CRÊPE: Clock-Reconfiguration–Aware Preemption Control in Real-Time Systems with Devices

Eva Dengler
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Peter Wägemann
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

The domain of energy-constrained real-time systems that are operated on modern embedded system-on-chip (SoC) platforms brings numerous novel challenges for optimal resource minimization. These modern hardware platforms offer a heterogeneous variety of features to configure the tradeoff between temporal performance and energy efficiency, which goes beyond the state-of-the-art of existing dynamic-voltage-frequency-scaling (DVFS) scheduling schemes. The control center for configuring this tradeoff on platforms are complex clock subsystems that are intertwined with requirements of the SoC's components (e.g., transceiver/memory/sensor devices). That is, several devices have precedence constraints with respect to specific clock sources and their settings. The challenge of dynamically adapting the various clock sources to select resource-optimal configurations becomes especially challenging in the presence of asynchronous preemptions, which are inherent to systems that use devices.

In this paper, we present CRÊPE, an approach to clock-reconfiguration–aware preemption control: CRÊPE has an understanding of the target platform's clock subsystem, its sleep states, and penalties to reconfigure clock sources for adapting clock frequencies. CRÊPE's hardware model is combined with an awareness of the application's device requirements for each executed task, as well as possible interrupts that cause preemptions during runtime. Using these software/hardware constraints, CRÊPE employs, in its offline phase, a mathematical formalization in order to select energy-minimal configurations while meeting given deadlines. This optimizing formalization, processed by standard mathematical solver tools, accounts for potentially occurring interrupts and the respective clock reconfigurations, which are then forwarded as alternative schedules to CRÊPE's runtime system. During runtime, the dispatcher assesses these offline-determined alternative schedules and reconfigures the clock sources for energy minimization. We developed an implementation based on a widely-used SoC platform (i.e., ESP32-C3) and an automated testbed for comprehensive energy-consumption evaluations to validate CRÊPE's claim of selecting resource-optimal settings under worst-case considerations.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded systems

Keywords and phrases energy-constrained real-time systems, time/energy tradeoff, system-on-chip, energy-aware real-time scheduling, resource minimization, preemption control, worst-case energy consumption (WCEC), worst-case execution time (WCET), static whole-system analysis

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2024.10

Supplementary Material *Software (ECRTS 2024 Artifact Evaluation approved artifact)*: `https://doi.org/10.4230/DARTS.10.1.2`
Software (Source Code): `https://gitos.rrze.fau.de/crepe/ecrts24-artifact-evaluation`
archived at `swh:1:dir:7565661d0acdbb0e01393213fdc6cf880a0de2c4`
Software (Hardware Description): `https://gitos.rrze.fau.de/crepe/hardware`
archived at `swh:1:dir:07450b5fd19c531c16e70d5af98748b65bb260c2`

Funding This work is funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under the project number 502947440 (WA 5186/1-1, Watwa: Whole-System Optimality Analysis and Tailoring of Worst-Case–Constrained Applications).

1 Introduction

Time- & Energy-Constrained Applications. The increasing number of embedded (IoT) devices [57] comes with several novel challenges for scheduling under resource constraints: On the one hand, these systems can have real-time requirements for the timeliness of results. On the other hand, they are often battery-operated or even harvest their energy from the environment as part of the Internet of Batteryless Things [1]. For their proper operation, these energy-harvesting systems have to provide results under available energy budgets. Some of these devices are embedded within uncritical environments (e.g., consumer electronics). However, also highly safety-critical devices, such as implantable medical devices, exist [40], which are the main focus of the rest of this paper. Such safety-critical devices require provable runtime guarantees for both the time and energy dimensions.

Runtime Guarantees by Static Analysis. Giving runtime guarantees for the execution within time/energy budgets is possible by utilizing static analysis tools for the application's tasks: Thereby, the analysis tool builds an abstraction of the tasks' execution paths and combines this information with a resource-consumption model of the target hardware platform. For the two dimensions of time and energy, numerous analyzers exist for the *worst-case execution time* [6, 20, 21, 24, 26, 29, 33, 35, 38, 39, 50], as well as for the *worst-case energy consumption (WCEC)* [31, 48, 51, 60, 61, 62, 64]. By employing these WCET and WCEC values, schedulers can determine sequences of job executions that are optimal (i.e., minimal) with respect to energy consumption under given real-time constraints. We refer to such sequences as *worst-case optimal solutions*.

Energy-Aware Real-Time Systems with Devices. A vast body of literature on energy-aware real-time scheduling exists, and we refer to the comprehensive survey article of Bambagini et al. [7]. Of particular interest for the presented paper are scheduling schemes that consider *devices* [13, 66, 67]. In this paper, we employ a generic notion for such devices: Any component in the system that (1) can be switched on/off, (2) can be parameterized (e.g., speed scaling), and (3) consumes energy (i.e., power over time) is treated as a device. That is, even the main processor is treated as a device. Further devices include sensors (i.e., all analog-to-digital converters: temperature, air quality), actuators, co-processors (e.g., for cryptographic operations), and especially transceivers. We assume that many devices in energy-constrained (e.g., battery-operated) systems are more significant with respect to power demand than the processor (device). While scheduling approaches exist for handling devices and their respective power states [13, 66, 67], they have shortcomings with regard to modern embedded system-on-chip (SoC) platforms. The novel characteristics of these SoC platforms are that they feature numerous (integrated) devices and support a sophisticated clock subsystem with reconfiguration options for all available clocks [15, 17, 54].

Time-Energy Tradeoff on Modern Clock Subsystems. The complexity of these clock subsystems goes beyond classic dynamic-voltage-frequency-scaling (DVFS) schemes [7], where usually only one clock source is configurable, and no further complex device constraints exist. As we will detail, modern SoCs have a variety of clock sources with different temporal performance and energy-efficiency characteristics to configure the time-energy tradeoff. Besides this time-energy tradeoff, end devices (such as transceivers) on SoCs can require specific clock settings as a precedence constraint for their service, which also needs to be taken into account for energy-aware real-time scheduling. Furthermore, reconfiguring clocks can involve non-negligible penalties (both time- and energy-wise) that can range from a few cycles up to hundreds of milliseconds [19, 54]. Such penalties, for example, originate from the hardware characteristics that clocks are not reconfigurable in place. That is, the SoC requires an intermediate clock source while reconfiguring the original clock. After that, the system can switch back to the reconfigured, original clock source.

Presence of Interrupts. Recent works on *Power Clocks* [15], *ScaleClock* [54], and *FusionClock* [17] underline the need to account for the clock subsystem in modern embedded systems. However, all these works miss an essential property that arguably all embedded real-time systems that interact with the environment inherit: *the presence of interrupts*. Many embedded devices (e.g., sensors, transceivers) communicate with the CPU via interrupts. Once an interrupt request is released by a device, for example, when a new packet is available and interrupts are activated, the CPU preempts the currently running task to serve the interrupt via the interrupt-service routine (ISR). The code for serving the interrupt usually involves communicating (via busses) with the device and is thus not negligible. Such preemptions are especially expensive in terms of energy when the system is executing in a high-power state once the interrupt occurs. As a consequence thereof, time- and energy-aware scheduling strategies need to address and control these interrupt-induced preemptions.

Contributions. In this paper, we present CRÊPE, an approach for clock-reconfiguration–aware preemption control. To the best of our knowledge, CRÊPE is the first to address preemption control in energy-aware real-time systems under consideration of modern clock subsystems on embedded SoCs. The main goal of CRÊPE is to find optimal sequences of clock reconfigurations in the presence of interrupts under consideration of worst-case assumptions for both time and energy. In summary, we propose the following three contributions:

1. *Offline Formalization, Scheduling, & Optimization*: This paper introduces a mathematical formalization to identify decisions and control preemptions that are worst-case optimal with regard to energy under real-time constraints. One aspect of this contribution is a discussion on the shortcomings of existing, widely-used frequency-centric power models in view of the complexity of modern embedded SoCs.
2. *Online Preemption Control & Dispatching*: Based on the possible worst-case optimal schedules with alternative execution paths identified offline during design time, CRÊPE forwards schedule tables to an online dispatcher. CRÊPE's dispatcher controls preemptions and configures worst-case optimal clock configurations for each task and interrupts during runtime.
3. *Testbed & Evaluation*: We implemented our publicly-available approach based on a real-world SoC platform that features a complex clock subsystem (i.e., the RISC-V–based platform ESP32-C3). Further, we developed a surrounding testbed that allows us to automate energy-consumption measurements to conduct a representative number of task sets in our evaluations. These evaluations account for all constrained resources on our target, precisely time, energy, and memory for CRÊPE's employed schedule tables.

CRÊPE is based on and extends the *FusionClock* approach for clock-aware scheduling [17]. In contrast to this work, CRÊPE has a different task-set model due to targeting preemption control for real-time systems with interrupts. This leads to both changes in the offline optimization as well as the online dispatching: The offline optimization automatically generates variants of schedule tables with awareness of possible interrupt occurrences, while the online dispatching of CRÊPE makes use of the previously built schedules.

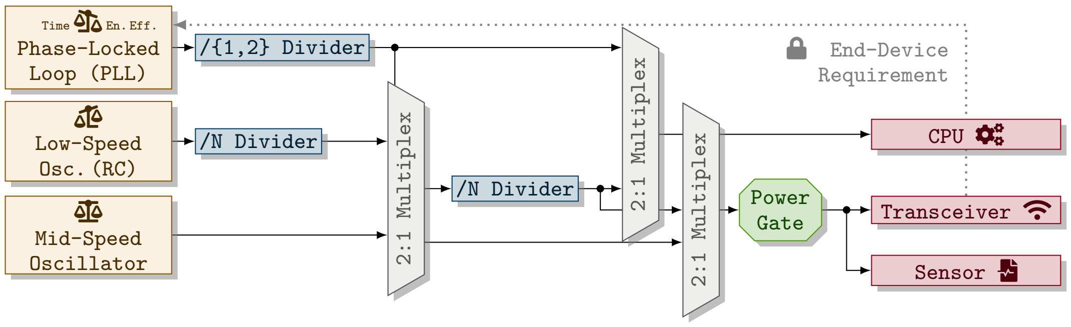


Figure 1 The possibilities to configure the system's clock and reconfigure it during runtime are manifolds on modern embedded SoCs. Configuring the clock source, dividers, multiplexers, and clock/power gates allows for balancing between the temporal performance and energy efficiency.

2 Background & System Model

Reconfiguring Clock Sources. We target highly resource-constrained embedded devices with less than one megabyte of RAM, execution speeds up to 200 MHz, and sleep power down to the µW range. For such SoCs, the ESP32-C3 serves as a representative platform [19]. On these systems, the heart for configuring the time-energy tradeoff is the *clock subsystem*: Figure 1 gives a schematic excerpt of such a subsystem [19, Figure 6.2] from this platform, which is later used in the evaluation in Section 7. The actual clock subsystem, which is synonymously also referred to as a *clock-distribution network* or *clock tree*, has more than the depicted configuration options and clock sources. Besides speed and energy efficiency, clock sources differ in temperature stability and temporal accuracy. Further, some end devices (leaf nodes in the overall network) have distinct requirements for the clock configuration (illustrated with the 🔒 symbol): For example, the WiFi transceiver only runs with the phase-locked loop (PLL). This clock, in turn, only has two speed settings (divider 1 or 2), while other clock sources have more discrete frequencies (dividers from 1 to 1024). Routing the signal from source to end devices happens via multiplexers, while power gates enable the system to entirely switch off parts of power-consuming devices, including their leakage power.

Device, Temporal, & Power Behavior. Each component on the system that consumes power –either by leakage or switching activities– is treated as a *device*. Devices have multiple possible states (e.g., active, idle, off). As part of our system model, we describe the *state* of the system, including all devices, with one *clock configuration* (CC). Each CC has an assigned *maximum power demand* P_{max}. For the execution of jobs J_i of a task τ_i in one CC, we determine a WCET in CRÊPE. We assume that the property of compositionality exists for energy and time [25, 49], excluding the presence of timing anomalies. We consider these assumptions to be realistic since they exemplarily hold for CRÊPE's RISC-V–based target platform. With the P_{max} value and the WCET, we can determine an upper bound of the energy demand $WCEC = P_{max} \cdot WCET$. Refinements for this temporal model are possible in two directions: (1) employing instruction-level energy models for the WCEC value [12, 34, 37, 45, 48, 55, 56, 59, 64], or (2) refining the P_{max} estimate by application-specific knowledge [14]. However, this work focuses on assessing optimal energy savings with preemption control and awareness of feasible CCs under worst-case assumptions.

Penalties for Clock-Configuration Transitions. The transition between clock configurations $c_N \rightarrow c_M$ requires special attention as the associated time and energy penalties are not negligible: When executing with one clock source, adapting the frequency can require an intermediate clock source when the frequency can not be changed in place, causing significant overheads. A further aspect of transition penalties is the *context sensitivity*: The penalty for changing to the new configuration c_M depends on the previously active setting c_N.

Sleep Modes. Crucial in energy-constrained systems is the use of the CPU's sleep modes because these modes can reduce the power demand by multiple orders of magnitude. Our generic notion of *"everything is a device"* and the unambiguous description of system states by *CC*s already integrate an understanding of sleep modes: In sleep mode, the CPU device is switched off, and other devices are kept active (i.e., I/O pins for waking up the system). Leaving a sleep mode and entering a CPU's run mode usually requires a significant transition penalty: For example, when exiting from a low-power deep sleep mode, with a P_{max} in the µW range, the transition involves a penalty of around 70 ms on Crêpe's target platform.

Task Model. We assume a set of periodic tasks τ_i and a preempting interrupt service routine ISR in our task model. The jobs $J_{i,j}$ of the tasks τ_i and interrupts are executed on a single-core processor with numerous other devices. The power-related behavior (i.e., *CC* changes) is explicitly controlled by jobs/interrupts. The periodicity of the periodic tasks is defined by p_i, while the interrupt has a minimum inter-arrival time i_{ISR}. The WCET of each task is smaller than the minimum inter-arrival time of each interrupt plus the ISR execution; if this assumption is not fulfilled, long-running tasks must be split. This splitting is comparable to dividing tasks into subjobs, as common in limited preemptive scheduling [11]. The set of tasks/interrupts is referred to as T. All elements in T are independent of each other, except for the subsequent clock configuration. The total task set and the interrupt have a relative deadline D_i. Further, each task/interrupt uses at least one device: In any case, they require the CPU device to execute instructions. Both tasks/interrupts can require specific devices that, in turn, require specific clock configurations. Crêpe has an understanding of the set of feasible *CC*s for each device. Crêpe knows how to transition from any clock configuration to a target configuration (which may involve multiple intermediate *CC*s).

Controlling Interrupt Preemptions. We argue that all realistic embedded systems that acquire data via sensors and communicate results face the challenge of interrupts. While serving interrupt requests immediately, when the interrupt (e.g., level change of a pin) occurs, minimizes the request's response time, this strategy has numerous shortcomings given the induced time and energy overhead: In most hardware architectures, the interrupt is directly executed on the memory stack of the currently running job. This execution comes with the problem of cache-related preemption delays [2, 10, 36] because interrupts share the instruction and/or data cache with the executed job. In addition to the problem of cache-related preemption delays, Crêpe is confronted with the problem that the interrupt is executed with the currently active clock configuration and thus power demand, an aspect which we will further detail in Section 3.1. For Crêpe, we require the possibility to control interrupt preemptions [53] (i.e., defer pending interrupt requests) as part of an online

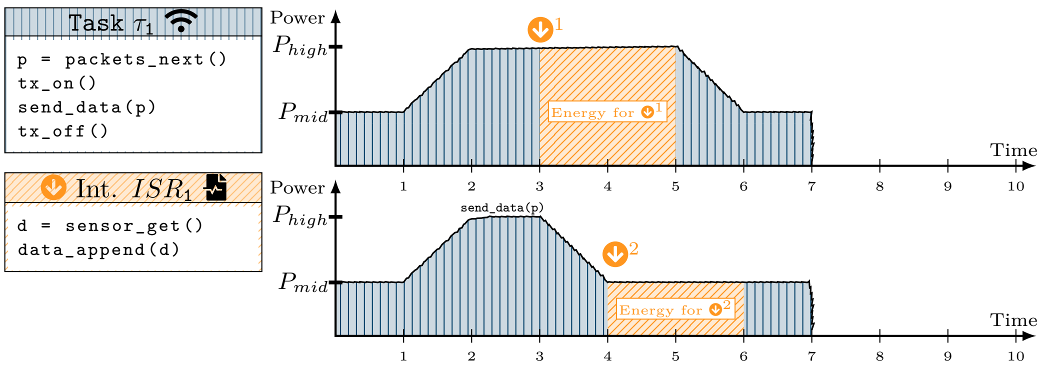


Figure 2 When immediately executing interrupts without control, the associated service routine can run in a comparably high power state, leading to energy inefficiency.

dispatcher unit[1]. This preemption control includes that interrupts are temporarily blocked: When interrupts occur while their service is blocked, the associated service routines are deferred until the interrupt is re-activated so that its ISR gets executed. Non-maskable interrupts are uncommon in the domain of embedded systems and, thus, beyond the scope of our system model.

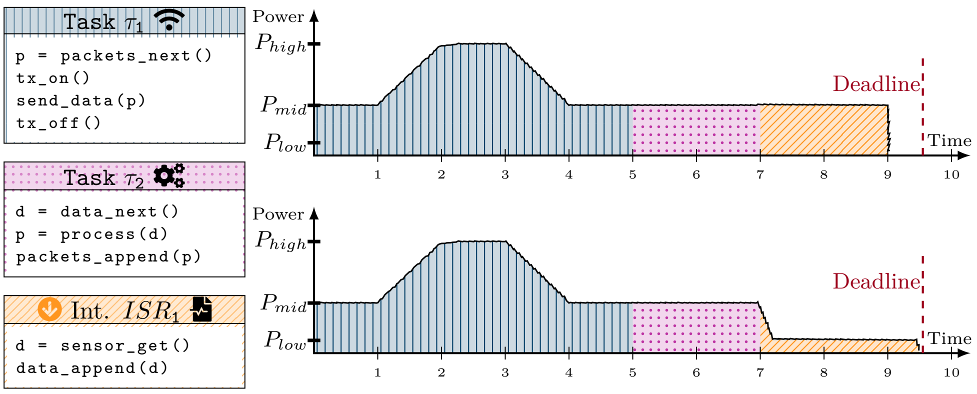
3 Problem Statement

This section details the three problems that Crêpe tackles for controlling preemptions in clock-reconfiguration–aware real-time systems. Crêpe's goal is to find sequences of clock reconfigurations with the objective of minimizing the system's energy demand while reacting to incoming interrupts in time. For this goal, the specific problems are power-state–unaware preemptions (see Section 3.1), clock-agnostic preemption control (see Section 3.2), and complex online scheduling (see Section 3.3), detailed in the following.

3.1 Problem # 1: Power-State–Unaware Preemptions

To illustrate the problem statement, we employ a running example, as shown in Figure 2. The task τ_1 uses a power-intensive transmitter device to send out packets. Activating the device requires a penalty of one time unit (from $t = 1$ to $t = 2$) and, subsequently, makes the system execute with a high-power state P_{high}. After one further time unit, the device is deactivated. In the presence of an interrupt ISR_1, two corner cases are relevant in terms of demanded energy: When the interrupt occurs during the high-power state (case 1), the interrupt duration causes the system to execute longer with a higher power usage. From a temporal point of view, the response times for both the task's job and the interrupt are identical. However, from an energy perspective, executing the interrupt in the low-power mode is superior (case 2). The main problem is that the interrupt is immediately executed without considering the system's current power state, precisely its clock configuration CC.

[1] Terminology for scheduler & dispatcher: We use the term scheduler for a component that determines an order of jobs/interrupts and settings for reconfigurations. A dispatcher operates during runtime and enforces the scheduling strategy based on distinct states (e.g., clock configurations) or binary properties (e.g., the presence of a pending interrupt).



```
p = packets_next()
tx_on()
send_data(p)
tx_off()
```

```
d = data_next()
p = process(d)
packets_append(p)
```

```
d = sensor_get()
data_append(d)
```

Figure 3 Device- and clock-agnostic approaches lead to subpar energy efficiency.

CRÊPE's Approach to Power States. In a nutshell, CRÊPE's scheduling has an awareness of the system's power demand when serving pending interrupts. That is, the execution of service routines is deferred to states where energy can be minimized while maintaining given real-time constraints.

3.2 Problem # 2: Clock-Agnostic Preemption Control

Execution with Previous Clock Configurations. Subsequently, the running example is extended in Figure 3 by a task τ_2 for processing sensed data, which is directly dispatched after τ_1, and the interrupt is served after τ_2. Thereby, the scenario completes standard sense-compute-actuate processing schemes, which are inherent to numerous embedded applications. In the upper power trace, both τ_2 and ISR_1 run with the power demand (i.e., clock configuration) present at the termination of τ_1 (P_{mid}). In this scenario, especially running ISR_1 without further consideration of the clock configuration leads to the problem of subpar resource use upon the depicted deadline (for all tasks/interrupts): Accessing sensor devices, as in the ISR_1, often involves communication over low-level busses, such as SPI [27], I^2C [30], or I^3C [41]. These busses often involve transactional semantics: Once the device access is initiated, the code has to run preemption-free to completion. Besides the transactional semantics, low-level busses are usually comparably slow compared to the CPU's possible execution speed. These observations when accessing devices make energy-demand optimizations under real-time constraints possible: As illustrated in Figure 3's lower trace, the system's speed is reduced, leading to a prolonged execution time of the ISR_1. Further, a penalty for reconfiguring the lower clock speed delays the interrupt's execution, similar to switching on the transceiver device. However, although these two factors increase the execution time, the system's overall energy demand is reduced with this reconfiguration. In light of the fact that the deadline is met, the reconfigured variant shows a resource-optimal solution here.

Related Clock & Preemption Approaches. The works on *Power Clocks* [15], *ScaleClock* [54], and *FusionClock* [17] underline the need to account for modern embedded systems' clock subsystems. We refer to the model notion of the system's power behavior as a *clock-oriented power model.* However, these works miss the property of arguably all real-time systems that interact with the environment, i.e., the presence of interrupts while giving timing guarantees. Consequently, these approaches can also not handle interrupts and their potential reconfigurations, as shown in Figure 3. Works on real-time scheduling that account for interrupts with preemption-control schemes [67] miss the complexity of modern system-on-chip platforms and the option to configure the time-energy tradeoff, highlighted as follows.

Shortcomings of the DVFS Power Model in View of Modern SoCs. A commonly used model for modeling the power-related behavior for energy-constrained real-time systems [3, 4, 5, 13, 22, 42, 46, 65, 66, 67] uses the processor's frequency f (i.e., processor's speed) as the only parameter $P(f) = P_s + P_d(f)$. The parameter P_s determines the CPU's static power demand due to leakage, while $P_d(f)$ represents the dynamic, frequency-dependent power loss due to switching transistors. We thus refer to this model as the *frequency-oriented model*. However, this model is not able to express the complexity of modern clock subsystems, as detailed in Section 2, with (1) multiple clock sources (with different aspects regarding speed or accuracy), (2) clock-specific frequency scalers and dividers (i.e., discrete clock speeds), (3) clock multiplexers, (4) power gates for devices, (5) end-devices requirements of clock settings (i.e., precedence constraints), and (6) device-state–dependent reconfiguration penalties.

Crêpe**'s Approach to Clock & Reconfiguration Awareness.** In contrast to the frequency-oriented model, the *clock-oriented power model*, as used for real-time scheduling in *Fusion-Clock* [17], uses a generic abstraction of the clock subsystem and, thereby, meets the complexity of modern SoCs. We leverage this abstraction for power demand in the context of Crêpe for preemption control in real-time systems.

3.3 Problem # 3: Complexity of Online Scheduling

Our system model targets embedded SoCs with resource constraints (i.e., time, energy, memory). Regarding memory, as detailed in Section 2, the systems we target support at most one megabyte of RAM. In fact, the system targeted in the evaluation (see Section 7) is equipped only with 400 KiB of SRAM memory. As a consequence, complex online scheduling schemes are subpar for the goal of optimal resource use during runtime. That is, existing preemption-control approaches [67] with (comparably complex) online scheduling contradict our pursued system design (besides the fact that these approaches do not meet the complexity of clock reconfigurability).

Crêpe**'s Approach to Avoiding Online Complexity.** In short, Crêpe comprises two design decisions to avoid complex online scheduling for resource-constrained devices: First, the scheduling strategy (considering possible preemptions) is part of the offline analysis, which includes a mathematical optimization problem. This requires high computing costs during the offline phase but relatively low online costs. Second, we make use of the tradeoff between performance and memory demand: Crêpe stores alternative schedules as *schedule tables* for each possible reconfiguration point in the system's memory and, thereby, tolerates high memory demand for low runtime overheads.

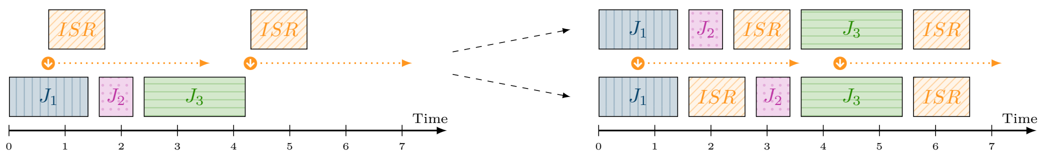


Figure 4 CRÊPE's approach to interrupts at rearrangement points: In this example, for the first interrupt occurrence ISR, two rearrangement points are possible: after J_1 and after J_2. The second occurrence ISR has to be executed after J_3. Therefore, in this case, there are two possible schedules to run the interrupts in time.

4 The CRÊPE Approach

This section outlines the main approach of CRÊPE for controlling preemptions in energy-aware real-time systems under clock reconfigurability, while the next Section 5 is devoted to the description of our mathematical optimization formulation. The outcomes of the optimization are interrupt-aware schedule tables, which are dispatched during CRÊPE's runtime.

Clock Configurations for Tasks. We have a set of tasks, where we have a set of possible CCs for each of them. Subsequently, we assume one interrupt, also with a set of possible CCs as precedence constraints. Multiple interrupt sources can conceptually be served in CRÊPE by one interrupt considered at the minimum inter-arrival time i_i along with server techniques and execution-time budgets accounting for each interrupt's WCET [58]. We further use a 1:1 mapping from tasks to jobs in the following description. The task set is executed periodically with an overall period derived from the periods of the single tasks to determine the overall length, after which the execution of the task set starts again. The minimum inter-arrival time is equal to the deadline D of the interrupt (i.e., implicit deadlines). Between the end of our task set's execution and the next iteration, the system may idle. By allowing our optimization to vary the tasks' CCs, we select more energy-efficient CCs. For each task, we can determine the worst-case execution time of the task τ_i in the clock configuration c ($C_{\tau_i}(c)$) as well as the worst-case energy consumption ($\mathcal{E}_{\tau_i}(c)$).

Reconfiguration Penalties. Between two CCs, we have to reconfigure the system such that the new CC is selected. This introduces reconfiguration penalties, both regarding time and energy consumption of the system. For each reconfiguration from clock configuration c to CC c', the worst-case time penalty $C_{\text{reconf}}(c, c')$ as well as the worst-case energy penalty $\mathcal{E}_{\text{reconf}}(c, c')$ are known. We aim to determine the optimal configuration and reconfiguration dispatching strategy for the task set to minimize the overall energy consumption.

Inclusion of Interrupt Handling. During the task set's execution, an interrupt (a sporadic task) can occur, whose ISR needs to be executed in time. This means that the execution of the ISR has to be finished before a relative deadline D. As an interrupted program also introduces overheads with regard to context switching and caching behavior, we actively avoid the immediate execution or dispatching of the ISR. In contrast, we determine the energy-optimal point between all available tasks for the ISR so that each incoming interrupt is still handled in time. We assume that the tasks are sufficiently small to allow the execution after each task and still meet its relative deadline. If this is not possible with the initial task set, this can be achieved by dividing single tasks into multiple subtasks to meet this

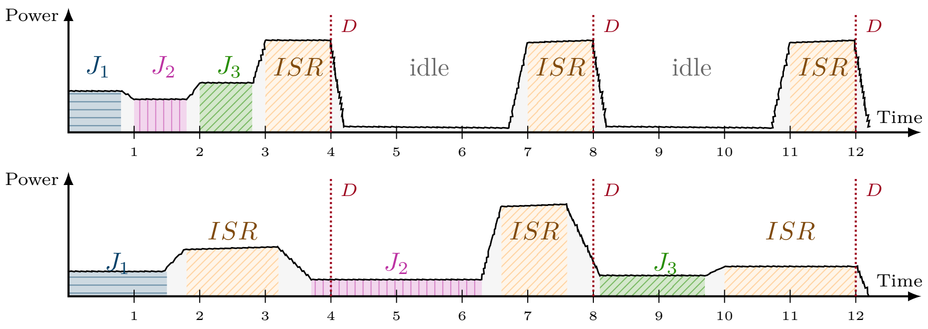


Figure 5 Effect of different CCs on the execution time (x-axis) and power demand (y-axis): In the first schedule, the three jobs J_1, J_2, J_3, and the three executions of the interrupt ISR are executed in fast but power-intensive CCs. Therefore, the execution of all three tasks is finished before the first interrupt is executed. The remaining time between the interrupt execution can be spent in a power-saving idle mode. In the second schedule, the execution uses low-power CCs for the tasks and the different interrupt executions. As a consequence, the system never enters an idle mode but saves energy during the execution itself.

precondition [11]. As our primary goal is to have a sound schedule in every possible case, we assume the maximum amount of interrupts to occur during task execution. To be able to manage these arrivals, we introduce a *rearrangement point, or RAP for short,* after each task. We use this term to emphasize the contrast to *preemption points*, initially introduced by Burns [9], for preemptive scheduling. RAPs are an extension to these (fixed) preemption points as well as an extension to reconfiguration points: At each of such points, a clock reconfiguration can also take place and, consequently, the system can run in an entirely different configuration following a RAP. They are additionally used to check for an interrupt and possibly dispatch it during the following tasks. Figure 4 demonstrates two possible execution plans for an example task set. In summary, at each RAP, two decisions take place in Crêpe: (1) reconfigure the clock or not, (2) execute an interrupt or not. The order of these decisions is relevant since the interrupt can run with a just previously reconfigured CC.

Determining the Optimal Clock Configuration Strategy. To determine the best strategy for an energy-optimal execution, we define a mathematical minimization problem over the total energy consumption of the system for one hyperperiod. It has to consider all parts of the systems with influence on the energy consumption. Therefore, the minimization objective is the sum of the following:

- *Tasks at Different CCs:* Each task has a different energy consumption and time behavior, depending on the selected CC. Therefore, we add a set of binary decision variables for each task, each specifying one of the possible configurations. To exactly choose one CC, the constraint that these variables have to sum up to exactly 1 is added to the optimization problem.
- *Interrupt Occurrences at Different CCs:* Similar to tasks, the ISR has multiple possible CCs, modeled with binary decision variables. But, as the ISRs are not necessarily executed after each task, we constrain the sum of the variables to be ≤ 1. An additional constraint enforcing that all interrupt occurrences are handled in time is detailed later in this section.
- *Costs of Idle Phases:* After the execution of all tasks and the ISRs in between, the remaining time can be spent in idle modes. The costs for these and the interrupt handling during these idle phases will also be detailed later.

- *Reconfiguration Penalties:* Between two CCs, the system is reconfigured to switch from the previous CC to the new CC. These time/energy costs are considered when optimizing the overall energy consumption and, therefore, also added to the optimization objective.

Figure 5 summarizes these costs and shows two possible executions for the same task set. The first schedule uses fast but power-intensive CCs, resulting in fast execution of all tasks and allowing for entering idle modes. The second schedule selects low-power CCs and, therefore, saves energy during the execution of the tasks itself instead of opting for idle modes. Crêpe's optimization problem describes all these possibilities during execution and determines the energy-optimal execution strategy for the underlying task set.

Controlling the Actual Execution of Interrupts. To make sure that interrupts are handled in time, the formalization contains additional constraints on the actual number of interrupts. After each task, there is the possibility that an ISR has to be executed. In order to exactly execute the required amount of ISRs, we count the number of interrupts at each possible execution and compare that to the overall time up until then. Before the execution of the i-th ISR, the lowest possible time since the start of the period to meet the timing requirements is that $i - 1$ ISRs have been executed so far. The assumption that the maximum amount of interrupts enters the system means that the lower bound of the total time before the i-th ISR starts executing is $t = (i - 1) \cdot D$. The highest possible time after the i-th ISR is when the end of the ISR execution exactly meets its deadline. Therefore, the upper bound of the total time directly after the execution of the i-th ISR is $t = i \cdot D$. To model that for the mathematical optimization problem solver, we sum up the binary variables for all ISRs until the i-th one. By using this constraint, we get the number of scheduled ISRs up until this point because if the execution of an ISR is selected, its corresponding CC has to be activated by enabling the binary selection variable. This number is then compared to the current time (with or without the i-th ISR). By adding all these constraints, we can let the mathematical optimization problem solver decide where to enable the binary decision variable. Therefore, it decides where to actually position the i-th execution of the ISR in which CC without violating the timing requirements when receiving the maximum possible number of interrupts.

Modeling the Idle Phases. The remaining time after executing all tasks and ISRs in between can be used for exploiting a low-power idling mode of the SoC. Usually, using an idle or sleep mode saves substantial amounts of energy. We want to utilize this property and add the possibility of reconfiguring our system in such ways during the idle phase. During idling, the system still has to be able to execute each possible occurrence of an interrupt in time. As a consequence, it has to be awake early enough to react to incoming ISR requests in a timely manner. For this, we calculate the longest possible sleep time for each interval by subtracting the WCET of the ISR in its selected CC and the enter-/exit-idle times for the chosen idle mode from the available total time until the interrupt's deadline. With this approach, we can enter, spend time in, and then leave the idle mode and still have enough time left to execute the ISR. Adding new variables (for each CC for all sleep modes and all associated ISR executions in between) and their constraints on the maximum sleep time for each CC at each sleep mode guarantee that our optimization procedure can determine the configurations for the lowest overall energy consumption. We also use these variables to place the constraint on the total execution time: We sum up all times of the reconfigurations, idle modes, and ISRs and add these to the overall time, which has to be equal to the total hyperperiod. After all tasks are finished, the remaining time is spent idling, while the minimal inter-arrival time of the interrupt still defines timeslots for each ISR execution. Therefore, Crêpe uses three different types of idle sequences:

1. The first idle phase is the remaining timeslot until the interrupt's deadline directly after the last task, which can be shorter than the interrupt deadline as the last tasks of our task set extend into that deadline.
2. The last idle phase is the part directly before the next task-set execution starts. We want to clear any outstanding interrupts to continue with the optimized schedule in the next iteration and be able to execute the maximum number of ISRs.
3. All others are intervals with the length of the interrupt deadline, where we check at the end whether an interrupt occurred or not since the last one was handled.

We assign an index to each timeslot defined by the interrupt's minimum inter-arrival time and also use these indices for each possible sleep phase. This means that some sleep phases are completely covered by the task-set execution and, therefore, do not belong to any of the three types above. To model which idle phases are not occupied by the task-set execution, and if not, belong to which idle type, we define two equations: First, we calculate the maximum number of interrupts that can occur during the remaining period with $\mathcal{H} = m_a \cdot D + u_a, 0 < u_a \leq D$ (constant expression, determined before solving). Based on its outcome, we know that at most $m_a + 1$ interrupts can happen, whereby the length of the last slot is u_a time units. Second, we determine how many interrupts have to be executed after the task-set execution with the equation $t(taskset) = m_b \cdot D + u_b, 0 < u_b \leq D$ (determined by the solver). Depending on those formulae, we can determine the type of idling for each sleep index and its length:

1. The last idle phase occurs when $i == m_a$. As m_a is predefined by the period and D of the interrupt, we can predetermine the formula for the last sleep length as well as the index. Therefore, we do not need an extra variable to indicate that. If $m_a > m_b$, then the total time of the idle phase will be u_a. If the last idle phase is also the first idle phase ($m_a == m_b$), then the total time will be $u_a - u_b$.
2. The first idle phase occurs when $i == m_b$ and $i < m_a$. As m_b is only available during solving, we add a decision variable for the first sleep phase. When $m_b == m_a$, there will not be a single variable meeting this requirement, so there will not be any first sleep phase (as this is also the last). In every other case, the time available will be $D - u_b$. For this, we also ensure that we do not schedule an ISR after the last task such that the ISR is executed at the end of the interrupt deadline.
3. The default idle phase occurs when $m_b < i < m_a$. As for the first sleep type, $i < m_a$ is always true for $0 \leq i < m_a$, so we only need to check whether $m_b < i$. When $m_b == m_a$, there will not be a single variable meeting this requirement, so there will not be any intermediate sleep. In every other case, the total maximum time of that idle phase is D.
4. If $i < m_b$, no idle phase can be scheduled since the task set is still in execution. In these cases, the binary variable for the idle phase is unset.

With these constraints, we can minimize the worst-case energy consumption of our system if the maximum number of interrupts occurs. Instead of leaving an idle mode after the predetermined point in time, one could also rely on the chosen target hardware to be woken up by an interrupt. The reconfiguration time when leaving the chosen idle mode must allow the execution of the ISR in time. With this optimization, it is possible to save even more energy if the interrupt occurs less frequently than expected.

Handling at Rearrangement Points. The previous paragraphs detailed the optimization procedure for a task set. As mentioned previously, there is a rearrangement point after each task, where, on the one hand the CC can be changed, but we also check whether an interrupt occurred or not. As the solution of the mathematical optimization problem assumes that the maximum number of interrupts is received, the behavior at these RAPs is as follows: If there is an interrupt to be handled, the system has actually received the maximum number of interrupts possible so far, and we can continue with the currently optimized plan. If no interrupt occurred, there might be a better plan to distribute the tasks over the available time, which additionally enables us to spend more time in the low-power idle modes. Therefore, we also determine the optimal schedule starting at each RAP, assuming the maximum number of interrupts occurring from that point in time. Each situation is defined by the number of tasks already finished, the number of interrupts handled so far, and the remaining time for the task-set execution. If the execution now reaches a RAP with no interrupt to be executed, it then switches to the new, better-optimized strategy before starting the next task.

Comparison to Strictly Periodic Task Sets. The topic of managing the clock subsystem of modern embedded devices is covered by multiple works, the closest related one being *FusionClock* [17]. Similar to CRÊPE, *FusionClock* targets energy-consumption reduction under real-time constraints. However, in contrast to the presented approach of CRÊPE, *FusionClock* assumes a strictly periodic task set, and therefore lacks any control of handling interrupts. CRÊPE accounts for sporadic interrupts, which arise both during the task-set execution and the idle phases, and need to be handled with care to not violate any deadline. CRÊPE not only acquires the necessary time to execute but is also able to select one of the multiple possible execution times and clock configurations for each occurrence. These dispatching strategies are managed by a *table-based scheduling* approach, which is favorable in the context of highly resource-constrained embedded systems [43]. At runtime, CRÊPE considers these tables and decides, depending on the actual presence of an interrupt, whether to reconfigure the system or not.

Exploiting Three Tradeoffs. From a generic point of view, CRÊPE considers three fundamental tradeoffs: (1) In the search space of the *energy-time tradeoff*, CRÊPE's goal is to find worst-case optimal solutions with the outlined strategy. (2) CRÊPE's table-based scheduling exploits the *memory-performance tradeoff* and favors a larger memory footprint for reducing time/energy effort during runtime. (3) Regarding the *offline-online tradeoff*, CRÊPE shifts efforts for making scheduling decisions and clock reconfigurations to the system's offline phase, which will be further detailed as follows in Section 5.

5 Formalization: Offline Optimization

This section formalizes the informative description of our optimization problem in Section 4. The table in Figure 7 introduces the variables used in the formula, whereby we follow the Burns Standard Notation [16] as closely as possible. Each optimization problem at an RAP is described with the following set of variables: The start task s is the task where the current RAP is placed before. The current configuration c_{curr} defines the configuration the system is in at this RAP. The time $\mathcal{H}$ represents the remaining time in the current hyperperiod until the task set has to be finished with execution. We predetermine $\mathcal{H} = m_a \cdot D + u_a, u_a > 0$. We also define two helper functions *tasks* and *idle* (see Figure 6): $tasks([C, E], s, v, w)$ sums up the time or energy consumption of all tasks starting from task s up to task v and possible ISR execution w. $idle([C, E])$ calculates the overall costs for all idle phases, including all reconfigurations and ISR executions. The first parameter to both functions describes whether the WCET (for $idle(C)$) or the WCEC (for $idle(E)$) is returned. For *tasks*, the following parameters describe the first (s) and the last task (v) to consider, and the last ISR during

$$
\begin{aligned}
\mathit{tasks}([C,E],s,v,w) = &\sum_{c\in\mathcal{X}_s} n_{s,c}\cdot[C,E]_{c_{curr}\to c} \;+\; \sum_{t=s}^{v-1}\sum_{c\in\mathcal{X}_t} n_{t,c}[C,E]_t(c)\\
&+\; \sum_{t=s}^{w-1}\sum_{d\in\mathcal{X}_{\hat{j}}} o_{t,d}[\mathbb{C},\mathbb{E}]_t(d) \;+\; \sum_{t=s}^{v-1}\sum_{c\in\mathcal{X}_t}\sum_{d\in\mathcal{X}_{\hat{j}}} n_{t,c}\cdot o_{t,d}\cdot[C,E]_{c\to d} \;+\\
&\sum_{t=s}^{w-1}\sum_{d\in\mathcal{X}_{\hat{j}}}\sum_{c'\in\mathcal{X}_{t+1}} o_{t,d}\cdot o_{(t+1),c'}\cdot[C,E]_{d\to c'} + \sum_{t=s}^{v-1}\sum_{c\in\mathcal{X}_t}\sum_{c'\in\mathcal{X}_{t+1}} (1-o_t)\cdot n_{t,c}\cdot n_{t+1,c'}\cdot[C,E]_{c\to c'}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{idle}([C,E]) = &\sum_{i\in\hat{\mathcal{I}}}\sum_{c\in\mathcal{X}_{N-1}}\sum_{d\in\mathcal{X}_{\hat{i}}} n_{N-1,c}\cdot q_{i,d}\cdot[C,E]_{c\to d} + \sum_{i\in\hat{\mathcal{I}}}\sum_{c\in\mathcal{X}_{\hat{j}}}\sum_{d\in\mathcal{X}_{\hat{i}}} r_{i-1,c}\cdot p_{i,d}\cdot[C,E]_{c\to d}\\
&+ \sum_{i\in\hat{\mathcal{I}}}\sum_{c\in\mathcal{X}_{\hat{i}}} p_{i,c}\cdot ts_{i,c}\cdot[1,P_{i,c}] + \sum_{i\in\hat{\mathcal{I}}}\sum_{c\in\mathcal{X}_{\hat{i}}}\sum_{d\in\mathcal{X}_{\hat{j}}} p_{i,c}\cdot r_{i,d}\cdot[C,E]_{c\to d} + \sum_{i\in\hat{\mathcal{I}}}\sum_{c\in\mathcal{X}_{\hat{j}}} r_{i,c}\cdot[\mathbb{C},\mathbb{E}]_i(c)
\end{aligned}
$$

Figure 6 Formalization of $\mathit{tasks}([C,E],s,v,w)$ and $\mathit{idle}([C,E])$.

task execution (w). The *tasks* function sums up the costs from the current configuration c_{curr} to the start task, the costs for all tasks, all ISRs if selected, and the reconfigurations. The reconfigurations are twofold: if the ISR is active, the costs for reconfiguring from the task before to the ISR and from the ISR to the following task are added; otherwise, the costs from the current to the next task. The *idle* function sums up the reconfiguration from the last task to the first idle phase, the reconfiguration costs between the idle phases and the following ISR and vice versa, the execution of the ISR, and the costs of the idle phase executions.

With these preconditions, we are able to formulate the description from Section 4: We want to minimize the overall energy consumption (minimize $\mathit{tasks}(E,s,N,N-1)+\mathit{idle}(E)$) with the following constraints on the variables:

- The total time has to sum up to the remaining time:

$$\mathcal{H} = \mathit{tasks}(C,s,N,N-1) + \mathit{idle}(C)$$

- For the idle phases, we determine the end of the task-set execution in relation to D:

$$\mathit{tasks}(C,s,N,N-1) = m_b\cdot D + u_b, 0 < u_b \leq D$$

- Per task, exactly one configuration is selected by a binary decision variable:

$$\forall t\in\hat{\mathcal{T}}: \sum_{c\in\mathcal{X}_t} n_{t,c} = 1$$

- After each task, an ISR can be executed ($o_t = 1$) or not ($o_t = 0$):

$$\forall t\in\hat{\mathcal{U}}: \sum_{d\in\mathcal{X}_{\hat{j}}} o_{t,d} = o_t,\;\; o_t\in\{0,1\}$$

- Depending on the task-set execution, an idle phase is active or not:

$$\forall i\in\hat{\mathcal{I}}: p_i = \min(\max(1+i-m_b,0),1)$$

$\mathcal{H}$	remaining time in current period of corresponding task set
N	number of tasks in task set
D	deadline of interrupt
$\hat{\mathcal{T}}$	ordered set of global indices corresponding to tasks in start-time order; $\forall i \in \hat{\mathcal{T}} : 0 \leq i < N$
$\hat{\mathcal{U}}$	ord. set of glob. indices cor. to interrupts in between tasks; $\forall i \in \hat{\mathcal{U}} : 0 \leq i < N - 1$
$\hat{\mathcal{I}}$	ord. set of glob. indices cor. to idle phases; $\forall \hat{i} \in \hat{\mathcal{I}} : 0 \leq \hat{i} \leq \lceil \mathcal{H}/D \rceil$
$\mathcal{X}_i$	set of CCs meeting required precedence constraints for task i respectively also $\mathcal{X}_{\hat{i}}$ for the idle phases and $\mathcal{X}_{\hat{j}}$ for interrupts
$n_{i,c}$	binary decision variable for configuration c of task i
$o_{i,c}$	binary decision variable for configuration c of interrupt after task i
$p_{i,c}$	binary decision variable for configuration c of idle phase i
$q_{i,c}$	binary decision variable for configuration c of idle phase i, indicating first idle phase
$r_{i,c}$	binary decision variable for configuration c of interrupt during idle phase $\hat{i}$
$C_{\hat{i}}(c)$	WCET of task corresponding to task index $\hat{\imath}$ in configuration c
$\mathbb{C}_{\hat{j}}(c)$	WCET of interrupt corresponding to index $\hat{\jmath}$ in configuration c
$C_{c \to c'}$	worst-case time penalty for reconfiguration from c to c'
$E_{\hat{i}}(c)$	WCEC of task corresponding to global task index $\hat{\imath}$ in configuration c
$\mathbb{E}_i(c)$	WCEC of interrupt corresponding to idle index i in configuration c
$E_{c \to c'}$	worst-case energy penalty for reconfiguration from c to c'
$ts_{i,c}$	duration of idle phase i in configuration c
$P_{i,c}$	power consumption for configuration c in idle phase i

Figure 7 Overview of the notation used for CRÊPE's formalization.

- The first idle phase has an additional constraint:

$$\forall i \in \hat{\mathcal{I}} : q_i = \min(\max(1 + m_b - i, 0), 1) \cdot p_i$$

- Only one configuration for each active idle phase and the following ISR:

$$\forall i \in \hat{\mathcal{I}} : \sum_{c \in \mathcal{X}_{\hat{i}}} p_{i,c} = p_i = r_i = \sum_{c \in \mathcal{X}_{\hat{j}}} r_{i,c}$$

- We enforce the required amount of ISR executions:

$$\forall w \in \{x \in \hat{\mathcal{U}} | s \leq x\} : \ tasks(C, s, w, w-1) \geq (\sum_{t=s}^{w-1} o_t - 1) \cdot D$$

$$\forall w \in \{x \in \hat{\mathcal{U}} | s \leq x\} : \ tasks(C, s, w, w) \qquad \leq (\sum_{t=s}^{w-1} o_t) \qquad \cdot D$$

- Idle times of the idle phases are bounded: $\forall i \in \hat{\mathcal{I}}, i < \lceil \mathcal{H}/D \rceil$:

$$ts_i \leq \ p_i \cdot D - q_i \cdot u_b - \sum_{c \in \mathcal{X}_{\hat{j}}} r_{i,c} \cdot \mathbb{C}_i(c) - \sum_{c \in \mathcal{X}_{\hat{j}}} \sum_{d \in \mathcal{X}_{\hat{i}}} r_{i-1,c} \cdot p_{i,d} \cdot C_{c \to d} - \sum_{d \in \mathcal{X}_{\hat{i}}} \sum_{c' \in \mathcal{X}_{\hat{j}}} p_{i,d} \cdot r_{i,c'} \cdot C_{d \to c'}$$

- Special constraint for the last idle phase: $i = \lceil \mathcal{H}/D \rceil, sel = (1 - \min(m_a - m_b, 1))$:

$$ts_i \leq \ u_a - sel \cdot u_b - \sum_{c \in \mathcal{X}_{\hat{j}}} r_{i,c} \cdot \mathbb{C}_i(c) - \sum_{c \in \mathcal{X}_{\hat{j}}} \sum_{d \in \mathcal{X}_{\hat{i}}} r_{i-1,c} \cdot p_{i,d} \cdot C_{c \to d} - \sum_{d \in \mathcal{X}_{\hat{i}}} \sum_{c' \in \mathcal{X}_{\hat{j}}} p_{i,d} \cdot r_{i,c'} \cdot C_{d \to c'}$$

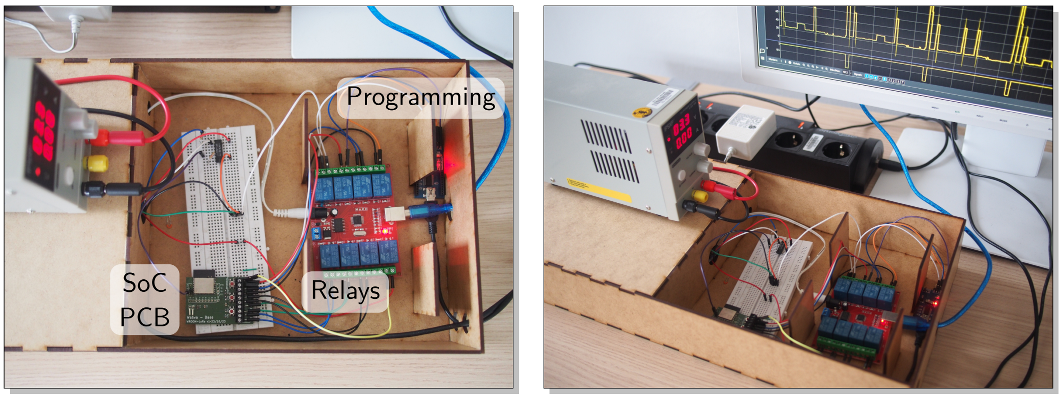


Figure 8 CRÊPE's testbed.

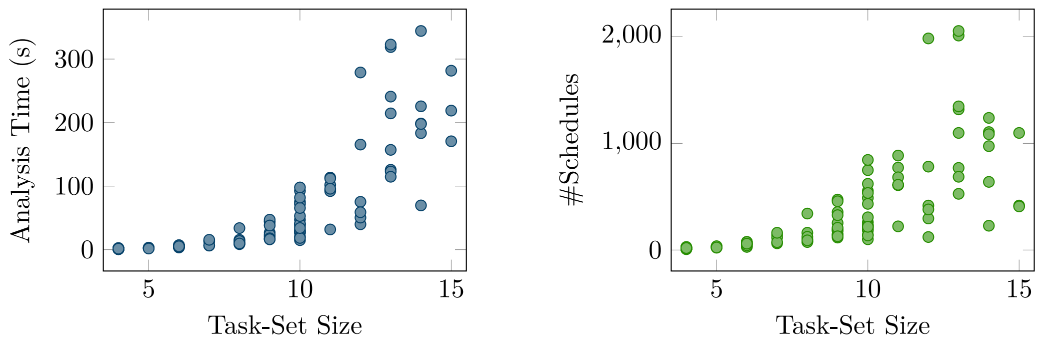
6 Implementation

To show the usability and feasibility of CRÊPE, we implemented our approach for the ESP32-C3 SoC [18, 19]. After introducing the hardware facts, its advantages, and its disadvantages, we explain the workflow to generate an optimized binary from a task-set definition.

Evaluation Hardware. The target platform of CRÊPE is the ESP32-C3 SoC (shown in Figure 8). It is a RISC-V single-core processor with configurable CPU clock frequencies up to 160MHz. One of its core features, making it attractive for CRÊPE, is its large number of available devices, such as WiFi, Bluetooth, SPI, or I2C. Additionally, it offers four different power modes (i.e., active, modem sleep, light sleep, and deep sleep). Therefore, the ESP32-C3 offers many possibilities for managing the system's energy requirements concerning its activated devices and performance. To evaluate the energy consumption and the timing behavior of the device under test as accurately as possible, we designed our own PCB. This circumvents the influence of any interference factors, such as debugging support or other hardware equipped on development boards, such as LEDs, drawing additional power. The ESP32-C3 is equipped with 384 kB internal ROM, 400 kB internal SRAM, and 8 kB RTC memory. The SRAM features single-cycle access to the CPU, making the WCET analysis of cache-related preemption delays obsolete. Therefore, the ESP32-C3 is sufficiently deterministic in its timing and energetic behavior to derive a clock-configuration–aware time- and energy-consumption model (detailed in Section 7.1).

CRÊPE's Workflow. To optimize a task set, CRÊPE needs two sets of information. The first is the information about the possible CCs (timing behavior as clock frequency, energy consumption) and the reconfiguration costs between two CCs. The second is the detailed information about the given task set and the occurring interrupt with its ISR. This corresponds to the possible CCs for each task/the interrupt in combination with the WCET $C_i(c)/\mathbb{C}_i(c)$ as well as the WCEC $E_i(c)/\mathbb{E}_i(c)$ for each CC c. With this information, CRÊPE automatically generates a mathematical optimization problem.

A quadratic problem solver determines the solution for all variables. The quadratic property is necessary since at least the energy costs of the idle phases are described by two variables: (1) the time actually spent in the idle mode, and (2) the decision variable which configuration is selected for the corresponding idle phase. We receive the energy consumption of an idle phase by multiplying these two variables, hence quadratic, with the previously



(a) Total runtime of optimizing regime. **(b)** Number of schedule tables.

Figure 9 Runtime requirements of optimization regime with the Gurobi solver, compared to the number of necessary schedules being optimized during the generation process.

known constants on the energy consumption of the selected mode. Both variables depend on the other variables of the system: the length of the current idle phase depends on the selection of the ISR configuration, and the selected configuration of the idle mode is influenced by the other parameters of the formalization. Therefore, as these variables cannot be transformed into linear constraints, the formalization gets quadratic and requires an appropriate solver. CRÊPE transforms the output of the quadratic solver into optimized code. This code consists of a sequence for each determined schedule per RAP. If a check returns no available interrupt, the schedule needs to be changed. A `goto` statement transfers the program flow to the new schedule. Therefore, CRÊPE features the minimal possible online overhead for scheduling decisions. This code is then compiled into the optimized binary.

7 Evaluation

This section first describes our energy-consumption model used for the evaluation (Section 7.1) and provides details on the developed testbed (Section 7.2). Then, we present our evaluation results in Section 7.3, where we also discuss the feasibility and usability of CRÊPE.

7.1 Energy-Consumption Model

Unfortunately, none of the manuals of the ESP32-C3 goes into detail about the energy consumption of the SoC. Therefore, we based our energy consumption model on the worst-observed power consumption over a set of measurements. To measure the power drawn by the ESP32-C3, we use a Joulescope JS220 precision DC energy analyzer [32]. It can measure the current, voltage, and power demand of the device under test. Additionally, one can start and stop measurements automatically with multiple general-purpose inputs and outputs. Therefore, it is used both to determine an appropriate energy-consumption model and to perform the overall evaluation measurements. For our WCEC model, we overestimate the power consumption of a specific clock configuration by multiplying the maximum observed power consumption by the time spent in that clock configuration: $E(c) = P_{max,c} \cdot C(c)$. The goal for this is to have a bounding energy model despite the absence of detailed documentation in the manuals. This linear approach has been shown to be overly pessimistic for reconfigurations such as entering or leaving sleep modes. Consequently, we determine these penalties by additional analysis of multiple measurements, still using the worst-observed power consumption.

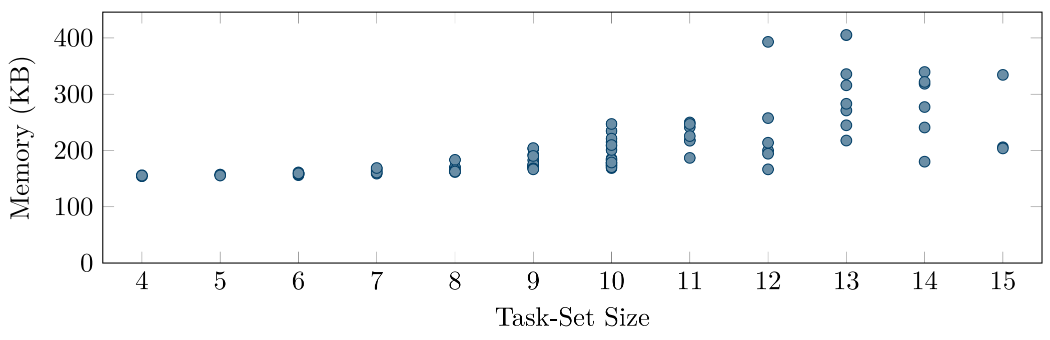


Figure 10 Total memory requirements of optimized binary for different task sets, including the operating system and associated components.

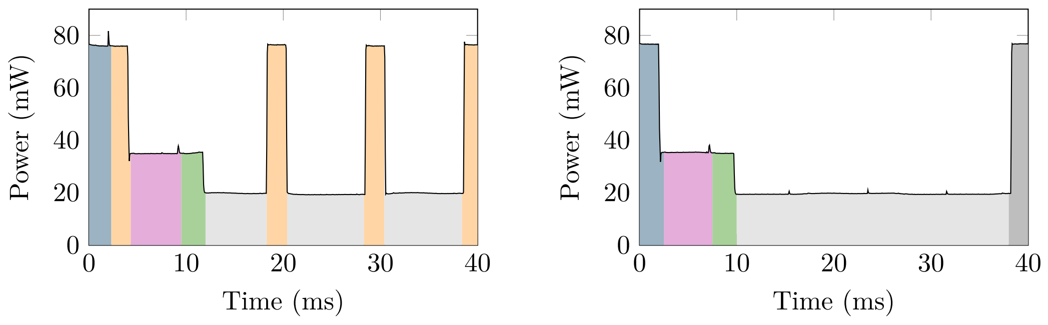
7.2 Testbed

For measuring the energy consumption of a device under test, there are many steps to evaluate a single task set, including several manual interactions, such as (un-)plugging cables and pressing buttons. We extend the setup of the device under test (i.e., ESP32-C3), the energy-measurement device (i.e., Joulescope JS220), and the computer (connected to the SoC with a programming adapter for flashing the binaries) by a USB-controlled relay card as shown in Figure 8. The relay card modifies the test setup such that the computer can build and flash new binaries while the device under test is decoupled from the computer to reduce measurement noise. While this automation is also helpful in avoiding mistakes during the measurements, the main benefit is that this enables us to test a large number of task sets without any human interaction.

7.3 Task Sets

In our evaluation, we generate 115 executable task sets with up to 15 tasks. In comparison, embedded real-time system benchmarks, representative for real-world applications, usually have around a dozen tasks. Specifically, the DEBIE software has 8 tasks [28], PapaBench has 13 tasks [44], and the Rosace case study has 10 tasks [47]. Consequently, we consider our evaluated task-set sizes to be more complex than existing real-world benchmarks for real-time systems, which is essential to assess the limits of Crêpe.

We generate task sets to test several properties of Crêpe with the following goals in mind: (1) evaluate the runtime requirements of the optimization procedure, (2) assess the memory requirements of the schedule tables on the device, and (3) compare our system's actual measured energy consumption to a clock-agnostic system. For this, we use an adapted version of an energy-aware task-set generator [63] based on the UUniFast algorithm [8]. We allow 5 different CCs for all tasks and ISR executions and additionally add two idle modes (light sleep, deep sleep) to the possible CCs for the idle phases. The overall period of the task sets ranges from single-digit millisecond periods up to one second. To simulate different usage patterns of the devices on the SoC, we assume that interaction with devices consists of three phases: First, the device *senses data* from its environment via sensors (e.g., ADCs). Then, the CPU *processes* the collected data before the results are used to *actuate* upon the computation outcome. This is modeled by dividing the tasks into two groups: 60 % are device interaction (sensing & actuating), often with a fixed-frequency timing behavior; 40 % of the tasks model compute-only tasks, heavily depending on the clock frequency of the CPU. Each of the resulting task sets is optimized and analyzed on hardware.



(a) Execution with all interrupts. **(b)** Execution without any interrupt.

Figure 11 Power traces on the target platform of one task set with three tasks and an interrupt with deadline $D = 10\,\text{ms}$ and overall time of 40 ms.

Offline Properties. To solve the optimization problem, Crêpe requires a powerful mathematical problem solver capable of solving quadratic problems. We deploy the `python` programming interface of Gurobi `v11.0.0` [23], which is capable of handling quadratic expressions, on a machine equipped with an AMD Opteron 6180 SE with a total of 48 cores, running at 2.5 GHz, with 64 GB of RAM. Figure 9 shows the analysis time of all task sets for all steps: from reading the configuration data, over optimizing each schedule table, to writing the optimized code. The smaller task sets, starting at a total of 4 tasks, are solved in less than 1 s. The largest task sets take more time but are still finished after at most 350 s, which is still a considerable amount of solving time. Therefore, the tradeoff to move all scheduling decisions into the offline phase to achieve a fast online dispatching scheme is an acceptable solving time for Crêpe to generate optimized schedule tables.

Memory Requirements. The tradeoff of providing a table-based scheduling approach is increased memory usage in favor of low runtime overheads. Our selected hardware platform, the ESP32-C3, has 400 kB of SRAM, which is the maximum possible memory size for a binary. The possibility exists to also store schedule tables in the ROM, but that was not necessary in our case. We use the Espressif IoT Development (`esp-idf`) framework to build the binaries, which extends the FreeRTOS kernel [52]. Therefore, we analyze the binary size of the finished binary in Figure 10. With an increasing task-set size, the memory requirements increase while already being around 150 kB for the smallest task sets. This is due to the size of the underlying FreeRTOS operating system. Nevertheless, the measured binary sizes prove the usability of our approach and support a successful tradeoff between the larger memory footprint and the low time and energy effort during runtime. For SoCs with even smaller memory sizes, reducing the memory overhead, while thereby likely achieving less energy-consumption savings, is possible by decreasing the number of RAPs. This topic is considered future work.

Single Task-Set Evaluation. Before diving into the final evaluation of the energy savings, Figure 11 describes example executions of a single task set. In the first instance, shown in Figure 11a, each possible interrupt arrives at the earliest point in time, whereas its minimum inter-arrival time and deadline is $D = 10\,\text{ms}$. As a consequence, the system has to execute each planned *ISR*. The first *ISR* is executed between the first and second jobs, while the remaining *ISR*s are executed after an idle section. In the second instance depicted in Figure 11b, no interrupt occurs. Therefore, the three jobs are executed directly behind each

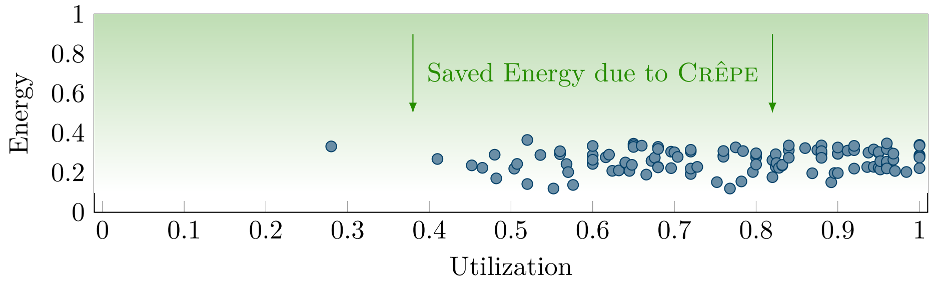


Figure 12 Crêpe's achieved energy savings (70 % on average) over the utilization of task sets.

other before the system enters an idle mode. In the end, the system is awake early enough to handle a possible interrupt, even if none is to be handled, as in this case. These two approaches, in comparison, show that on top of the energy savings achieved by optimizing the clock configurations, the different execution strategies –even if this just means substituting the *ISR* execution time with an idle mode– can save additional energy, depending on the actual execution conditions.

Energy Savings in Comparison to Clock-Agnostic Approaches. The final part of our evaluation compares Crêpe to a clock-agnostic approach. Clock-agnostic describes that the system under investigation uses its default clock configuration, which allows the execution of all tasks and does not reconfigure the system at any time. A static, randomly generated binary sequence defines when an interrupt is available for the execution at a certain time. For the binaries generated by Crêpe, this means that at each possible reconfiguration point, there is a check on the existence of an interrupt. If there is one, the current schedule is kept. Otherwise, the execution strategy is changed to the next schedule table. The clock-agnostic approach executes the interrupt's *ISR* immediately after a task, therefore also preempting the immediate execution, but without reconfiguring the system. Figure 12 shows the results for all task sets. On average, Crêpe reduces energy consumption by 70 % while still giving runtime guarantees. Therefore, this evaluation shows the success of Crêpe in achieving energy savings while still providing worst-case optimal solutions for both energy and timing guarantees.

8 Related Work

Our work on Crêpe has two major areas of related work, which are discussed below. However, to the best of our knowledge, Crêpe is the first approach that tackles preemption control on modern SoCs with their unique reconfigurability of clocks/devices. The first topic of related work touches on energy-aware real-time scheduling along with preemption control. The second paragraph discusses the reconfigurability of modern clocks and the influence of making use of the clock-distribution network of embedded SoCs.

Energy-Aware Scheduling & Preemption Control. Energy-aware scheduling comprises a substantial amount of research, and we refer to the survey of Bambagini et al. [7] for a comprehensive overview. In this context, several works include devices [13, 67, 66], also referred to as *non-DVFS components*. However, these works have shortcomings in view of the complexity of modern clock subsystems: That is, they need to catch up on the configurability of multiple clock sources and end-device requirements with regard to clock configurations in modern SoCs. As a further example, the work of Yang et al. [66] handles penalties for increasing and later decreasing the clock speed as a single penalty. However, our evaluations show that different, context-dependent penalties are involved when switching between arbitrary clock configurations $c_1 \rightarrow c_2$. Crêpe handles such context-dependent configurations and allows arbitrary transitions between clock settings. Crêpe further makes use of controlling by deferring preemptions, a topic part of *limited preemptive scheduling*, where we refer to the survey of Buttazzo et al. [11]. For Crêpe, we make use of fixed preemption points [9] termed as rearrangement points, in order to emphasize their possibility of reconfiguring the entire system's clock state. We envision that future scheduling approaches make use of the *clock-oriented power model*, used by Crêpe, in favor of the frequency-oriented model to meet the requirements of modern SoCs.

Reconfigurability of Modern Clocks. The reconfigurability of clock subsystems has gained attention over the last few years, with the works on *Power Clocks* [15], *ScaleClock* [54], and *FusionClock* [17]. All agree upon the fact that selective clock reconfigurations play a key role in the time-energy tradeoff on embedded SoCs. While having the same goal –reducing energy by reconfiguring the clock subsystem– each approach achieves the goal differently: *ScaleClock* by exploring the clock subsystem during runtime to build its power model, *Power Clocks* by dynamically choosing the most efficient clock, and *FusionClock* by relying on extensive a-priori hard- and software analysis. Crêpe builds upon their understanding of the clock-centric power model and targets the energy-consumption optimization for preemption control.

9 Conclusion

Modern clock subsystems on highly integrated embedded SoC platforms offer a comparably large configuration space for trading-off between timeliness and energy efficiency. On these platforms, dynamically reconfiguring clock speeds and even entire clock sources comes with new challenges and opportunities for energy-aware real-time scheduling beyond the state of the art on DVFS.

With Crêpe, we bring this reconfigurability together with the presence of interrupts and an associated preemption-control scheme. Crêpe's goal is to find worst-case optimal configurations during the system's compile time under consideration of possibly occurring interrupts. At runtime, the actual presence of interrupts is assessed and, based on this information, different clock configurations are dispatched. Crêpe tolerates comparably long analysis times during design time in favor of worst-case optimal and task-aware clock settings. A further tradeoff in Crêpe concerns the memory demand: The offline-determined tables demand a considerable amount of memory. This tradeoff is in favor of low energy/time overhead during runtime, meaning the avoidance of complex, performance-intensive online scheduling. In our evaluation, the analysis durations for representative task-set sizes for our targeted embedded applications show acceptable effort, with always less than six minutes for each task set. For our assessing the energy savings of Crêpe, we developed a testbed for automatically analyzing many task sets: The evaluations show that Crêpe is able to save around 70 % on average in comparison to clock-agnostic approaches. Crêpe's software (i.e., offline optimizer and runtime system), hardware implementation (SoC board and testbed), and artifact evaluation are publicly available under an open-source license:

***Source code of* Crêpe*:* `https://gitos.rrze.fau.de/crepe`**

References

1 S. Ahmed, B. Islam, K. S. Yildirim, M. Zimmerling, P. Pawełczak, M. H. Alizai, B. Lucia, L. Mottola, J. Sorber, and J. Hester. The internet of batteryless things. *Communications of the ACM*, 67(3):64–73, 2024. doi:10.1145/3624718.

2 Sebastian Altmeyer, Robert I Davis, and Claire Maiza. Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, 2012. doi:10.1007/s11241-012-9152-2.

3 J. H. Anderson and S. K.. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS '04)*, pages 428–435, 2004. doi:10.1109/ICDCS.2004.1281609.

4 H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings 13th Euromicro Conference on Real-Time Systems (ECRTS '01)*, pages 225–232, 2001.

5 Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53(05):584–600, 2004. doi:10.1109/TC.2004.1275298.

6 C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: An open toolbox for adaptive WCET analysis. In *Proceedings of the 8th International Workshop on Software Technolgies for Embedded and Ubiquitous Systems (SEUS '10)*, pages 35–46, 2010. doi:10.1007/978-3-642-16256-5_6.

7 Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 15(1):7:1–7:34, 2016. doi:10.1145/2808231.

8 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30:129–154, 2005. doi:10.1007/s11241-005-0507-9.

9 Alan Burns. Preemptive priority-based scheduling: an appropriate engineering approach. In *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, Inc., 1995.

10 J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Proceedings of the 2nd Real-Time Technology and Applications Symposium (RTAS '96)*, pages 204–212, 1996.

11 Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013. doi:10.1109/TII.2012.2188805.

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

13 Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD '07)*, pages 289–294, 2007. doi:10.1109/ICCAD.2007.4397279.

14 Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Determining application-specific peak power and energy requirements for ultra-low-power processors. *ACM Transactions on Computer Systems (TOCS)*, 35(3):9:1–9:33, 2017. doi:10.1145/3148052.

15 Holly Chiang, Hudson Ayers, Daniel Giffin, Amit Levy, and Philip Levis. Power Clocks: Dynamic multi-clock management for embedded systems. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN '21)*, pages 139–150, 2021.

16 Robert I Davis. Burns standard notation for real time scheduling. *Real-Time Systems: The past, the present, and the future. N. Audsley, SK Baruah Editors*, pages 38–41, 2013.

17 Eva Dengler, Phillip Raffeck, Simon Schuster, and Peter Wägemann. FusionClock: Energy-optimal clock-tree reconfigurations for energy-constrained real-time systems. In *Proceedings of the 35th Euromicro Conference on Real-Time Systems (ECRTS '23)*, volume 262, pages 6:1–6:24, 2023. doi:10.4230/LIPIcs.ECRTS.2023.6.

18 Espressif Systems. *ESP32-C3 Technical Reference Manual*, 2022. Pre-release v0.7. URL: `https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf`.

19 Espressif Systems. *ESP32-C3 Series Datasheet Ultra-Low-Power SoC with RISC-V Single-Core CPU*, 2023. Version 1.4. URL: `https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf`.

20 H. Falk and P. Lokuciejewski. A compiler framework for the reduction of worst-case execution times. *Real-time Systems*, 46(2):251–300, 2010. `doi:10.1007/s11241-010-9101-x`.

21 Christian Ferdinand and Reinhold Heckmann. aiT: Worst-case execution time prediction by static program analysis. *Building the Information Society*, 156:377–383, 2004. `doi:10.1007/978-1-4020-8157-6_29`.

22 Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient multi-core scheduling for real-time dag tasks. In *Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS '17)*, volume 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:21, 2017. `doi:10.4230/LIPIcs.ECRTS.2017.22`.

23 Gurobi Optimization, LLC. Gurobi optimizer reference manual. `https://www.gurobi.com/`.

24 Sebastian Hahn, Michael Jacobs, Nils Hölscher, Kuan-Hsun Chen, Jian-Jia Chen, and Jan Reineke. LLVMTA: An llvm-based wcet analysis tool. In *Proceedings of the 20th International Workshop on Worst-Case Execution Time Analysis (WCET '22)*, pages 2:1–2:17, 2022. `doi:10.4230/OASIcs.WCET.2022.2`.

25 Sebastian Hahn, Jan Reineke, and Reinhard Wilhelm. Towards compositionality in execution time analysis: Definition and challenges. *ACM SIGBED Review*, 12(1):28–36, 2015. `doi:10.1145/2752801.2752805`.

26 Damien Hardy, Benjamin Rouxel, and Isabelle Puaut. The Heptane static worst-case execution time estimation tool. In *Proceedings of the 17th International Workshop on Worst-Case Execution Time Analysis (WCET '17)*, pages 8:1–8:12, 2017. `doi:10.4230/OASIcs.WCET.2017.8`.

27 Mark Heene, Susan Hill, and Joseph Jelemensky. Queued serial peripheral interface for use in a data processing system, 1989. patent.

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

29 N. Holsti and S. Saarinen. Status of the Bound-T WCET tool. In *Proceedings of the 2nd International Workshop on Worst-Case Execution Time Analysis (WCET '02)*, pages 36–41, 2002.

30 J.-M. Irazabal and S. Blozis. *AN10216-01 I2C MANUAL*, 2003.

31 Ramkumar Jayaseelan, Tulika Mitra, and Xianfeng Li. Estimating the worst-case energy consumption of embedded software. In *Proceedings of the 12th Real-Time and Embedded Technology and Applications Symposium (RTAS '06)*, pages 81–90, 2006. `doi:10.1109/RTAS.2006.17`.

32 Jetperch LLC. *Joulescope JS220 User's Guide Precision DC Energy Analyzer*, 2022. Revision 1.3. URL: `https://download.joulescope.com/products/JS220/JS220-K000/users_guide/`.

33 Daniel Kästner, Markus Pister, Simon Wegener, and Christian Ferdinand. TimeWeaver: A tool for hybrid worst-case execution time analysis. In *Proceedings of the 19th International Workshop on Worst-Case Execution Time Analysis (WCET '19)*, pages 1:1–1:11, 2019. `doi:10.4230/OASIcs.WCET.2019.1`.

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

35 Raimund Kirner. The wcet analysis tool CalcWcet167. In *Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '12)*, pages 158–172, 2012. `doi:10.1007/978-3-642-34032-1_17`.

36 Chang-Gun Lee, Hoosun Hahn, Yang-Min Seo, Sang Lyul Min, Rhan Ha, Seongsoo Hong, Chang Yun Park, Minsuk Lee, and Chong Sang Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Transactions on Computers*, 47(6):700–713, 1998. doi:10.1109/12.689649.

37 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.

38 Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1):56–67, 2007. doi:10.1016/j.scico.2007.01.014.

39 B. Lisper. SWEET – a tool for WCET flow analysis. In *Proceedings of the 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '14)*, pages 482–485, 2014. doi:10.1007/978-3-662-45231-8_38.

40 Dominique Méry, Bernhard Schätz, and Alan Wassyng. The pacemaker challenge: Developing certifiable medical devices (dagstuhl seminar 14062). *Dagstuhl Reports*, 4(2):17–37, 2014. doi:10.4230/DagRep.4.2.17.

41 MIPI Alliance, Inc. MIPI I3C & MIPI I3C Basic, 2018.

42 S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad. Exploring energy saving for mixed-criticality systems on multi-cores. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS '16)*, pages 1–12, 2016.

43 Mitra Nasri and Björn B. Brandenburg. Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems (outstanding paper). In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS '17)*, pages 75–86, 2017. doi:10.1109/RTAS.2017.34.

44 F. Nemer, H. Cassé, P. Sainrat, J.-P. Bahsoun, and M. De Michiel. PapaBench: A free real-time benchmark. In *Proceedings of the 6th International Workshop on Worst-Case Execution Time Analysis (WCET '06)*, pages 1–6, 2006.

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 Santiago Pagani and Jian-Jia Chen. Energy efficient task partitioning based on the single frequency approximation scheme. In *Proceedings of the 34th Real-Time Systems Symposium (RTSS '13)*, pages 308–318, 2013.

47 Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The rosace case study: From simulink specification to multi/many-core execution. In *Proceedings of the 19th Real-Time and Embedded Technology and Applications Symposium (RTAS '13)*, pages 309–318, 2014. doi:10.1109/RTAS.2014.6926012.

48 James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. Data dependent energy modeling for worst case energy consumption analysis. In *Proceedings of the 20th Workshop on Software and Compilers for Embedded Systems*, pages 51–59, 2017. doi:10.1145/3078659.3078666.

49 Peter Puschner, Raimund Kirner, and Robert G Pettit. Towards composable timing for real-time programs. In *Proceedings of the 1st International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD '09)*, pages 1–5, 2009. doi:10.1109/STFSSD.2009.26.

50 Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, and Gernot Gebhard. The t-crest approach of compiler and wcet-analysis integration. In *Proceedings of the 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–8, 2013. doi:10.1109/ISORC.2013.6913220.

51 Phillip Raffeck, Johannes Maier, and Peter Wägemann. WoCA: Avoiding intermittent execution in embedded systems by worst-case analyses with device states. In *Proceedings of the 25th ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '24)*, 2024. doi:10.1145/3652032.3657569.

52 Real Time Engineers Ltd. The FreeRTOS reference manual: Api functions and configuration options v9.0.0, 2016.

53 John Regehr and Usit Duongsaa. Preventing interrupt overload. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '05)*, pages 50–58, 2005. `doi:10.1145/1065910.1065918`.

54 M. Rottleuthner, T. C. Schmidt, and M. Wahlisch. Dynamic clock reconfiguration for the constrained iot and its application to energy-efficient networking. In *Proc. of the International Conference On Embedded Wireless Systems And Networks (EWSN '22)*, pages 168–179, 2023.

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

56 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 *Proceedings of the 20th International Symposium on Real-Time Distributed Computing (ISORC '17)*, pages 1–10, 2017. `doi:10.1109/ISORC.2017.10`.

57 Philip Sparks. White paper: The economics of a trillion connected devices, 2017.

58 Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1:27–60, 1989. `doi:10.1007/BF02341920`.

59 V. Tiwari and M. T.-C. Lee. Power analysis of a 32-bit embedded microcontroller. *VLSI Design*, 7(3):225–242, 1998. `doi:10.1155/1998/89432`.

60 D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla. Worst-case energy consumption: A new challenge for battery-powered critical devices. *IEEE Transactions on Sustainable Computing*, pages 1–8, 2019. `doi:10.1109/TSUSC.2019.2943142`.

61 Peter Wägemann, Christian Dietrich, Tobias Distler, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS '18)*, volume 106, pages 24:1–24:25, 2018. `doi:10.4230/LIPIcs.ECRTS.2018.24`.

62 Peter Wägemann, Tobias Distler, Timo Hönig, Heiko Janker, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Worst-case energy consumption analysis for energy-constrained embedded systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS '15)*, pages 105–114, 2015. `doi:10.1109/ECRTS.2015.17`.

63 Peter Wägemann, Tobias Distler, Heiko Janker, Phillip Raffeck, Volkmar Sieh, and Wolfgang Schröder-Preikschat. Operating energy-neutral real-time systems. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 17(1):11:1–11:25, 2017. `doi:10.1145/3078631`.

64 Simon Wegener, Kris K. Nikov, Jose Nunez-Yanez, and Kerstin Eder. EnergyAnalyzer: Using static wcet analysis techniques to estimate the energy consumption of embedded applications. In *Proceedings of the 21th International Workshop on Worst-Case Execution Time Analysis (WCET '23)*, pages 9:1–9:14, 2023. `doi:10.4230/OASIcs.WCET.2023.9`.

65 Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. *ACM SIGPLAN Notices*, 40(7):1–10, 2005. `doi:10.1145/1065910.1065912`.

66 C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele. Energy reduction techniques for systems with non-dvs components. In *Proceedings of the IEEE Conference on Emerging Technologies & Factory Automation (ETFA '09)*, pages 1–8, 2009. `doi:10.1109/ETFA.2009.5347153`.

67 Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. Preemption control for energy-efficient task scheduling in systems with a dvs processor and non-dvs devices. In *Proceedings of the 13th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '07)*, pages 293–300, 2007. `doi:10.1109/RTCSA.2007.56`.