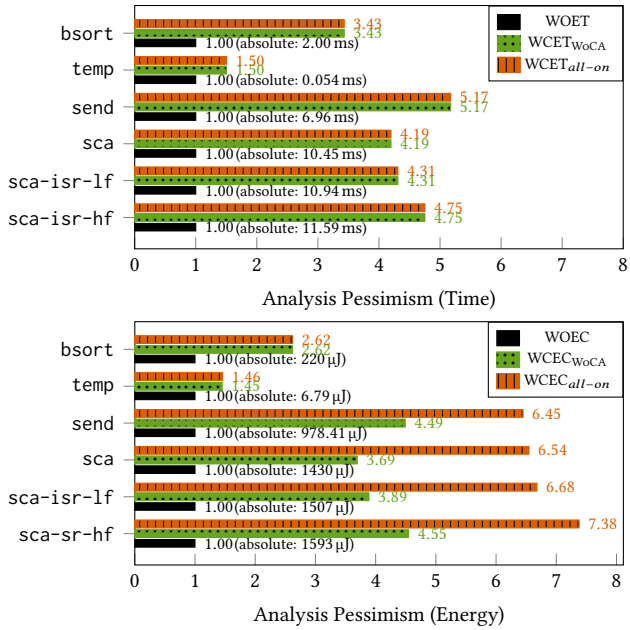

Figure 4. Analysis bounds normalized to worst-observed execution time and energy consumption

6.2 Static Analysis

Pessimism Evaluation. Any static-analysis approach is inherently only as good as the underlying models. For WoCA, two models come into play: the hardware model for the timing analysis and the energy model for the device graph. Both models are a potential source of analysis pessimism, which may affect the usability of the results. We, thus, compare the WCET and WCEC estimates from the *Platin* analyzer for the benchmarks against the worst-*observed* execution time (WOET) and the worst-*observed* energy consumption (WOEC) under a steady supply of 3.5 V. With the Gurobi 10.0.0 [17] solver, all analyses finish within 17 s. Figure 4 shows that all analysis results are upper bounds of the observed values, underlining WoCA's claim of safe resource budgeting. The pessimism factor of the WCEC bounds is comparable to that of the WCET bounds, showing that the timing-related pessimism transfers to energy bounds,

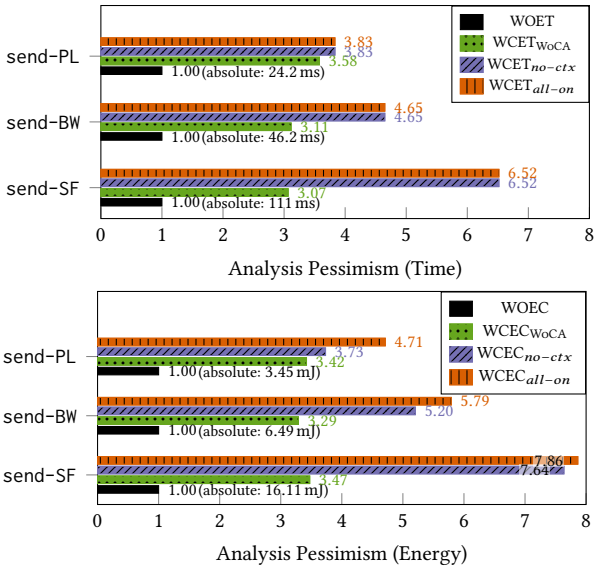

Figure 5. Influence of device states normalized to WoCA's less pessimistic bound WCEC$_{WoCA}$

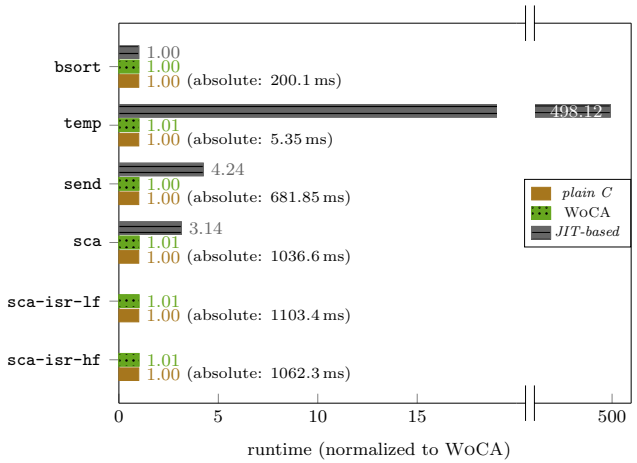

Figure 6. Runtime for 100 execs. with *continuous power*

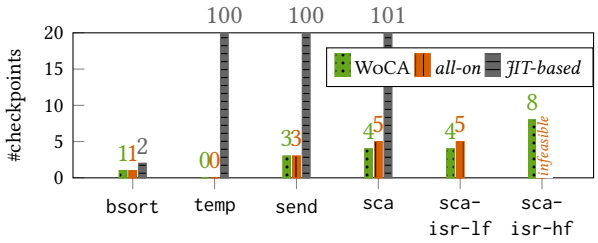

Figure 7. Checkpoints for 100 execs. on *intermittent power*

which comes from WoCA's approach of multiplying P_{max} with the WCET bound. To assess the influence of different device states, we also compute WCEC bounds for a device-agnostic approach (*all-on*), which has to assume that devices are running at full power for the complete execution. While the pessimism is the same for the WCET bounds, WoCA's context awareness leads to lower pessimism for the WCEC bounds. For the transceiver, the WCEC$_{all-on}$ bound represents the benevolent case, where it is known that the device will only send in the lowest power configuration. When this is not known, the pessimism factor increases even more, for example, to 11.7 for `send`. This again shows the necessity to consider devices' active times and power configurations.

Device-State Evaluation. In an application, devices may perform operations in different contexts, which can affect the energy demand of the device, for example, through a longer transmission time. We evaluate this effect using tasks sending with different spreading factors (`send-SF`), different bandwidths (`send-BW`), and different payload sizes (`send-PL`). As shown in Figure 5, analyzing with no knowledge about the device context (WCEC$_{no-ctx}$) yields higher pessimism than WoCA's context awareness (WCEC$_{WoCA}$). Combined with a pessimistic all-always-on approach (WCEC$_{all-on}$), which has to be used by the competing approach *RockClimb* [12], the pessimism increases even more but by a significantly smaller margin. This influence of single parameters shows the need to model device contexts beyond power states. WoCA yields accurate bounds with its notion of device states.

6.3 Continuous Power

To assess the overhead of the WoCA approach, we executed the benchmarks for 100 executions with a steady power supply of 3.5 V, measuring the response time from the start of the first execution to the end of the last. This allows us to quantify the overhead of the measures against intermittent execution (i.e., checkpoint management) in phases of sufficient energy. We compare WoCA against calling the analyzed function without any measures (*plain C*) as a baseline. Additionally, we compare against the *JIT-based* approach. As it has no notion of interrupts, we skip the interrupt benchmarks for *JIT-based*. Figure 6 shows that *JIT-based* incurs significantly more overheads. This stems from the fact that such worst-case–agnostic approaches, like *Samoyed* [28], have to checkpoint before each execution phase with device interactions. This effect becomes worse with larger checkpoint sizes and shorter device interactions, as can be seen with `temp`. WoCA only needs to query the current state of charge before execution, leading to a reduced degree of overhead. In summary, this reduced resource overhead is possible with WoCA's use of device-aware WCEC bounds.

6.4 Intermittent Power

To evaluate the behavior under intermittent execution, we equip the system with a 12 mF capacitor as an energy buffer and use a separate microcontroller (i.e., Arduino Nano) to disrupt the power supply as described in Section 5.1 in a periodic pattern of 100 ms charging and 500 ms discharging. Our runtime system collects the number of checkpoints. We also record the start and end of experiments by tracing the output of a GPIO pin using the JouleScope JS220 [23].

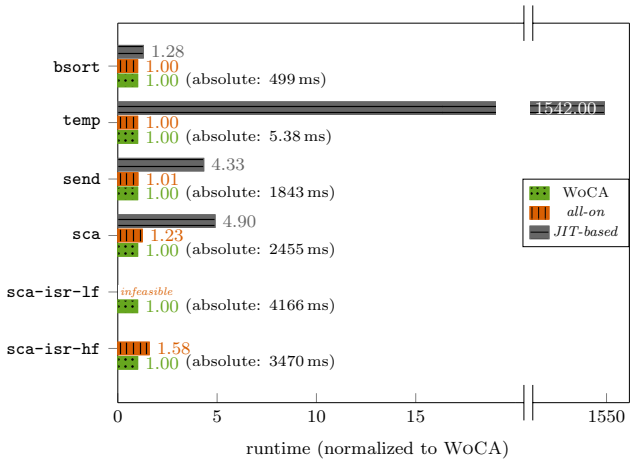

Figure 8. Runtime for 100 execs. with *intermittent power*

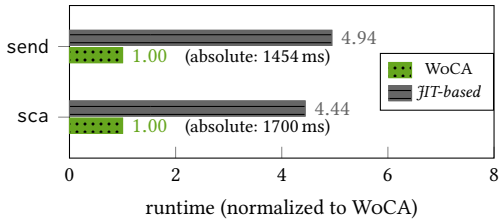

Figure 9. Runtime for 100 execs. with *short-active intermittent power* demonstrating WoCA's starvation freedom

We compare WoCA to a device-agnostic variant by using the WCEC_{all-on} values derived in Section 6.2. As device-agnostic approaches, such as *RockClimb* [12], have no knowledge of the selective device usage of tasks, they have to rely on more pessimistic bounds. We also compare WoCA to the *JIT-based* approach. Figure 7 shows the number of checkpoints, Figure 8 the runtime for WoCA, *all-on*, and *JIT-based* observed for 100 executions, again skipping the interrupt benchmarks for *JIT-based*. We consistently observed fewer checkpoints for WoCA compared to *JIT-based*. By design, *JIT-based* requires a checkpoint before each device use, leading to a notably larger number of checkpoints, especially for `temp`, which is short enough to fit completely into the charging period. This indicates that WoCA is able to avoid checkpointing due to its notion of resource bounds and the subsequent execution guarantees. With the case study (`sca`), the advantage of device awareness is visible as WoCA requires fewer checkpoints than *all-on*. The need to checkpoint before each device use leads to significantly larger runtimes for *JIT-based*. For the case study (`sca`), WoCA achieves shorter runtimes than *all-on*, as WoCA's more accurate bounds allow shorter charging times.

Another effect of WoCA's more accurate bounds shows for `sca-isr-lf`, as it is not safely executable with the pessimistic bound of *all-on* and the used capacitor. Dimensioning the capacitor and choosing the energy level at which the system wakes up and starts executing is a crucial design parameter in intermittent systems. Waking up too early risks wasting energy for checkpointing and unnecessary re-execution, leading to starvation in the worst case. Waiting for too much energy misses opportunities to execute meaningful work in a timely manner.

The two extremes of the spectrum of potential wakeup points are (at the upper end) a fully charged energy storage and (at the lower end) sufficient energy to power the system and perform a checkpoint. Especially for larger energy storage with accordingly large charging times, a fully charged buffer may rarely be reached, depending on the environmental conditions of the harvesting source. WoCA's energy bounds help to derive a sensible wakeup point, where it is guaranteed that completely executing a task is possible.

The actual system behavior is highly dependent on the energy-harvesting conditions and the system's configured wakeup charge. To validate this statement, we examine a scenario where execution benefits from WoCA's energy bounds for the two transceiver-using benchmarks (`send`, `sca`) as the most prominent examples for the effect. To simulate unstable energy-harvesting conditions, we toggle the connection of the power source to have a short active time in microsecond intervals (1 µs on, 10 µs off) for a total duration of 6 s; afterwards, the power source is constantly connected. Under these power-supply conditions, we compare the *JIT-based* approach against WoCA. Due to lack of knowledge about the energy consumption of tasks, the *JIT-based* approach can start execution either as soon as sufficient energy for creating a potential checkpoint is available or when the energy storage is fully charged. As the fully-charged state is never reached during the unstable phase, Figure 9 compares the runtime for the first 100 executions of the `send` and `sca` benchmark for the *JIT-based approach* with the low-energy wakeup level and WoCA with its derived energy bounds as the wakeup level. The *JIT-based* approach fails to make progress during the unstable conditions, only finishing the first 100 executions when the power supply becomes stable (i.e., after 6 seconds). During the unstable phase, *JIT-based* faces starvation. WoCA, on the other hand, benefits from its energy bounds, does not waste energy for re-execution, and – due to its power-failure freedom – is able to make progress and finish the first 100 executions during the unstable conditions.

7 Related Work

The body of related work on intermittent systems has substantially increased over the last decade. However, to our knowledge, WoCA is the first approach that achieves guaranteed execution of transactional operations with its whole-system analysis, its handling of asynchronous interrupts, its accounting for arbitrary device states, along with their context-sensitive transition penalties.

Samoyed [28] uses JIT-checkpointing and is related to WoCA due to its peripheral handling. *Samoyed*'s support for devices consists of an energy-profiling phase. Inherent to profiling-based approaches is their lack of providing resource-consumption bounds, which is in contrast to WoCA. Further, *Samoyed* has a notion of how to split device accesses into smaller workloads when execution attempts fail. However, this scaling of workloads only works for memory-oriented device accesses, such as DMA interfaces. In contrast, WoCA supports devices, such as transmitters, that have unavoidable transactional semantics for their service.

An approach that relies on worst-case analyses is *RockClimb* [12]. It has the notion of a safe active time and uses this information to guarantee safely reaching a subsequent checkpoint. That way, *RockClimb* achieves rollback freedom. For computing the safe active time, *RockClimb* conducts a WCET analysis and combines this with the highest power mode of the system. This device-unselective (i.e., all-always-on) approach has the problem of yielding overly pessimistic worst-case energy-consumption estimates. WoCA avoids this pessimism through its context-sensitive handling of device states and their transition penalties.

A further approach that relies on WCEC estimates was presented by Yarahmadi et al. [51]: This approach determines WCEC values for basic blocks and uses this information in order to decide whether to safely reach a checkpoint without facing a power-failure interrupt. Thereby, these runtime guarantees further lead to the property of having guaranteed forward progress with the available, harvested energy. Common to WoCA is the guaranteed forward progress by means of static worst-case analysis. However, in addition to their work, we emphasize the necessity to accurately model interactions of the system with devices since they are the most power-hungry components in the system. *EPIC* motivates the variability in power demand of intermittent systems and proposes a respective energy model [2]. In contrast to this work, WoCA is able to capture devices and statically provide system-wide worst-case bounds for the energy demand.

Pudu [36] shares our view that devices are of major importance in intermittent systems. *Pudu* uses the notion of *energy types* to describe the various states of peripheral devices. In contrast to WoCA, *Pudu* is not able to conduct worst-case analyses with an underlying formal model that captures all possible program flows. Thus, *Pudu* is unable to give runtime guarantees, which are necessary for many device-driven intermittent systems with guaranteed forward progress. WoCA's notion of the worst-case energy behavior by means of maximum-flow techniques includes context-sensitive device states to reduce analysis pessimism. Similar to *Pudu*, Berthou et al. give a formal specification for intermittent computing with peripherals [6]. In contrast to both works, WoCA is able to give forward-progress guarantees by means of device-aware worst-case analysis.

8 Conclusion

Existing intermittent-computing approaches have shortcomings with regard to the transactional semantics of devices, which can lead to starvation. With our WoCA approach, we leverage whole-system WCEC analyses to guarantee the transactional execution of operations under given energy budgets. The employed analysis of WoCA keeps track of all possible system paths and the respective device states. WoCA's awareness of context-sensitive program paths avoids analysis pessimism. Besides runtime guarantees, the WoCA approach facilitates checkpointing and thereby avoids runtime overheads. Our evaluations on WoCA's hardware show that WoCA uses the harvested energy more efficiently while providing forward guarantees. As other researchers [38], we share the concern that future trillion IoT devices could lead to trillion dead batteries, which remain unrecycled. With WoCA, we want to contribute with our safe execution to more sustainable and reliable battery-free IoT systems.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 502615015, ResPECT; project number 502947440, Watwa.

WoCA's hard- & software are publicly available:
gitos.rrze.fau.de/woca

References

[1] A. Cohen et al. 2018. *Inter-Disciplinary Research Challenges in Computer Systems for the 2020s.* Technical Report. NSF, USA.

[2] Saad Ahmed, Abu Bakar, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. The betrayal of constant power × time: finding the missing Joules of transiently-powered computers. In *Proc. of LCTES '19*. 97–109.

[3] Saad Ahmed, Bashima Islam, Kasim Sinan Yildirim, Marco Zimmerling, Przemysław Pawełczak, Muhammad Hamad Alizai, Brandon Lucia, Luca Mottola, Jacob Sorber, and Josiah Hester. 2024. The Internet of Batteryless Things. *Commun. ACM* 67, 3 (2024), 64–73.

[4] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 35, 12 (2016), 1968–1980.

[5] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. 2004. lp_solve 5.5, open source (mixed-integer) linear programming system. Software.

[6] Gautier Berthou, Pierre-Évariste Dagand, Delphine Demange, Rémi Oudin, and Tanguy Risset. 2020. Intermittent computing with peripherals, formally verified. In *Proc. of LCTES '20*. 85–96.

[7] Gautier Berthou, Kevin Marquet, Tanguy Risset, and Guillaume Salagnac. 2020. Accurate power consumption evaluation for peripherals in ultra low-power embedded systems. In *Proc. of GIoTS '20*. 1–6.

[8] Naveed Anwar Bhatti and Luca Mottola. 2017. HarvOS: Efficient code instrumentation for transiently-powered embedded sensing. In *Proc. of IPSN '17*. 209–219.

[9] Markus Buschhoff, Robert Falkenberg, and Olaf Spinczyk. 2019. Energy-aware device drivers for embedded operating systems. *ACM SIGBED Review* 16, 3 (2019), 8–13.

[10] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. 2017. Determining Application-Specific Peak Power and Energy Requirements for Ultra-Low-Power Processors. *ACM Trans. on Computer Systems (ACM TOCS)* 35, 3 (2017), 9:1–9:33.

[11] Holly Chiang, Hudson Ayers, Daniel Giffin, Amit Levy, and Philip Levis. 2021. Power Clocks: Dynamic Multi-Clock Management for Embedded Systems. In *Proc. of EWSN '21*. 139–150.

[12] Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. 2022. Compiler-Directed High-Performance Intermittent Computation with Power Failure Immunity. In *Proc. of RTAS '22*. 40–54.

[13] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL '77*. 238–252.

[14] Gerion Entrup, Benedikt Steinmeier, and Christian Dietrich. 2019. ARA: Automatic Instance-Level Analysis in Real-Time Systems. In *Proc. of OSPERT '19*. 7–16.

[15] Espressif Systems. 2022. *ESP32-C3 Series Datasheet*.

[16] Espressif Systems. 2022. *ESP32-C3 Technical Reference Manual*.

[17] Gurobi Optimization, LLC. 2023. Reference Manual. gurobi.com/.

[18] Sebastian Hahn, Jan Reineke, and Reinhard Wilhelm. 2015. Towards Compositionality in Execution Time Analysis: Definition and Challenges. *ACM SIGBED Review* 12, 1 (2015), 28–36.

[19] Stefan Hepp, Benedikt Huber, Daniel Prokesch, and Peter Puschner. 2015. The platin Tool Kit – The T-CREST Approach for Compiler and WCET Integration. In *Proc. of KPS '15*. 277–292.

[20] Hope Microelectronics. 2013. *RFM95/96/97/98(W) – Low Power Long Range Tranceiver Module*.

[21] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. 2014. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In *Proc. of VLSID '14*. 330–335.

[22] R. Jayaseelan, T. Mitra, and X. Li. 2006. Estimating the Worst-Case Energy Consumption of Embedded Software. In *Proc. of RTAS '06*. 81–90.

[23] Jetperch LLC. [n. d.]. *Joulescope JS220 User's Guide Precision DC Energy Analyzer*.

[24] Steve Kerrison and Kerstin Eder. 2015. Energy Modeling of Software for a Hardware Multithreaded Embedded Microprocessor. *ACM Trans. on Embedded Computing Systems (ACM TECS)* 14, 3 (2015), 56.

[25] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. of CGO '04*. 75–86.

[26] Yau-Tsun Steven Li and Sharad Malik. 1995. Performance Analysis of Embedded Software Using Implicit Path Enumeration. In *ACM SIGPLAN Notices*, Vol. 30. ACM.

[27] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *Proc. of SNAPL '17*, Vol. 71. 8:1–8:14.

[28] Kiwan Maeng and Brandon Lucia. 2019. Supporting Peripherals in Intermittent Systems with Just-in-time Checkpoints. In *Proc. of PLDI '19*. 1101–1116.

[29] J. Pallister, S. Kerrison, J. Morse, and K. Eder. 2017. Data Dependent Energy Modeling for Worst Case Energy Consumption Analysis. In *Proc. of SCOPES '17*.

[30] P. Puschner, D. Prokesch, B. Huber, J. Knoop, S. Hepp, and G. Gebhard. 2013. The T-CREST Approach of Compiler and WCET-Analysis Integration. In *Proc. of ISORC '13*. 1–8.

[31] P. Puschner and A. Schedl. 1997. Computing Maximum Task Execution Times: A Graph-Based Approach. *Real-Time Systems* 13 (1997), 67–91.

[32] Phillip Raffeck, Christian Eichler, Peter Wägemann, and Wolfgang Schröder-Preikschat. 2019. Worst-Case Energy-Consumption Analysis by Microarchitecture-Aware Timing Analysis for Device-Driven Cyber-Physical Systems. In *Proc. of WCET '19*. 6:1–6:12.

[33] Ceferino Gabriel Ramirez, Anton Sergeyev, Assya Dyussenova, and Bob Iannucci. 2019. LongShoT: Long-Range Synchronization of Time. In *Proc. of IPSN '19*. 289–300.

[34] John Regehr and Usit Duongsaa. 2005. Preventing interrupt overload. In *Proc. of LCTES '05*. 50–58.

[35] Michel Rottleuthner, Thomas C. Schmidt, and Matthias Wahlisch. 2023. Dynamic Clock Reconfiguration for the Constrained IoT and Its Application to Energy-Efficient Networking. In *Proc. of EWSN '22*. 168–179.

[36] Emily Ruppel. 2022. *Peripheral and Power Management in Battery-less, Energy-Harvesting Systems*. Ph. D. Dissertation. Carnegie Mellon University.

[37] Simon Schuster, Peter Wägemann, Peter Ulbrich, and Wolfgang Schröder-Preikschat. 2021. Annotate Once – Analyze Anywhere: Context-Aware WCET Analysis by User-Defined Abstractions. In *Proc. of LCTES '21*.

[38] Esther Shein. 2021. A battery-free internet of things. *Commun. ACM* 64, 7 (2021), 16–18.

[39] 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*.

[40] Philip Sparks. 2017. White paper: The economics of a trillion connected devices.

[41] Texas Instruments Inc. 2018. *MSP430FR57xx Family*.

[42] D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla. 2019. Worst-Case Energy Consumption: A New Challenge for Battery-Powered Critical Devices. *IEEE Trans. on Sustainable Computing* (2019), 1–8.

[43] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent computation without hardware support or programmer intervention. In *Proc. of OSDI '16*. 17–32.

[44] Lorenzo Vangelista. 2017. Frequency shift chirp modulation: The LoRa modulation. *IEEE Signal Processing Letters* 24, 12 (2017), 1818–1821.

[45] Peter Wägemann, Christian Dietrich, Tobias Distler, Peter Ulbrich, and Wolfgang Schröder-Preikschat. 2018. Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems. In *Proc. of ECRTS '18*, Vol. 106. 24:1–24:25.

[46] Peter Wägemann, Tobias Distler, Timo Hönig, Heiko Janker, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. 2015. Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. In *Proc. of ECRTS '15*. 105–114.

[47] Simon Wegener, Kris K. Nikov, Jose Nunez-Yanez, and Kerstin Eder. 2023. EnergyAnalyzer: Using Static WCET Analysis Techniques to Estimate the Energy Consumption of Embedded Applications. In *Proc. of WCET '23*. 9:1–9:14.

[48] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The Worst-case Execution-time Problem – Overview of Methods and Survey of Tools. *ACM Trans. on Embedded Computing Systems (ACM TECS)* 7, 3 (2008), 1–53.

[49] Harrison Williams, Xun Jian, and Matthew Hicks. 2020. Forget Failure: Exploiting SRAM Data Remanence for Low-Overhead Intermittent Computation. In *Proc. of ASPLOS '20*.

[50] Weitao Xu, Jun Young Kim, Walter Huang, Salil S Kanhere, Sanjay K Jha, and Wen Hu. 2019. Measurement, characterization, and modeling of LoRa technology in multifloor buildings. *IEEE Internet of Things Journal* 7, 1 (2019), 298–310.

[51] Bahram Yarahmadi and Erven Rohou. 2020. Compiler optimizations for safe insertion of checkpoints in intermittently powered systems. In *Proc. of SAMOS '20*. 169–185.

[52] Eren Yildiz, Saad Ahmed, Bashima Islam, Josiah Hester, and Kasim Sinan Yildirim. 2023. Efficient and Safe I/O Operations for Intermittent Systems. In *Proc. of EuroSys '23*.

Received 2024-02-29; accepted 2024-04-01