FusionClock: Energy-Optimal Clock-Tree Reconfigurations for Energy-Constrained Real-Time Systems

Eva Dengler
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Phillip Raffeck
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Simon Schuster
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

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

Abstract

Numerous embedded real-time systems have, besides their timing requirements, strict energy constraints that must be satisfied. Examples of this class of real-time systems are implantable medical devices, where knowledge of the worst-case execution time (WCET) has the same importance as of the worst-case energy consumption (WCEC) in order to provide runtime guarantees. The core hardware component of modern system-on-chip (SoC) platforms to configure the tradeoff between time and energy is the system's clock tree, which provides the necessary clock source to all connected devices (i.e., memory, sensors, transceivers). Existing energy-aware scheduling approaches have shortcomings with regard to these modern, feature-rich clock trees: First, with their reactive, dynamic (re-)configuration of the clock tree, they are not able to provide static guarantees of the system's resource consumption (i.e., energy and time). Second, they only account for dynamic voltage/frequency scaling of the CPU and thereby miss the reconfiguration of clock sources and clock speed for the other connected devices on such SoCs. Third, they neglect the reconfiguration penalties of frequency scaling and clock/power gating in the presence of the CPU's sleep modes.

In this paper, we present FusionClock, an approach that exploits a fine-grained model of the system's temporal and energetic behavior. By means of our developed clock-tree model, FusionClock processes time-triggered schedules and finally generates optimized code for a system where offline-determined and online-applied reconfigurations lead to the worst-case–optimal energy demand while still meeting given timing-related deadlines. For statically determining these energy-optimal reconfigurations on task level, FusionClock builds a mathematical optimization problem based on the tasks' specifications and the system's resource-consumption model. Specific components like transceivers of SoCs usually have strict requirements regarding the used clock source (e.g., phase-locked loop, RC network, oscillator). FusionClock accounts for these clock-tree requirements with its ability to exploit application-specific knowledge within an optimization problem. With our resource-consumption model for a modern SoC platform and our open-source prototype of FusionClock, we are able to achieve significant energy savings while still providing guarantees for timeliness, as our evaluations on a real hardware platform (i.e., ESP32-C3) show.

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

Keywords and phrases energy-aware scheduling, device-aware whole-system analysis, clock tree

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2023.6

Supplementary Material *Software (ECRTS 2023 Artifact Evaluation approved artifact)*: `https://doi.org/10.4230/DARTS.9.1.2`
Software (Source Code): `https://gitlab.cs.fau.de/fusionclock`

Funding by the Deutsche Forschungsgemeinschaft (DFG) – 502947440 (WA 5186/1-1, Watwa).

Acknowledgements We thank Tim Rheinfels for sharing his expertise with the energy measurements.

1 Introduction

Providing Static Time & Energy Guarantees. Developing systems that meet resource requirements (i.e., time and energy in this paper) with provably optimal resource usage is a central challenge in computer science [9, 32]. While the real-time–systems community has developed numerous analysis approaches to guarantee timing requirements with energy awareness in embedded single-core systems [3, 53], the combined handling of both time and energy constraints still goes beyond the current state of the art in view of modern system-on-chip (SoC) hardware platforms. Specific application scenarios that need to meet both timing and energy requirements include implantable medical devices, such as artificial cardiac pacemakers or defibrillators. Guaranteeing timeliness demands the worst-case execution time (WCET) [51] in order to build a schedule that eventually meets the given tasks' deadlines. Likewise, the tasks' worst-case energy consumption (WCEC) [22, 47, 48, 49] is a fundamental measure for enabling a guaranteed execution of jobs under given limited energy budgets. Having an accurate model of the target hardware platform's temporal and energetic behavior is essential in order to give static resource-consumption guarantees.

Configuring the Time & Energy Tradeoff with Clock Trees. Modern integrated SoC platforms [13] offer a huge variety of features and options to configure the tradeoff between performance (i.e., execution speed) and energy demand, with the heart of this configuration space being the system's *clock tree* [8, 40]. The purpose of the clock tree is to distribute available clock signals to all components on the SoC by utilizing different signal-forwarding mechanisms such as *multiplexers*, *scalers*, or *clock gates*. Figure 1 shows an example of such a clock tree, which will be detailed later. Since components provided with a clock source via an active signal through the clock-distribution network eventually lead to an increase in the whole system's energy consumption (i.e., power over time), these *clock gates* are also referred to as *power gates*. Besides gating power (i.e., on/off switching), the clock tree includes the possibility to scale frequencies up/down with multipliers by using *scalers* or select one out of multiple input signals with the use of *multiplexers*. In summary, modern clock trees of SoCs provide substantial configuration spaces for the tradeoff between time and energy, which has not yet been sufficiently addressed in the context of energy-constrained real-time systems.

Energy Demand of Devices. The clock tree not only spans the configuration options for the CPU: Modern SoC platforms are characterized by their multitude of integrated components, such as transceivers (e.g., WiFi, Bluetooth, LoRa), sensors (e.g., analog-to-digital converter, ADC), controllers (e.g., USB, SPI, DMA), or storage devices (e.g., non-volatile memory). From a generic point of view, all these components have the same behavior as the CPU with regard to clock gating: Switching off devices by clock gating them (when their service is not needed) significantly reduces the system's power and, therefore, is beneficial for energy savings. While numerous energy-aware real-time scheduling approaches account for the systems' energy demand [3, 53], they have shortcomings with (1) comprehensively modeling the resource demand of devices and (2) handling their hardware-related constraints with respect to multiple available clock sources: For example, the WiFi device on a SoC [13] typically requires a specific clock-tree configuration to operate. During operation, its power demand is up to 1105 mW, being significantly larger than the CPU's demand in run mode (i.e., at 1 MHz: 20 mW, at 160 MHz: 100 mW). In summary, to address optimal energy-consumption reduction in real-time systems, knowledge of the whole system's resource-consumption behavior, with all connected consumers, is inevitable.

Low-Power Phases & Sleep Modes. As with the mentioned devices, the CPU is a device itself and does not necessarily have a utilization of 100 % in embedded, energy-constrained settings. Lower utilizations, and thereby slack time, offer the possibility to enter sleep modes that decrease the system's power demand down to, for example, 0.15 mW for the previously mentioned SoC [13]. From a technical perspective, entering a sleep mode means a reconfiguration of the clock tree. An essential accompanying, unavoidable effect of clock-tree reconfigurations is the *penalty* for the reconfiguration. That is, both clock gating and clock scaling involve significant time and accompanying energy overheads, which, for example, come from the duration to stabilize the frequency of a phase-locked loop (PLL) clock source. To give an example, reconfiguring the clock to wake up from a deep sleep mode, enter run mode, and being able to execute user-provided code takes 70.26 ms on the example SoC [13], which needs to be accounted for (1) resource-optimal and (2) deadline-aware execution. With regard to the given real-time constraints, *break-even points* decide whether a specific clock-tree reconfiguration is beneficial: For example, entering and subsequently exiting a sleep mode could have adverse effects on the system's resource consumption. In this paper, we exploit a comprehensive clock-tree model including reconfiguration penalties (i.e., transition costs between clock configurations) to determine worst-case–optimal reconfigurations.

Application-Dependent Reconfiguration. Not all tasks in real-time systems make use of further devices besides the CPU. When considering chains of sense-compute-actuate tasks, the sense phase requires data from sensor devices, and the actuate phase could use transmitters to share results. Neither is the sensor required by the actuation phase nor the transmitter during sensing. For computation tasks, no further devices are needed. Having active devices while executing the compute phase leads to subpar energy demands. From a general perspective, task-agnostic clock-tree reconfigurations inevitably lead to subpar results. Thus, making use of application dependencies is essential for resource-optimal solutions.

Paper's Contributions. This paper introduces FusionClock, an approach that addresses the challenge of meeting deadlines of real-time tasks on single-core SoC platforms while determining an optimum for the energy demand with the use of WCET and WCEC knowledge. FusionClock handles time-triggered schedules and generates code for online reconfiguration of the system's clock tree, tailored for the specific device usage. The name FusionClock originates from our objective of resource-optimally fusing clock-tree reconfigurations with the application's device requirements. FusionClock is able to handle clock trees with multiple input sources and clock/power gates. The paper makes five contributions:

1. *Problem Formalization*: We present a generic quadratic optimization formulation that makes use of a resource-consumption model and is able to adhere to real-time constraints while optimally fulfilling the objective of minimizing worst-case energy demands.
2. *Resource-Consumption Model*: We developed a hardware model for clock-tree reconfigurations on a SoC platform, which is the basis to resource guarantees.
3. *Application-Aware Approach*: We propose an application-aware approach that exploits knowledge of clock requirements and device-aware resource-consumption analyses.
4. *Code Generation*: Based on the quadratic problem's solution for a given taskset and time-triggered schedule, our code-generation approach yields a functionally equivalent but resource-optimal schedule with code for reconfiguring the clock tree between jobs.
5. *Evaluation*: Relying on a hardware platform and employing accurate energy measurements, we demonstrate the effectiveness and validity of our open-source prototype of FusionClock.

2 Background & System Model

FusionClock targets embedded energy-constrained real-time systems executed on single-core SoC platforms. For the tasksets to be optimized, FusionClock assumes a strictly periodic, cyclic task model. As FusionClock optimizes the clock-tree configurations of pre-existing time-triggered schedules, we further assume the availability of a valid, non-preemptive schedule for the taskset. As common in real-time systems, each individual task τ_i can be described by the parameters of its period T_i, its WCET C_i, and its relative deadline D_i. After the hyperperiod H, which is the least common multiple of task periods, the schedule (with its determined clock-tree reconfigurations) repeats. Specific to our approach, we give WCET bounds dependent on the clock-tree configuration, replacing the single C_i with the mapping $C_i(conf)$. Additionally, the task description is extended by the set of device dependencies (e.g., τ_1 uses ADC device 2).

As an ahead-of-time mechanism, FusionClock has requirements on the hardware architecture. We assume static analyzability in the temporal and energetic domain: The hardware's structure allows for the derivation of a static, sound model for the purpose of WCET and WCEC analyses with acceptable analysis pessimism. Besides the feasibility of capturing the microarchitectural behavior, such suitability for resource analysis includes the possibility to describe the energy demand of particular program sections with a known system state (i.e., clock-tree and device configuration) as (monotonic) function of its execution time [48]. We further assume compositionality for energy and time [19, 37]: The validity of safely combining the individual resource demands of continuous sections to a total demand. The limited complexity of the hardware (RISC architecture, simple microarchitecture) found in the targeted system class [13] facilitates modeling for static analyses.

Apart from the processor, SoC systems in the targeted domain also feature numerous devices, such as sensors, actuators, or peripheral communication devices. Due to the nature of SoCs with their integration of various components, all peripherals are generally seen as devices. Those devices significantly influence the system's overall power consumption. We assume to have an accurate bound on the maximum power demand of those devices in all of their different operation modes as well as the transitions between different modes. Accurately determining such application-specific maximum power demands is possible, as shown by Cherupalli et al. [7], which validates FusionClock's related assumption. The same consideration of being able to accurately model time and energy penalties holds for clock-tree reconfigurations, as shown by Park et al. [36].

We assume a feature-rich clock tree that can be reconfigured at runtime by software, allowing fine-grained control over the power consumption of any devices in the system. As for peripheral devices, accurate upper bounds on the time and energy demand of clock-tree–configuration switches are statically determinable. Lastly, every sleep mode of the clock tree has a lower power consumption than all modes used for the execution of tasks. This assumption prevents that unexpected idle times consume more power than the execution of tasks, which is given for our target SoC [13].

The Clock Tree. The *clock tree* is the clock-distribution network within a system, routing the signal from a clock source to all components in the system, potentially modifying the source signal for specific devices. The complexity of this network heavily depends on the chosen hardware platform. Modern SoC platforms feature a variety of clock sources based on different technologies to serve diverse needs and provide a suitable signal source for different application and device demands. The provided clock sources differ from each other with

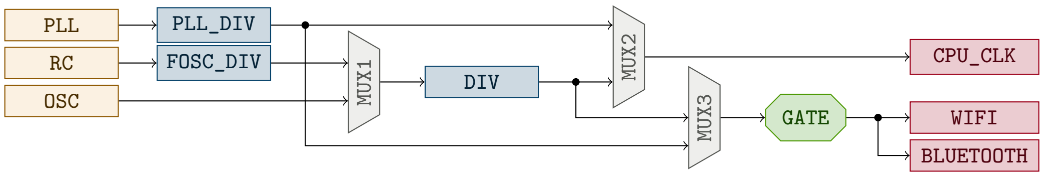


Figure 1 Example of a clock tree for modern SoC platforms [13]. The requirement for the clock-tree configuration are both task- and hardware-related: (1) Tasks can require a specific device (e.g., WiFi) and (2) devices can require a specific clock source with specific multiplexer/divider settings.

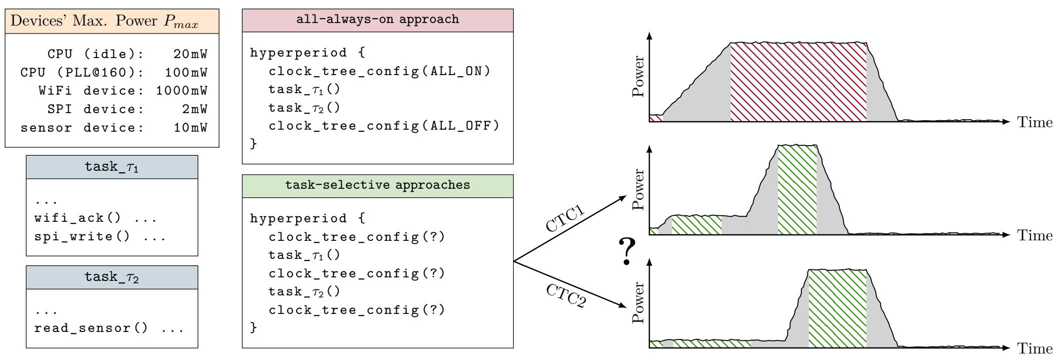
regard to, for example, precision, energy efficiency, signal stability, and robustness against environmental conditions [40]. In general, not every signal source may be suitable for every device or at least not for every operation mode of a device, for example, because a device operation requires a particularly high frequency that cannot be provided by every source.

Figure 1 shows an exemplary clock tree capable of clock-source selection and signal modification. In the example, the tree has three different clock sources (`PLL`, `RC`, `OSC`) and three different devices (`CPU_CLK`, `WIFI`, `BLUETOOTH`), whereas `CPU_CLK` denotes the source clock for the CPU itself. A network of intermediate nodes in the clock tree allows for the modification of the input signal to achieve the requested output signals. This network consists of mainly three different types of nodes: First, there are *clock gates* (`GATE`), the simplest form of modifying an input signal: It can either let the signal pass through the gate to the output or block it off, which can be used to deactivate devices or even complete peripheral busses (i.e., clock-tree subsystems). The second node type in a clock tree is a *scaler* (`PLL_DIV`, `FOSC_DIV`, `DIV`). It modifies the input signal by multiplying or dividing it with a factor before forwarding it to the remaining network. Finally, *multiplexers* (`MUX1`–`MUX3`) receive multiple input signals and, depending on the configuration, select one of them as the output.

This richness of features and configuration possibilities creates high flexibility, enabling trading off between performance and energy efficiency for all devices. It comes, however, with *penalties* for each clock-tree reconfiguration. A penalty describes the time and energy needed to perform the reconfiguration. On the hardware level, modifications to the clock-tree configuration come with varying penalties ranging from miniscule (i.e., few cycles) to significant overheads (i.e., hundreds of milliseconds) [13, 40]. A simple change of a (pre-)scaler, for example, usually requires only a single write to the corresponding divider register. More complex changes, on the other hand, can even require intermediate changes to a temporary helper clock before switching back to the reconfigured original clock (e.g., when switching between PLL clock sources in a microcontroller [33]). These reconfiguration penalties, with regard to both power consumption and time, heavily influence the system's behavior.

3 Problem Statement

In our target domain of embedded energy-constrained real-time systems, applications consist of a set of tasks, the canonically smallest workload unit. Each of these tasks may have different requirements regarding device usage and, thus, different demands on the clock-tree configuration. The naive approach to satisfy device demands is to unselectively choose one configuration that fits all tasks (*all-always-on* approach), but this comes with a higher consumption than necessary for some tasks, for example, because unneeded peripheral devices are enabled. Figure 2 illustrates this problem for a taskset comprising two tasks (`task_`τ_1, `task_`τ_2), with each task requiring one or more devices in addition to the CPU. The upper variant of the `hyperperiod` function displays the *all-always-on* variant mentioned above and



Devices' Max. Power P_{max}

```
    CPU (idle):    20mW
CPU (PLL@160):    100mW
  WiFi device:   1000mW
   SPI device:      2mW
sensor device:     10mW
```

task_τ_1

```
...
wifi_ack() ...
spi_write() ...
```

task_τ_2

```
...
read_sensor() ...
```

all-always-on approach

```
hyperperiod {
  clock_tree_config(ALL_ON)
  task_τ1()
  task_τ2()
  clock_tree_config(ALL_OFF)
}
```

task-selective approaches

```
hyperperiod {
  clock_tree_config(?)
  task_τ1()
  clock_tree_config(?)
  task_τ2()
  clock_tree_config(?)
}
```

Figure 2 The decision when to apply clock-tree configurations (CTC), for example, to de-/activate devices, significantly influences the system's resource consumption. Different power consumptions and reconfiguration penalties affect both the execution time and the energy demand. The energy demands (i.e., integral over power) of reconfiguration penalties are illustrated as gray areas ▨.

the higher-than-necessary power consumption caused by it. The lower variant is expected to have a lower power consumption, as depicted. The effectiveness of selective clock-tree configurations over the all-always-on approach depends on multiple factors, such as the execution time in one clock-tree configuration and the associated reconfiguration penalties. Thus, break-even points between resource demands determine optimal configurations. In summary, an operating system or runtime environment trying to optimize the handling of these different demands with regard to resource usage faces multiple problems:

Problem # 1: Purely dynamic, feedback-based energy minimization approaches pay for their flexibility with a loss of predictability for the system behavior, which is unacceptable in real-time systems executing under strict time and energy budgets.

Problem # 2: CPU-only, device-independent approaches miss the optimization potential of clock-tree reconfigurations that target arbitrary devices on the SoC.

Problem # 3: Modern clock trees with various features are subject to reconfiguration costs that have adversarial effects on reconfigurations and, thus, have to be selectively considered in a configuration-specific approach.

Problem # 1: Resource-Consumption Guarantees. As FusionClock targets real-time systems, optimizing solely for energy consumption does not suffice and threatens the correct system behavior if deadlines cannot be met. The temporal behavior of devices and the tasks using them depends, among other factors, on the active configuration of the clock tree. In the case of the CPU, for example, a lower frequency allows for a more energy-efficient execution while simultaneously prolonging the task execution. The fact that a task running at a lower frequency jeopardizes not only its own deadline but potentially the deadlines of all following tasks exacerbates the problem. In view of this complexity, only static guarantees enable a safe execution at runtime.

Problem # 2: CPU-only Modeling & Energy-Aware Scheduling. The body of related work for energy-aware real-time scheduling is substantial; we refer to the survey of Bambagini et al. [3] and to the overview of device-aware scheduling techniques [52] for further reading. Common to all these works is either the use of CPU-only models or the use of models that do not cover the complexity of modern clock trees in SoC platforms, especially for configuring devices. The energy consumption of peripheral devices such as those used for communication (e.g., WiFi or Bluetooth) often heavily outweighs the demand of the CPU. Consequently, to have usable models for real-world scenarios, WCET and WCEC analyses have to consider the whole system, including all devices [48]. Otherwise, they are not able to give safe estimates of the total energy demand.

But even without modeling additional devices, the different configurations of embedded SoC platforms, which are available for online reconfiguration, yield a broad range of energy consumption: For example, the ESP32-C3 [13], a RISC-V microcontroller SoC, uses 100 mW for the highest available CPU frequency of 160 MHz, while only 20 mW are consumed at a CPU frequency of 1 MHz. For the available sleep modes, this demand drops to 1.3 mW for light sleep and 0.15 mW for deep sleep.

Problem # 3: Clock-Tree–Reconfiguration Penalties. To achieve minimal energy consumption, the specific requirements of all tasks in a system concerning device usage have to be considered. In theory, reconfiguring the clock tree enables us to address selective device demands of each task and to operate devices only in the state required by the current task. Knowledge about these device dependencies alone is, however, not enough, as every reconfiguration itself also influences the system behavior. Depending on the concrete parts of the clock tree that need to be changed, the degree of this influence varies. As such, complex changes come with time and energy penalties: Frequently switching clock-tree configurations tailored to the needs of each specific task can even amount to a higher resource consumption than operating the system at the same configuration for all tasks.

Our Approach. In order to deploy task-specific configurations, we *re*configure the clock tree during the system's runtime making substantial use of a-priori knowledge about the tasks: device usage, temporal properties (period, release time, deadline), and bounds on the WCET and WCEC. Modeling the clock tree in a graph, which includes accurate costs for the transitions between configurations, allows us to assess the influence of potential reconfigurations between tasks and consequently decide which reconfigurations are beneficial. By expressing the WCET of tasks dependent on the active clock configuration and considering transition penalties, we are able to determine the slack time in the schedule and use it as optimization potential by shifting task executions and idle phases for a more energy-efficient schedule. Combining all of the above enables us to generate an energy-optimal variant of the system's schedule by inserting reconfigurations and idle phases that optimize energy usage while simultaneously guaranteeing the correct real-time behavior of all tasks.

4 The FusionClock Approach

In the following, we detail FusionClock, our approach to determining application- and device-aware, guaranteed, worst-case–optimal clock-tree–reconfiguration schedules for embedded systems. Fundamentally, FusionClock is based on the realization that within a given time-triggered schedule, the search for a worst-case–optimal clock-tree configuration is equivalent to a *minimum-cost flow problem* within a suitably structured clock-tree–reconfiguration graph that incorporates the required application- and device-dependent knowledge as construction constraints and transition costs. This section is structured as follows: First, we describe the structure of FusionClock's central data structure, the *clock-tree–reconfiguration graph*. FusionClock uses this graph for flow restrictions in a quadratic-programming problem solvable by mathematic optimizers. We then show how an extension of this problem allows

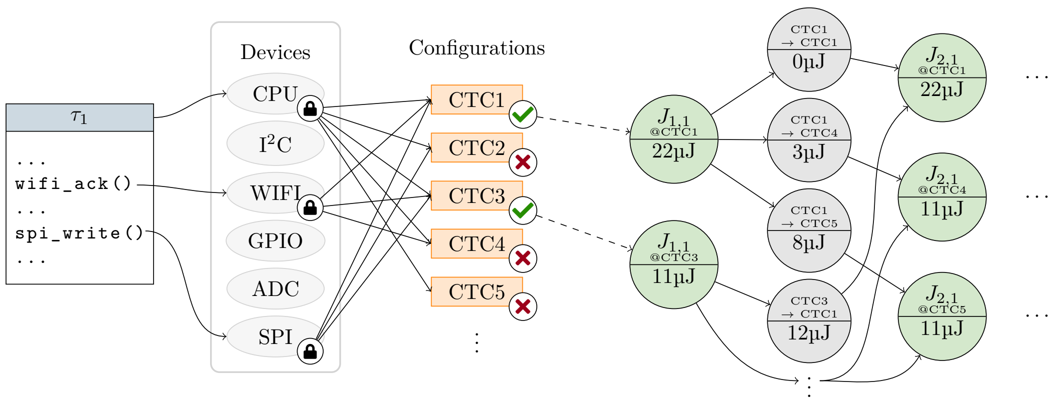


Figure 3 Construction of a *clock-tree–reconfiguration graph* from application- and device-dependent knowledge, here indicated for a job instance $J_{1,1}$ of task τ_1 along with the transitions to the job instance $J_{2,1}$ of the subsequent task τ_2.

the optimizer to redistribute slack within the schedule to reduce a hyperperiod's worst-case energy demand to automatically generate an optimized executable from the solver's output. Finally, we provide a complete formal depiction of the problem.

Clock-Tree–Reconfiguration Graph. The clock-tree–reconfiguration graph provides the basis to determine a schedule's worst-case–optimal sequence of clock-tree configurations (CTC), which is the sequence that will minimize the schedule's WCEC per hyperperiod by selecting optimal reconfiguration points and configurations. At its core, a time-triggered schedule provides a non-preemptive sequence of individual jobs $J_{i,k}$ for the different tasks τ_i within the taskset. In this work, we regard those tasks as the indivisible unit of processing. Here, especially for embedded/cyber-physical systems, it is typical for the individual tasks to interact with various devices within the system, both internal ones such as the CPU device as well as other devices (i.e., sensors, transceivers). However, usage of individual components here requires preparations: As outlined above, tasks can require a specific clock-source configuration to work. At the same time, power gating or frequency scaling different components is vital to minimize energy consumption, an energy-optimal execution of a taskset progresses through a series of different clock-tree configurations. If a task consists of multiple phases with very varied device usage patterns, it thus may be advisable to split those phases into a series of individual subtasks whose CTCs can be optimized individually. Devising a suitable splitting strategy, however, is out of scope of this work. As a first step towards optimization, FusionClock collects the static device-usage information for the individual tasks, which is highlighted with the 🔒 symbol in Figure 3. This is then combined with the SoC-specific knowledge of the individual requirements of a particular device, such as minimum bus frequencies or a specific clock source (e.g., WiFi can require a distinct, highly stable clock [13]), and the corresponding feasible clock-tree configurations. As displayed in Figure 3, these two information sources allow FusionClock to determine the set of feasible clock-tree configurations (✓) for this particular job as the intersection of the different device dependencies in an *application-aware* manner.

Even more, by performing a static WCET analysis and combining the result with the SoC's *resource-consumption model*, we further enrich the graph by attributing upper bounds on the energy consumption of every job's execution phase within the schedule. With this knowledge, it is possible to determine the optimal configuration for every individual phase by comparing the different consumptions of the various viable configurations (see green nodes with @ symbols in Figure 3). However, as each clock-tree reconfiguration comes with transition penalties, those have to be considered when searching for an optimal configuration sequence. For example, performing a rather energy- and runtime-intensive reconfiguration to reach the optimal operation point for a rather short job can have adverse effects on the energy demand. If the former job was executed in a compatible but slightly more energy-intensive configuration, keeping this configuration can turn out cheaper than paying the reconfiguration penalties. To correctly model those penalties, we add an additional set of transition nodes to our flow graph (gray nodes with $\rightarrow$ labels): For every pair of job instances of subsequent jobs (in their respective feasible clock-tree configurations), we add a transition node on the edge to which we assign the particular transition costs obtained from a SoC-specific resource-consumption model for the clock tree. That way, a minimal-flow analysis through the graph is guaranteed to retrieve the optimal, penalty-aware reconfiguration sequence for the application's taskset. The right side of Figure 3 shows the initial section of an exemplary clock-tree–reconfiguration graph, displaying the first two job instances of the tasks τ_1 and τ_2.

Linear Constraints. We first express this minimal-flow analysis as an *integer linear program (ILP)*. Later, we extend the ILP and formulate a *quadratic program (QP)* to account for deadlines in multi-rate systems. In the ILP, we add a binary decision variable (n) for every node in the graph that describes whether the particular node is part of the optimal clock-tree–configuration switching sequence or not. Furthermore, we require that for every set of configuration alternatives of a particular job instance in the graph, at least one has to be taken (i.e., their decision variables have to sum up to one). Additionally, we enforce flow-preservation constraints within the graph: We assign binary decision variables to all transition nodes, and each configuration node's decision variable is equal to those of the sum of the transition nodes on its incoming and outgoing edges. Our constraints formulate a single consecutive path through the flow graph. With respect to the flow constraints, the energy-minimization objective of the sum of all binary decision variables multiplied by the respective energy cost of its particular job or transition yields an energy-optimal assignment to the decision variables, sufficient to reconstruct the optimal schedule. A full formalization, along with the QP extension described subsequently, is listed at the end of this section.

Quadratic Constraints for Slack Redistribution. This ILP is not yet sufficient as it neglects important aspects of temporal constraints. Not every energy-optimal configuration sequence guarantees deadline adherence. Also, in the case of multi-rate systems, the ILP formulation does not yet guarantee the correct handling of release times. Therefore, we refine the formulation of FusionClock to incorporate those constraints for the final *QP formulation*.

As indicated, FusionClock iterates upon a pre-existing time-triggered schedule. When considering an exemplary schedule, such as the one displayed in Figure 4, two observations are essential: (1) In addition to the actual job instances and their compute time, the schedule further contains slack time (ts_i). However, as the execution of the schedule is periodic and repeats after the hyperperiod H, the energy consumption of the complete hyperperiod has to be considered and optimized for. (2) As long as neither releases nor deadlines are violated, FusionClock can redistribute slack across this schedule by "compressing" and "stretching" appropriate parts of the schedule within those limits, as illustrated by the springs in Figure 4. Considering that sleep modes represent one of the most energy-efficient clock-tree states but – especially in the case of deep sleep modes – come with high reconfiguration/wake-up penalties, exploiting this slack redistribution is crucial. Therefore, we model and expose idle

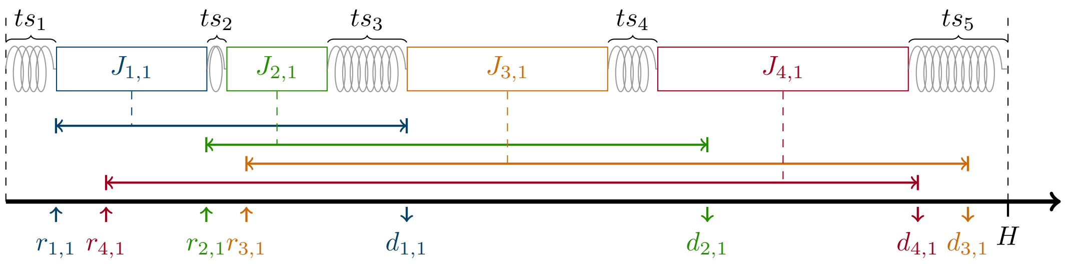


Figure 4 Time-triggered schedules with utilizations below 100 % contain slack time (ts_i). As long as no release or deadline requirements of the underlying tasks are violated, this slack can be redistributed. This image displays a simplified view, as the ILP does not pack fixed-duration jobs but the optimizer is allowed to select different clock-tree configurations for each job instance – each of which can shrink or expand the instance's variable (i.e., clock-frequency–dependent) WCET.

phases as first-class citizens in our optimization formulation: For every idle phase, we add an additional set of alternatives (i.e., different sleep modes and idling variants), along with their reconfiguration/wake-up penalties, to the clock-tree–reconfiguration graph. What sets these phases apart, however, is that they are of variable length: Their combined duration $\sum ts_i$, along with the selected job-instance variants' WCETs $\sum C_{i,j}(conf)$, has to sum up to the system's hyperperiod. That way, the solver is allowed to redistribute the slack for energy minimization. However, this extension comes with the cost of creating a QP based on the initial ILP: When multiplying the variable-length idle times by their selection variables to choose an appropriate sleep mode, we form a multiplication and, thereby, a quadratic optimization problem.

Additionally, we enforce that the scheduled work preceding any job instance's dispatch time (i.e., the time when it is scheduled to start executing) sums up to or surpasses the job instances' release time – this ensures that the optimizer includes sufficient idle time (e.g., $ts_1 + C_{\text{reconf}}(c_{s1}, c_{1,1}) + C_{1,1}(c_{1,1}) + C_{\text{reconf}}(c_{1,1}, c_{s2}) + ts_2 + C_{\text{reconf}}(c_{s2}, c_{2,1}) \geq r_{2,1}$). At the same time, we enforce that when further adding the selected configuration's WCET to that timespan, we still finish before the job instance's deadline: For example, $ts_1 + C_{\text{reconf}}(c_{s1}, c_{1,1}) + C_{1,1}(c_{1,1}) + C_{\text{reconf}}(c_{1,1}, c_{s2}) + ts_2 + C_{\text{reconf}}(c_{s2}, c_{2,1}) + C_{2,1}(c_{2,1}) \leq d_{2,1}$. Thereby, FusionClock guarantees timeliness of the optimized schedule. For providing guarantees, this formulation operates on worst-case values. In practice, job instances may not exercise their full WCET and WCEC. This is, however, not a problem, as idling in the same CTC or entering sleep modes earlier and thus sleeping longer only reduces the system's online energy consumption.

By solving the min-cost flow problem, FusionClock is thus able to determine the global, worst-case–optimal clock-tree configuration with optimized dispatch timings. FusionClock's code generation then extracts this information, in particular the adjusted dispatch timings as well as the CTC-selection variables, from the QP's solution to generate a minimal, tailored implementation of the optimal schedule for the taskset that includes the previously determined clock-tree reconfigurations.

Formalization. Consider a schedule as an alternating sequence of idle phases and job executions, both marked by common but unique indices. For the sake of readability, we omit the mapping of these global job-execution indices to the corresponding task and task-specific job (i.e., from $C_{\hat{\jmath}}$ to $C_{i,j}$), as this mapping is always reconstructable from the available knowledge when constructing concrete formalizations. With this notion and the variables described in Table 1, we are able to formulate the description given above:

$$\min \left(\begin{array}{c} \sum_{\hat{\jmath} \in \hat{\mathcal{J}}} \sum_{c=0}^{f_{\hat{\jmath}}-1} \underbrace{n_{\hat{\jmath},c}\ \mathcal{E}_{\hat{\jmath}}(c)}_{\text{energy costs of jobs}} + \sum_{i \in I} \sum_{c=0}^{f_i-1} \underbrace{n_{i,c}\ ts_{i,c}\ P_{i,c}}_{\text{energy costs of idle phases}} \\ + \sum_{i=0}^{N-1} \sum_{c=0}^{f_i-1} \sum_{c'=0}^{f_{(i+1)}-1} \underbrace{n_{(i,c)\to(i+1,c')}\ \mathcal{E}_{reconf}(c,c')}_{\text{energy penalty for reconfiguration}} \end{array} \right) \text{ wrt.}$$

Linear constraints:

exactly one active configuration per job:

$$\forall i \in \{0,\dots,N-1\} : \quad \sum_{c=0}^{f_i-1} n_{i,c} = 1$$

flow-preservation constraint for incoming edges:

$$\forall i \in \{0,\dots,N-1\} : \forall c \in \{0,\dots,f_i-1\} : \quad \sum_{c'=0}^{f_{i-1}-1} n_{(i-1,c')\to(i,c)} = n_{i,c}$$

flow-preservation constraint for outgoing edges:

$$\forall i \in \{0,\dots,N-1\} : \forall c \in \{0,\dots,f_i-1\} : \quad \sum_{c'=0}^{f_{i+1}-1} n_{(i,c)\to(i+1,c')} = n_{i,c}$$

Quadratic constraints:

idle-phase durations and configuration-specific job WCETs sum up to the hyperperiod:

$$\underbrace{\sum_{i \in I} \sum_{c=0}^{f_i-1} n_{i,c}\ ts_{i,c}}_{\text{idle durations}} + \underbrace{\sum_{\hat{\jmath} \in \hat{\mathcal{J}}} \sum_{c=0}^{f_{\hat{\jmath}}-1} n_{\hat{\jmath},c}\ C_{\hat{\jmath}}(c)}_{\text{execution times}}$$
$$+ \underbrace{\sum_{i=0}^{N-1} \sum_{c=0}^{f_i-1} \sum_{c'=1}^{f_{(i+1)}-1} n_{(i,c)\to(i+1,c')}\ C_{reconf}(c,c')}_{\text{time penalty for clock-tree reconfiguration}} = H$$

preceeding work and idle time sums up to or surpasses release time:

$$\forall \hat{\jmath} \in \hat{\mathcal{J}} : \underbrace{\sum_{i \in I, i<\hat{\jmath}} \sum_{c=0}^{f_i-1} n_{i,c}\ ts_{i,c}}_{\text{idle durations}} + \underbrace{\sum_{\hat{\jmath}' \in \hat{\mathcal{J}}, \hat{\jmath}'<\hat{\jmath}} \sum_{c=0}^{f_{\hat{\jmath}'}-1} n_{\hat{\jmath}',c}\ C_{\hat{\jmath}'}(c)}_{\text{execution times (note the <)}}$$
$$+ \underbrace{\sum_{i=0}^{\hat{\jmath}-1} \sum_{c=0}^{f_i-1} \sum_{c'=1}^{f_{(i+1)}-1} n_{(i,c)\to(i+1,c')}\ C_{reconf}((c,c')}_{\text{time penalty for clock-tree reconfiguration}} \geq r_{\hat{\jmath}}$$

preceeding work and idle time plus job WCET adheres to deadline:

$$\forall \hat{\jmath} \in \hat{\mathcal{J}} : \underbrace{\sum_{i \in I, i<\hat{\jmath}} \sum_{c=0}^{f_i-1} n_{i,c}\ ts_{i,c}}_{\text{idle durations}} + \underbrace{\sum_{\hat{\jmath}' \in \hat{\mathcal{J}}, \hat{\jmath}'\leq\hat{\jmath}} \sum_{c=0}^{f_{\hat{\jmath}'}-1} n_{\hat{\jmath}',c}\ C_{\hat{\jmath}'}(c)}_{\text{execution times (note the } \leq)}$$
$$+ \underbrace{\sum_{i=0}^{\hat{\jmath}-1} \sum_{c=0}^{f_i-1} \sum_{c'=1}^{f_{(i+1)}-1} n_{(i,c)\to(i+1,c')}\ C_{reconf}(c,c')}_{\text{time penalty for clock-tree reconfiguration}} \leq d_{\hat{\jmath}}$$

Table 1 Overview of the notation used for FusionClock's ILP and QP formalization. Not shown for the sake of readability is the mapping from $\hat{\jmath}$ to the corresponding job j and task i, which should be trivially available when constructing concrete problem formalizations.

variable	meaning
H	hyperperiod
N	number of jobs and idle phases
$\hat{\mathcal{J}}$	ordered set of global indices corresponding to jobs in start-time order; $\forall \hat{\jmath} \in \hat{\mathcal{J}} : \hat{\jmath} < N$
I	ordered set of indices corresponding to idle phases (with variable length); $\forall i \in I : i < N$
f_i	number of possible clock-tree configurations for job/idle phase i
$n_{i,c}$	binary decision variable for configuration c of job/idle phase i
$n_{(i,c)\rightarrow(i',c')}$	binary decision variable for reconfiguration from c to c' between jobs/idle phases i and i'
$C_{\hat{\jmath}}(c)$	WCET of the job corresponding to the global job index $\hat{\jmath}$ in configuration c
$\mathcal{E}_{\hat{\jmath}}(c)$	WCEC of the job corresponding to the global job index $\hat{\jmath}$ in configuration c
$r_{\hat{\jmath}}$	absolute release time of the job corresponding to the global job index $\hat{\jmath}$
$d_{\hat{\jmath}}$	absolute deadline of the job corresponding to the global job index $\hat{\jmath}$
$ts_{i,c}$	duration of idle phase i in configuration c
$P_{i,c}$	power consumption for configuration c in idle phase i
$C_{\text{reconf}}(c, c')$	worst-case time penalty for reconfiguration from c to c'
$\mathcal{E}_{\text{reconf}}(c, c')$	worst-case energy penalty for reconfiguration from c to c'

5 Implementation of FusionClock

To show FusionClock's feasibility and evaluate its performance, we created a prototypical implementation of FusionClock on the ESP32-C3 SoC [13]. Figure 5 shows FusionClock's key components and data structures. After discussing important aspects of the hardware and their implications, this section explains how FusionClock uses this information.

FusionClock's Target Platform. The ESP32-C3 is a RISC-V single-core microprocessor, running up to 160 MHz. It features many devices, such as WiFi, Bluetooth, SPI, UART, and multiple low-power modes, along with frequency-scaling support for the CPU device. The SoC is partitioned into nine power domains, de-/activated in four predefined power modes (i.e., active, modem sleep, light sleep, and deep sleep). This offers for a broad tradeoff between energy consumption and performance, depending on the clock-tree configuration. As such, the ESP32-C3 constitutes a suitable test bed for our clock-tree–reconfiguration approach. For our evaluations, we designed a minimal custom PCB. This allows us to observe energy and timing behavior as accurately as possible while avoiding interference factors such as noisy switching regulators. By using the PCB, the hardware is sufficiently deterministic in its temporal and energetic behavior to derive a reliable, clock-tree–aware resource-consumption model from measurements. We detail this process in Section 6.1. This resource-consumption model is the basis for the *cost-annotated clock-distribution graph*: It captures all timing and energy costs for each clock-tree configuration as well as the reconfiguration penalties. Further, we derive the SoC's *device-dependency graph* from the relevant documentation [10, 12, 13] by manual inspection. It captures the dependence of individual devices on certain properties of the clock-tree configuration (e.g., minimal bus frequencies, power gates). This hardware-related model is the basis for our software-related contributions.

FusionClock's Workflow. FusionClock's input comprises two parts: (1) information about the clock tree, which splits up into the device-dependency graph and the cost-annotated clock-distribution graph, as derived above, and (2) information about the application tasks,

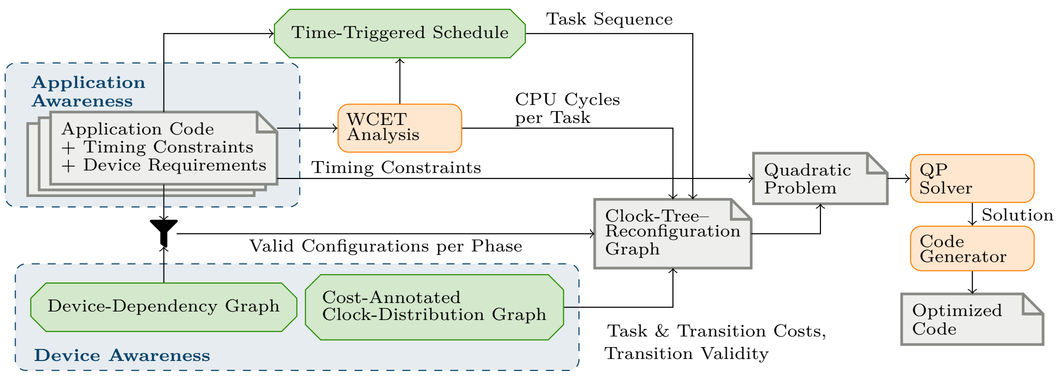


Figure 5 Overview of the FusionClock approach with its central inputs, derived data structures, processing steps, and final output result.

their timing constraints, and device requirements. The information on the tasks' requirements is combined (▼) with the device-dependency graph to determine the valid configurations for each task. As a next step, we use our WCET analyzer (see Section 6.1) to obtain WCETs for all individual tasks. Along with the phase order extracted from a time-triggered schedule, this information then allows to construct the system's clock-tree–reconfiguration graph.

As explained in Section 4, the min-cost flow problem of the clock-tree–reconfiguration graph is translated into a formula understandable by a mathematical solver. After executing the solver, FusionClock uses its output to extract the selected configurations for each task and the time of all variable-length idle tasks. This allows FusionClock to generate code of a specialized system instance by prepending a special *prepare*-hook for each task. It inserts tailored reconfiguration code to transition the SoC to the new clock-tree configuration if the QP's decision variables indicate a reconfiguration. If this reconfiguration did not utilize its full WCET as accounted for in the QP, the task awaits the dispatch time of the subsequent phase to ensure compliance with the time-triggered schedule. Then, the task's workload is evaluated. In a similar fashion, for sleep phases, the system prepares the sleep timer and enters the sleep mode or – in case of active idle – the system idles for the predicted sleep time. This process ensures that the worst-case–optimal system configuration – as determined by the QP's solution – is effectively applied to the target system in a fully automatic manner.

6 Evaluation

This section first describes our evaluation setup (see Section 6.1). Then, we present the evaluation results (see Section 6.2), which consist of three parts: (1) a scalability test for the QP, followed by energy measurements on the SoC for (2) executable tasksets, and (3) a break-even analysis for sleep modes with the help of a benchmark from TACLeBench [14].

6.1 Evaluation Setup

Timing Analysis. Static analyses are conducted using the open-source toolkit Platin [38]. We add a custom RISC-V 32-bit architecture for the ESP32-C3 SoC in addition to the previously supported architectures PATMOS [41] and ARM. As the documentation [11, 12, 13] does not provide the microarchitectural details required to create a static hardware model for this chip, the model is based on measurements of individual instruction-cycle timings utilizing a hardware configuration tailored towards deterministic execution. In this configuration, all

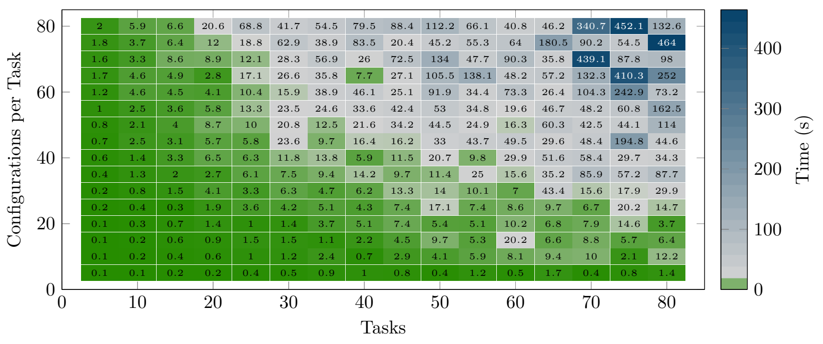


Figure 6 Heatmap showing the solver efficiency for a variable number of tasks on the x-axis and a variable number of possible configurations on the y-axis. With increasing values, the time needed to solve the QP also grows, with a largest value of 464 s, which is still considered acceptable.

application code resides within zero-wait–cycle accessible SRAM, bypassing the need to model flash-access latency and caching behavior. The backing measurements further exercise both pipelining- as well as alignment-related effects that originate from the RV32C (Compressed Instructions) extension [39]. While we cannot guarantee soundness of this model in the current prototype, we did not experience any underestimation in all subsequent evaluations.

Resource-Consumption Model. As the documentation of the energy consumption for the SoC does not provide detailed information about the energy consumption, we used a measurement-based approach to build an expressive energy-consumption model for FusionClock's clock-tree configurations. Here, to derive upper bounds, we base our model on the worst-observed power consumption of the individual clock-tree configurations c weighed by execution time, or expressed formally: $\mathcal{E}_{i,j}(c) = P_{max,c} \cdot C_{i,j}(c)$. Relying on the assumption that P_{max} bounds the maximum configuration-specific power demand and that the WCET bound (C) is safe, this linear approximation determines a valid upper bound of the WCEC. While FusionClock supports expressing the WCEC as a general function ($\mathcal{E}$), this model emphasizes safety over accuracy. To approximate $P_{max,c}$ in a measurement-based manner, we make use of the Joulescope JS220 energy-measurement tool [23], which allows for simultaneously measuring current/voltage, and consequently power demand. To make the model as reliable as possible, the worst-observed energy consumption over multiple runs is taken as a pessimistic reference value for the power consumption of the individual configurations under evaluation. When obtaining WCETs, our model differentiates between two types of tasks: CPU-centric workloads scaling with the underlying clock's frequency, for which we perform the static analyses with Platin as well as fixed-time workloads where a particular device latency determines the (fixed) execution time. For the device-related latencies, we have to rely on worst-observed timings. We found that modeling transition penalties between sleep modes and run modes, with the linear approximation mentioned above ($P_{max} \cdot WCET$), yielded comparably pessimistic results. As a result, we determine these penalties through measurement-based analysis. In all cases, our model provides upper bounds over all observed runs of our evaluation setup.

6.2 Evaluation Results

Solution Efficiency of the Quadratic Program. To solve QPs, a powerful mathematical solver such as Gurobi [17] is required. For evaluation, we deploy Gurobi in version `10.0.0` with a set of synthetic test cases on a machine equipped with two AMD EPYC 7702 64-core processors with a total of 512 GiB of RAM. As test-case sizes, we use a varying number of jobs up to 80 and a varying number of possible clock-tree configurations per job up to 80 possible configurations, allowing a maximum of 6400 different reconfigurations between two subsequent jobs. We consider these sizes comparably large for the context of energy-constrained real-time systems running on modern SoCs. Each job in the test cases has, besides a synthetic WCET, a randomly generated release time and deadline provided to the solver. To allow the presented dynamic redistribution of slack for each job, we bridge every two jobs with an additional idle phase. The resulting evaluation times of Gurobi are presented in Figure 6. As expected, the time needed to solve the test cases grows with the number of configurations and jobs. The maximum time needed to solve the QP within the given evaluation scenario is 464 s, which we consider practical with regard to the fact that static analyses are conducted once during design time. Due to our approach of producing generic QP formulations, we further benefit from continuous improvements (e.g., parallelization) of mathematical solvers.

Break-Even–Point Evaluation. FusionClock strives to provide energy-optimal clock-tree–reconfiguration sequences for the individual tasksets. However, to likewise provide guarantees, FusionClock's underlying resource-consumption model contains some degree of pessimism. This effect is especially visible close to *break-even points*, that is, instances where the optimal CTC for a certain phase changes due to varying task parameters such as its WCET. Subsequently, we study this effect by the example of a variable-sized sleep phase, as the ESP32-C3's different sleep modes (i.e., active idle, light sleep, deep sleep) with their varying reconfiguration penalties and energy consumptions show a representative evaluation scenario. In this scenario, the system executes a single task, the `binarysearch` benchmark of the TACLeBench benchmark suite [14], before awaiting the hyperperiod's next execution. We then vary the length of the hyperperiod H to determine the break-even points between active idling, light sleep, and deep sleep. Figure 7 displays the maximum measurements over 5 runs for all modes, as well as the optimized estimation of the QP (dotted).

For the computation workload on this SoC, the highest CPU frequency is the most energy efficient in terms of instructions per Joule, and, as a consequence, the solver chooses this configuration for the `binarysearch` benchmark for every tested QP formulation. However, the same is not true for the idle phases: Here, minimizing Joule per time is required, and consequently, FusionClock correctly prefers the lowest CPU frequency in this case (active idle □). Still, regarding this metric, the SoC's two sleep modes fare even better, but their high, static reconfiguration penalties still make them unfavorable for short hyperperiods (i.e., $H \leq 8$ ms). Here, FusionClock accurately predicts the break-even point from active idle □ to light sleep △ (left side of Figure 7). For the second break-even point (around 3200 ms, right hand figure) signaling the switch from light sleep △ to deep sleep ○, we observe a deviation between the theoretical prediction based on our resource-consumption model P_{max} (dotted) and the observed, measured break-even point around values of $H = 6000$ ms. We trace this gap back to our pessimistic hardware model of the ESP32-C3 during light sleep: The maximum observed power consumption is 1.31 mW, while the consumption averages around 0.65 mW with a standard deviation of 0.11 mW, the maximum value manifesting in outliers. To illustrate this point, we introduce two additional variations of our power model for light sleep into Figure 7, where we instead base the model on the average value μ with an additional safety margin derived from the standard deviation σ: $P_{3sig} = \mu + 3\sigma$ and $P_{1sig} = \mu + \sigma$. These two models move the solver estimation substantially closer to the observed break-even point. Thus, this observation indicates one way of further improving FusionClock's current

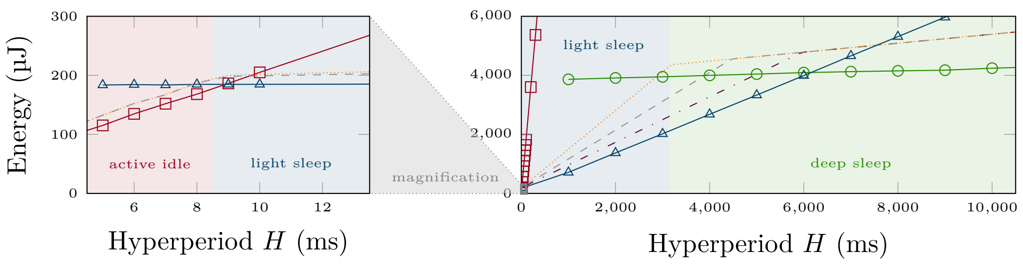


Figure 7 Break-even–point measurements for active idle □, light sleep △, and deep sleep ○ in comparison to QP results with different energy models. P_{max} (dotted) assumes the maximum measured power consumption for each phase for the QP. The models P_{3sig} (dashed) and P_{1sig} (dashdotted) assume a power consumption of $\mu + 3\sigma$ and $\mu + \sigma$ of the measured values for light sleep. In the left part, measurements from 5 ms to 10 ms are in steps of 1 ms, for the right in steps of 1000 ms, with additional measurement points for active idle □ to visualize the higher energy costs compared to both sleep modes. QP results are shown in 1 ms steps (left) and in 100 ms steps (right).

prototype: A more accurate sleep-energy prediction model with less pessimism shifts the reconfiguration sequence towards the observed measurements. In this evaluation, no case of the QP's predicted resource demand shows an underestimation compared to the actual measured demand of the code with its reconfigurations. Thus, the main conclusion for FusionClock from this evaluation is the ability to generate reconfiguration sequences to optimize the energy consumption of the system under observation while accounting for reconfiguration penalties between the respective modes.

Taskset Evaluation. We automatically generate test cases to evaluate whether our approach actually (1) provides a reliable upper bound for the energy consumption and (2) minimizes the energy in comparison to a device-unselective (i.e., all-always-on) application. The generation happens with the following steps: First, we generate tasksets according to an energy-aware generator [50] relying on the UUniFast algorithm [4]. Then, we used a simulation of RMA scheduling [31] at design time to create a time-triggered schedule for these tasksets, splitting tasks where necessary, which eventually creates a sequence of independent tasks. To simulate the tasks' device usage, we assume that device interaction consists of three phases: First *sensing* the environment, then *computing* the resulting action, followed by *actuating* depending on the computation outcome. Therefore, each task is put into one of two groups: 70 % are fixed-time slots, as these devices are assumed to have a non-frequency–scalable timing behavior. The remaining 30 % are considered compute-only tasks where the time depends on the CPU frequency of the SoC. Thereby, we achieve a similar distribution between these three phases, slightly favoring device interactions. As configuration options, we support here 5 clock-tree options in addition to 2 sleep modes, selectively reconfigurable for each task. The hyperperiods range from $H = 25$ ms to $H = 125$ ms over the tasksets consisting of 9 to 18 tasks within a multi-rate system having harmonic periods, which we consider as realistic for our target scenarios. The QP description groups all this information, for which the Gurobi solver then determines an optimized reconfiguration schedule. Next, FusionClock's code generator builds the necessary reconfiguration and idle tasks. For the task's temporal behavior, a loop waits to reach each task's determined WCET. As these evaluations on real hardware require manual interaction with the energy-measurement device over the course of several hours, we conduct this evaluation based on a practically manageable set of ten tasksets, which we randomly selected from the generated tasksets. Figure 8 shows the results of this evaluation by increasing utilizations: The left bars in blue (for the respective

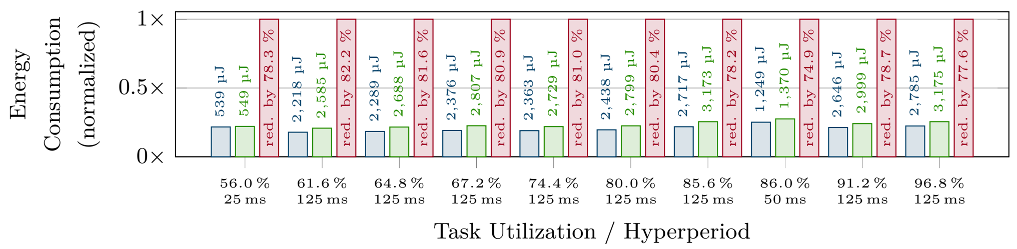


Figure 8 Normalized comparison of an all-always-on approach to the QP solution to the measured energy consumption. The tasksets are ordered by the utilization of the taskset before optimization.

utilization) show the measured, worst-observed energy consumption for each taskset over 50 runs for the utilizations. The middle (green) bars are the solution determined through FusionClock's QP solution, while the right bars show the task-unselective approach to clock configuration, which is also determinable with the QP by restricting the configuration space to the all-always-on option. In all experiments, the QP either selected active idling on the lowest frequency or the light sleep, as the penalty for the deep-sleep mode (i.e., 70.26 ms) is relatively high for these utilizations and hyperperiods. We observe close estimations between the measured and the predicted values: The smallest relative overestimation is in the first shown taskset with 1.8 % (10 µJ in total), while the largest relative difference is in the fourth taskset 15 % (431 µJ in total). The reason for this difference is due to the fact that the tasks consume less than their given budget by the P_{max} energy model. Throughout all evaluated benchmarks, the total measured energy demand is below FusionClock's estimation given by the solver with the help of our resource-consumption model. These values confirm the validity of our modeling approach. Regarding the energy minimization in contrast to the unselective approach, we observed significant improvements: Due to energy savings with the most energy-efficient configuration over time for fixed-time tasks (20 mW to 100 mW) and the energy savings for idle modes, the geometric mean over all observed improvements is 79.4 %. As a main conclusion, we state that FusionClock achieves significant energy improvements, while showing overestimations with acceptable accuracy of the resource model.

7 Related Work

While real-time systems with energy constraints are a well-explored topic, FusionClock goes beyond the current state of the art with its use of static reasoning for guarantees, its penalty-aware handling of multi-source clock trees, its exploitation of application-aware device constraints, and its final code generation. Due to FusionClock's multi-faceted approach to resource-consumption optimization, our work has several scopes of related work: (1) clock-tree (re)configuration, (2) energy-aware real-time scheduling, and (3) static resource-consumption analysis, which are subsequently discussed in this order.

Clock-Tree Reconfiguration. The recent work on *ScaleClock* [40] for the RIOT operating system [1] presents an approach for the online exploration of the clock tree. Instead of building a static clock-tree model, *ScaleClock* employs reconfigurations of the clock tree and thereby learns the model during the system's runtime. In contrast to *ScaleClock*, our FusionClock approach relies on a statically determined clock-tree model with the associated reconfiguration penalties. Having a static model is mandatory in order to give guarantees for timeliness and the execution within energy budgets, which is not possible with *ScaleClock*. As a commonality, we share the same argumentation as *ScaleClock*: Selective clock reconfigurations play a key role in configuring the tradeoff between energy and time in modern SoCs. From FusionClock's perspective, *ScaleClock* is similar to *Power Clocks* [8] with its management of multiple clocks in embedded systems. In contrast to both existing clock-tree reconfiguration techniques, FusionClock has the major difference of giving static runtime guarantees and finding a resource-optimal configuration. Our energy optimizations are possible due to our underlying resource-consumption model of the target hardware.

Industry also tries to satisfy tool-supported configurations of clock trees: The integrated development platform of CubeMX [45] from STMicroelectronics has an option to brute-force clock-tree configurations until an option satisfying all device constraints is found. This clock-tree option is fixed since no reconfigurations happen during runtime prior to dispatching jobs. Therefore, CubeMX makes suboptimal use of resources. With FusionClock, we exploit the notion of the clock tree along with the tasks' WCET and WCEC under respective configurations in order to yield resource-optimal, job-specific configurations.

Energy Awareness in Real-Time Systems. Energy-aware real-time scheduling is a well-explored topic with a substantial body of related work, as surveyed by Bambagini et al. [3]. In this context, works closer related to FusionClock target the scheduling of *non-DVS components*; we refer to the overview on these techniques from Yang et al. [53]. These scheduling techniques partly share our notion of devices, which show an increase in power consumption when active. For example, these techniques cover the topics of device-aware procrastination [6] and preemption control [52]. However, these works have shortcomings in light of modern SoC platforms featuring feature-rich clock trees, which we address with FusionClock. In contrast to prior work, FusionClock has the expressiveness to cope with multiple clock input sources for energy minimization under timing constraints. Further, FusionClock is able to optimally select clock sources based on the fact that devices can have distinct clock-source constraints (e.g., a specific clock source running at its highest frequency). Additionally, due to our generic clock-tree abstraction, we support arbitrary changes and their respective context-sensitive penalties of clock-tree reconfigurations. Finally, the expressiveness of FusionClock's abstraction seamlessly integrates the handling of low-power/sleep modes. Consequently, FusionClock worst-case optimally answers the question of whether to race or pace to idle [26].

Recent work on energy-constrained real-time systems addresses the energy hotspot detection *EHDE* [44] with static analysis techniques. Similar to FusionClock's argumentation, their work emphasizes the need to account for devices and their dynamic range in power demand for energy-aware real-time scheduling. *EHDE* uses a notion of best-case execution time and WCET to move the activation and deactivation in the control-flow graph to yield energy reductions while maintaining a logically and temporally correct system. *EHDE*'s energy-optimization formulation, which includes the best case, helps to approximate the benefits of the code transformation. In contrast to their work, FusionClock has a comprehensive model of the system's clock tree along with the associated transition latencies under worst-case assumptions. That way, FusionClock is able to jointly account for task dependencies on hardware devices while selecting feasible and worst-case–optimal clock sources along with their configuration (i.e., multiplexer, scalers). Thereby, FusionClock generates systems that execute during runtime energy-optimal clock-tree reconfigurations under real-time constraints based on the tasks' WCEC and WCET.

WCET & WCEC Analysis. The resource-consumption analysis for the WCET is well-explored with numerous presented tools [2, 15, 16, 18, 20, 21, 24, 27, 29, 30, 38]. Likewise, several static WCEC-analysis approaches are presented in literature [22, 35, 47, 48, 49].

However, none of these worst-case approaches addresses the static analysis of penalties when reconfiguring the clock tree. The work closest related to FusionClock with regard to WCET analysis with frequency awareness is *FAST* from Seth et al. [42]. *FAST* is able to accurately model caching behavior, which we solve with our zero-wait–cycle accessible SRAM. In contrast to *FAST*, FusionClock has a notion of multiple clock sources and multiple devices driven by means of the SoC's clock tree.

FusionClock's implementation uses the tasks' WCET combined with the maximum power demand of the respective clock-tree configuration. That way, FusionClock is supposed to yield overestimations but leads to pessimism. Numerous related works exist on energy-cost models for the processors [5, 25, 28, 34, 35, 43, 46]. Employing more fine-grained, instruction-level energy models reduces the pessimism with regard to the energy demand of machine code. However, considering the relations of power consumers (i.e., nW to W) in energy-constrained systems, processing cores only take a minor portion of the whole system. Further reducing FusionClock's pessimism in maximum-power determination is possible with more sophisticated approaches as presented by Cherupalli et al. [7].

8 Conclusion

Clock trees are the heart of modern SoC platforms providing the heartbeat to any connected component. We argue that building FusionClock's clock-tree abstraction and employing this abstraction for energy minimization in real-time systems by means of a mathematical optimization problem advances the state of the art: While existing techniques for clock-tree reconfigurations are able to reduce the energy demand, they are not capable of providing both WCEC and WCET guarantees, being now possible with FusionClock. Thereby, FusionClock yields the optimal energy demand with respect to worst-case assumptions. Further, our abstraction integrates the scaling and gating of clocks for all devices, including the CPU itself, in a uniform way. Having such an abstraction is powerful in light of our whole-system resource-consumption optimization. One part of our future work is the reordering of tasks with respect to their precedence constraints, such that, for example, tasks benefitting from the same (or a similar) clock-tree configuration are executed subsequently. Further, the handling of interrupts is considered future work, which will allow us to handle a broader range of real-time applications. Our evaluation on a hardware platform along with measurements emphasizes what the name FusionClock expresses: Fusing the clock-tree configurations between jobs, which, in turn, can use power-consuming devices (e.g., transceivers, memory controllers, sensors), leads to energy-optimal execution sequences, while maintaining deadlines of tasks. The evaluations further outline that significant (i.e., around 80 %) energy savings are possible compared to clock-tree–agnostic approaches. Besides these savings, FusionClock's analysis time to determine worst-case–optimal solutions in large clock-tree–configuration spaces is considerably short (i.e., few minutes) for static-analysis approaches, which is due to our approach of formulating quadratic problems for fast mathematical solvers. As a future perspective, we envision that more approaches use clock-tree abstractions as the basis for resource-consumption optimizations in embedded systems.

Source code of FusionClock: `https://gitlab.cs.fau.de/fusionclock`

References

1 Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. RIOT: an open source operating system for low-end embedded devices in the iot. *IEEE Internet of Things Journal*, 5(6):4428–4440, 2018. doi:10.1109/JIOT.2018.2815038.

2 Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal 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.

3 Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio C. 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.

4 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.

5 Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED '00)*, pages 185–190, 2000. doi:10.1145/344166.344576.

6 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.

7 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 (ACM TOCS)*, 35(3):9:1–9:33, 2017. doi:10.1145/3148052.

8 Holly Chiang, Hudson Ayers, Daniel B. Giffin, Amit Levy, and Philip Alexander Levis. Power clocks: Dynamic multi-clock management for embedded systems. In *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks (EWSN '21)*, pages 139–150, 2021. URL: https://dl.acm.org/doi/10.5555/3451271.3451284.

9 Albert Cohen, Xipeng Shen, Josep Torrellas, James Tuck, Yuanyuan Zhou, Sarita Adve, Ismail Akturk, Saurabh Bagchi, Rajeev Balasubramonian, Rajkishore Barik, Micah Beck, Ras Bodik, Ali Butt, Luis Ceze, Haibo Chen, Yiran Chen, Trishul Chilimbi, Mihai Christodorescu, John Criswell, Chen Ding, Yufei Ding, Sandhya Dwarkadas, Erik Elmroth, Phil Gibbons, Xiaochen Guo, Rajesh Gupta, Gernot Heiser, Hank Hoffman, Jian Huang, Hillery Hunter, John Kim, Sam King, James Larus, Chen Liu, Shan Lu, Brandon Lucia, Saeed Maleki, Somnath Mazumdar, Iulian Neamtiu, Keshav Pingali, Paolo Rech, Michael Scott, Yan Solihin, Dawn Song, Jakub Szefer, Dan Tsafrir, Bhuvan Urgaonkar, Marilyn Wolf, Yuan Xie, Jishen Zhao, Lin Zhong, and Yuhao Zhu. Inter-disciplinary research challenges in computer systems for the 2020s. Technical report, National Science Foundation, USA, 2018.

10 Espressif Systems. *ESP32-C3-DevKitM-1*, 2022. v5.0.1. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/hw-reference/esp32c3/user-guide-devkitm-1.html.

11 Espressif Systems. *ESP32-C3-Mini-1 Datasheet*, 2022. Version 1.3. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c3-mini-1_datasheet_en.pdf.

12 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.

13 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.

14 Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In *Proceedings of the 16th International Workshop on Worst-Case Execution Time Analysis (WCET '16)*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/OASIcs.WCET.2016.2.

15 Heiko Falk and Paul 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.

16 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.

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

18 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASIcs.WCET.2022.2.

19 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.

20 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/OASIcs.WCET.2017.8.

21 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.

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

23 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/.

24 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.WCET.2019.1.

25 Steve Kerrison and Kerstin Eder. Energy modeling of software for a hardware multithreaded embedded microprocessor. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 14(3):56:1–56:25, 2015. doi:10.1145/2700104.

26 David H. K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *Proceedings of the 3rd International Conference on Cyber-Physical Systems, Networks, and Applications (ICCPS '15)*, pages 78–85, 2015. doi:10.1109/CPSNA.2015.23.

27 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. Springer, 2012. doi:10.1007/978-3-642-34032-1_17.

28 Sheayun Lee, Andreas Ermedahl, Sang Lyul Min, and Naehyuck Chang. An accurate instruction-level energy consumption model for embedded RISC processors. *SIGPLAN Notices*, 36(8):1–10, August 2001. doi:10.1145/384198.384201.

29 Xianfeng Li, Liang Yun, 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.

30 Björn 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. Springer, 2014. doi:10.1007/978-3-662-45231-8_38.

31 Chang L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.

32 Dominique Méry, Bernhard Schätz, and Alan Wassyng. The pacemaker challenge: Developing certifiable medical devices (dagstuhl seminar 14062). *Dagstuhl Reports*, 4(2):17–38, 2014. doi:10.4230/DagRep.4.2.17.

33 Microchip. *PIC32 Family Reference Manual, Section 6. Oscillators*, 2011. Revision G. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/61112G.pdf.

34 Spiridon Nikolaidis, Nikolaos Kavvadias, Periklis Neofotistos, K. Kosmatopoulos, Theodore Laopoulos, and Labros Bisdounis. Instrumentation set-up for instruction level power modeling. In *Integrated Circuit Design*, pages 71–80, 2002. doi:10.1007/3-540-45716-X_8.

35 James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. Data dependent energy modeling for worst case energy consumption analysis. In *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES '17)*, pages 51–59, 2017. doi:10.1145/3078659.3078666.

36 Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, Massoud Pedram, and Naehyuck Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Compututer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013. doi:10.1109/TCAD.2012.2235126.

37 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.

38 Peter P. Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, and Gernot Gebhard. The T-CREST approach of compiler and wcet-analysis integration. *16th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2013*, pages 1–8, June 2013. doi:10.1109/ISORC.2013.6913220.

39 RISC-V Foundation. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*, December 2019. Document Version 20191213. URL: https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf.

40 Michel Rottleuthner, Thomas C. Schmidt, and Matthias Wählisch. Dynamic clock reconfiguration for the constrained iot and its application to energy-efficient networking. In *Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks (EWSN '22)*, pages 168–179, 2022. URL: https://dl.acm.org/doi/abs/10.5555/3578948.3578964.

41 Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: a time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, 2018. doi:10.1007/s11241-018-9300-4.

42 Kiran Seth, Aravindh Anantaraman, Frank Mueller, and Eric Rotenberg. FAST: Frequency-aware static timing analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS '03)*, pages 40–51, 2003. doi:10.1109/REAL.2003.1253252.

43 Yakun Sophia Shao and David M. 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. doi:10.1109/ISLPED.2013.6629328.

44 Mohsen Shekarisaz, Lothar Thiele, and Mehdi Kargahi. Automatic energy-hotspot detection and elimination in real-time deeply embedded systems. In *Proceedings of the Real-Time Systems Symposium (RTSS '21)*, pages 97–109, 2021. doi:10.1109/RTSS52674.2021.00020.

45 STMicroelectronics. *User manual STM32CubeMX for STM32 configuration and initialization C code generation*, 2023. Rev. 39. URL: https://www.st.com/resource/en/user_manual/dm00104712.pdf.

46 Vivek Tiwari and Mike Tien-Chien Lee. Power analysis of a 32-bit embedded microcontroller. *VLSI Design*, 7(3):225–242, 1998.

47 David Trilla, Carles Hernández, Jaume Abella, and Francisco J. Cazorla. Worst-case energy consumption: A new challenge for battery-powered critical devices. *IEEE Transactions on Sustainable Computing*, 6(3):522–530, 2021. doi:10.1109/TSUSC.2019.2943142.

48 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. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ECRTS.2018.24.

49 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.

50 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.

51 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 7(3):36:1–36:53, 2008. doi:10.1145/1347375.1347389.

52 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.

53 Chuan-Yue Yang, Jian-Jia Chen, Tei-Wei Kuo, and Lothar Thiele. Energy reduction techniques for systems with non-dvs components. In *Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '09)*, pages 1–8, 2009. doi:10.1109/ETFA.2009.5347153.