Positional Paper: A Clock-Distribution Abstraction for Resource-Constrained Real-Time Systems in Zephyr

Eva Dengler[1], Tobias Häberlein[1], Phillip Raffeck[1] and Peter Wägemann[1]

[1]Friedrich-Alexander-Universität Erlangen-Nürnberg, Martensstr. 1, 91058 Erlangen, Germany

Abstract

Energy-constrained real-time systems are subject to multiple constraints during their execution: On the one side, applications come with tasks that require real-time guarantees. On the other hand, the available energy is constrained through a limited power supply. These constraints require a careful handling of resources and resource savings are especially beneficial. The key components for energy savings, for example, are the clock subsystems on modern embedded platforms. These subsystems are intertwined with all components on System-on-Chip devices and regulate both energy consumption and temporal behavior. As Zephyr is already aware of the clock components via its device tree, we want to bring attention to the availability and usability of a clock-subsystem abstraction and discuss possible goals in the near future.

Keywords

Embedded Systems, Time/Energy Tradeoff, Device-Aware Optimization, Clock Distribution Network, Energy-Aware Real-Time Scheduling

1. Introduction

The demand for portable and energy-efficient computing has grown steadily in recent years. As a result, embedded systems are increasingly being utilized across a wide range of applications. Notable examples include the Internet of Things (IoT) [1], which comprises a rapidly expanding number of connected devices, as well as intermittent systems that operate without a fixed power supply. These systems often face environmental constraints, such as limited time or energy resources. Consequently, the importance of energy-constrained real-time systems, which operate on modern embedded System-on-Chip (SoC) platforms, continues to rise.

1.1. Embedded Real-Time Systems and their Challenges

The rise of IoT and intermittent systems has led to a significant increase in the use of embedded devices. Their applications, ranging from automotive systems to medical devices, often have specific *timing* and *energy* constraints, which are fundamental to many safety-critical domains. Therefore, it is crucial that the developers of these systems take care to comply with those constraints:

Timing Constraints To ensure guarantees regarding their timely execution, real-time tasks must be finished within predefined deadlines. Therefore, it is necessary to know an upper bound for the *worst-case execution time* (WCET) of each task, which denotes the most extended timespan a task needs for its execution on a given hardware platform for any given input and adverse hardware states (e.g., cache misses). Determining accurate WCET bounds is a problem that has been the focus of extensive research [2]. Note that the WCET heavily depends on the underlying hardware platform.

This work is funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under the project number 02947440 (WA 5186/1-1, Watwa: Whole-System Optimality Analysis and Tailoring of Worst-Case–Constrained Applications).

ZiSE 2025: 1. International Conference on Zephyr in Science and Education, Sept 23–24, 2025, Jena, Germany

dengler+zise25@cs.fau.de (E. Dengler); haeberlein+zise25@cs.fau.de (T. Häberlein); raffeck+zise25@cs.fau.de (P. Raffeck); waegemann+zise25@cs.fau.de (P. Wägemann)

https://sys.cs.fau.de/~dengler/ (E. Dengler); https://sys.cs.fau.de/~haeberlein/ (T. Häberlein); https://sys.cs.fau.de/~raffeck/ (P. Raffeck); https://sys.cs.fau.de/~waegemann/ (P. Wägemann)

0009-0001-3444-852X (E. Dengler)

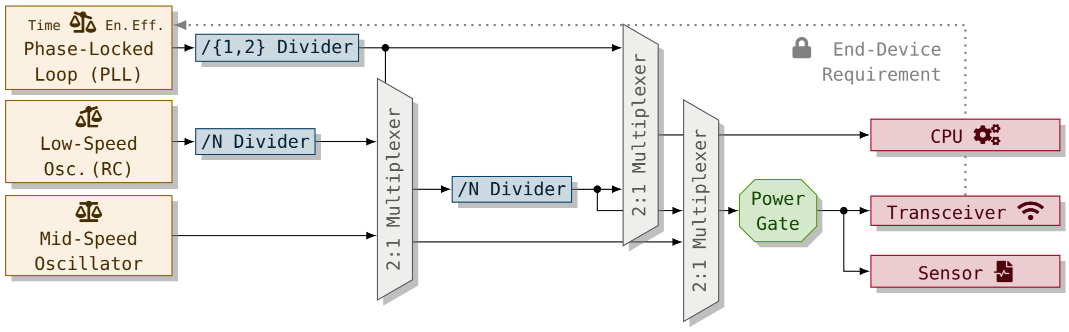

Figure 1: The Clock Distribution Network [8]: By configuring the system clock during runtime, the clock distribution networks of modern embedded SoCs offer numerous possibilities for a tradeoff between temporal performance and energy efficiency.

Energy Constraints Another consideration is energy, as the embedded nature of IoT devices often imposes difficulties concerning their energy storage and, therefore, energy consumption. Many of these systems are either battery-operated or harvest energy from their surroundings [3]. Therefore, they rely on certain guarantees regarding their energy consumption in order to operate for a specified period of time. In this context, the notion of worst-case energy consumption (WCEC) has been introduced as a parallel to WCET, and has become an important subject of ongoing research [4, 5, 6, 7].

1.2. Clock Subsystems

One of the critical challenges in embedded real-time systems is managing the tradeoff between energy consumption and computing time. This balance is primarily controlled through complex clock subsystems known as *clock distribution networks* or *clock trees*, which serve the essential function of delivering clock signals to all components within the system. An example clock distribution network is shown in Fig. 1. It is composed of several types of nodes, each serving a distinct role:

- *Input nodes*: The inputs provide the network with various clock sources, each offering a different set of properties. For real-time systems, the frequency of the clock source is a crucial factor as it impacts the device's timing behavior and power consumption. Additionally, the precision and stability of the clock source may play a significant role for some output devices.
- *Output nodes*: These nodes typically correspond to processing units, sensors, or actuators. Examples are the central processing unit, speed or temperature sensors, and motors or LEDs. Each of these devices may have different requirements regarding its input clock signal. These include the frequency, as well as the accuracy and stability of the input clock signal. For instance, the radio on an ESP32-C3 only works with the phase-locked loop (PLL) clock source [9].
- *Gates*: Gates act as switches within the clock distribution network, controlling whether the clock signal is propagated to downstream components. By selectively turning off parts of the network, gates enable substantial reductions in the system's overall energy consumption. Developers must ensure that they only power-off devices that are not needed during a specific code section.
- *Multiplexers*: Multiplexers act as a selector between multiple different input signals. They route one of the available input signals to the output. This enables a flexible frequency adaption and selection, depending on the requirements of the connected components.
- *Scalers*: Scalers adjust the frequency of the clock signal using configurable multipliers or dividers, allowing for different frequencies at the output nodes.

With these components, the clock distribution network can be configured not only to meet timing objectives but also to improve the system's energy efficiency simultaneously. However, a key factor when interacting with the clock distribution network is the overhead associated with clock-configuration changes. These *reconfiguration penalties* influence both the timing and energy behavior and include both software- and hardware-related costs, e.g., for initializing or reconfiguring hardware. The significance of the reconfiguration costs becomes evident when the transition costs outweigh the potential energy savings. For instance, although using a low-power sleep mode is generally the most energy-efficient option during idle periods, the transition overhead associated with entering and exiting the sleep mode can result in a higher overall energy consumption.

When reconfiguring the system, additional reconfiguration penalties may arise. These penalties can be hardware-related or result from necessary software initialization or device configurations. In both cases, the time and energy needed for these reconfigurations is non-negotiable and may render an otherwise optimal configuration unprofitable in terms of overall energy savings. Therefore, although dynamically adjusting the clock subsystem between tasks can offer significant benefits, the impact of reconfiguration overheads should be considered to ensure that such adjustments truly improve the overall efficiency.

2. Previous Work on Clock Subsystems

As discussed in Section 1.1, the primary component controlling the energy consumption and timing behavior of modern embedded devices is the clock distribution network. There exist multiple approaches to improve different aspects of operation for this topic, which we will discuss in this chapter.

2.1. Detecting Possible Clock-Subsystem Configurations

Before the clock distribution network can be utilized, the possible configuration space must be explored and evaluated. This process involves identifying all possible configuration parameters within the system, which requires studying the hardware along with its available clock sources. In practice, implementing clock reconfigurations in software typically requires modifying specific system registers or memory-mapped bit fields. Once the configuration parameters are understood, the next step is to establish which settings within the clock distribution network are both possible and valid. To utilize the potential energy savings, the energy consumption behavior of each clock configuration as well as its reconfiguration penalties must be known. These parameters can be determined either through manual measurements or by exploration during an initial setup phase. One example of such an online setup phase is *ScaleClock* [10]: Initially, *ScaleClock* determines the system's characteristics through an online performance utilization assessment directly on the device itself. The key insight is to compare the relative execution times of a task at different frequencies: If the execution time of a task decreases proportionally with an increase in frequency, it benefits from higher-frequency clock configurations. Otherwise, the task does not scale well with faster frequencies and therefore performs more energy-efficiently at a lower frequency. Through this process, *ScaleClock* can identify the characteristics of different clock configurations without requiring extensive manual interaction.

2.2. Integration into the Kernel

To implement optimal configurations, the operating system must not only be aware of all available configurations but also actively apply them. In the case of ScaleClock, the insights gained during the initialization phase are employed at runtime within the RIOT operating system: For each task, the system determines its most efficient clock configuration in terms of energy consumption based on the derived performance metrics. The operating system then applies this configuration before the task is dispatched. A different approach called *Power Clocks* [11] is integrated into the Tock operating system. *Power Clocks* does not act upon predetermined performance metrics of a task but instead makes use of the basic observation that, generally, slower clocks are more energy efficient for I/O operations. In comparison, faster clocks are more efficient for computational tasks. To prevent misbehavior due to incorrect configurations, the system tracks potential peripheral-specific constraints. The so-called ClockManager then selects the most energy-efficient clock configuration in accordance with the current clock constraints. Both *ScaleClock* and *Power Clocks* enable dynamic clock management without requiring additional user intervention.

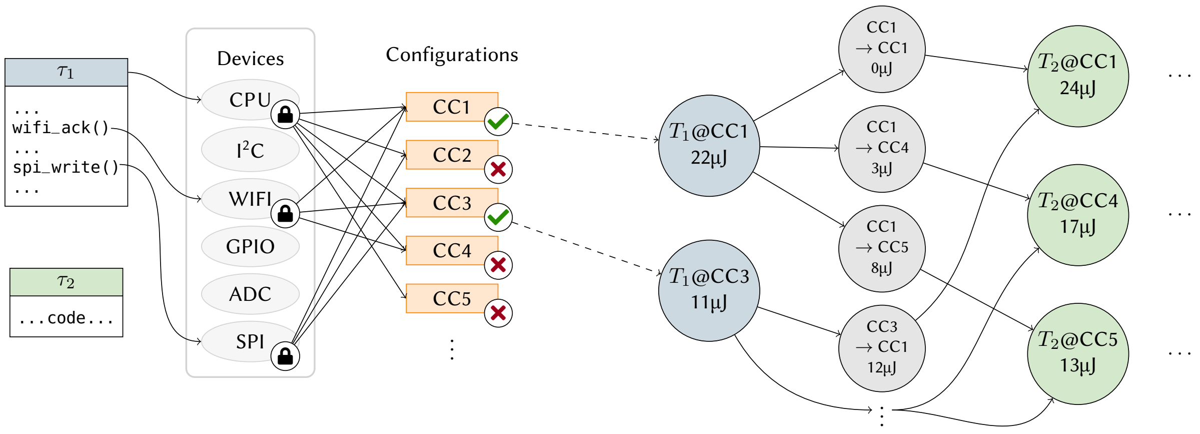

Figure 2: Visualization of the core idea of FusionClock and Crêpe.

2.3. Real-Time Observations for Clock Subsystems

A key challenge in real-time systems is ensuring that real-time guarantees are met. Although strategies exist for scheduling task execution, integrating these strategies with dynamic clock reconfiguration to lower energy consumption is a relatively recent development. Our group has introduced two approaches – FusionClock [8] and Crêpe [12] – that aim to minimize energy consumption in embedded systems while still providing real-time guarantees. The idea of both approaches is visualized in Fig. 2: By evaluating the possible configuration options of a task based on its device requirements, a graph is constructed where each node corresponds to a possible configuration of a task. These nodes are then connected by adding edges that represent the reconfiguration costs incurred when switching from one configuration to another, effectively modeling a mininimum-cost flow problem. A mathematical solver is used to solve the resulting optimization problem with additional constraints regarding execution time, and it returns the best possible (re-)configuration strategy. This results in an energy-optimized system, which still provides real-time guarantees.

3. An IoT-Ready Clock-Subsystem Architecture for Zephyr

As an operating system supporting IoT devices, Zephyr should aim to cater to IoT-specific system characteristics of the target domain. The operation of IoT systems is often subject to resource constraints: Timing constraints require real-time–aware operation, energy constraints necessitate efficient and energy-aware operations. To make the operating system ready to support the management of such constrained applications, the modeling of power domains and sleep states should be considered together with the clock tree configuration. Clocks play a central role as specialized devices that power the processor and peripheral devices. The following sections provide a detailed examination of the current state of clock modeling in Zephyr (Section 3.1) and propose three goals (Sections 3.2, 3.3, 3.4).

3.1. Excerpts of Zephyr's Status Quo

As of now, Zephyr enables the expression of a myriad of features, relations, and requirements regarding clock configurations through device-tree specifications. Zephyr allows specifying available power states for the platform and groups them into predefined categories covering different sleep states [13]. With the properties `min-residency-us` and `exit-latency-us`, reconfiguration penalties for transitioning between different power states can be provided. Additionally, the property `disabling-power-states` enables a device to express the power states in which the device is powered off in the device tree [14]. In general, device trees allow devices to specify their requirements on clock sources [15]. Vendor and

Listing 1: Excerpt of the ESP32-C3 device tree showing the use of the `power-states` property [18].

```
power-states {
        light_sleep: light_sleep {
                compatible = "zephyr,power-state";
                power-state-name = "standby";
                min-residency-us = <200>;
                exit-latency-us = <60>;
        };
        deep_sleep: deep_sleep {
                compatible = "zephyr,power-state";
                power-state-name = "soft-off";
                min-residency-us = <660>;
                exit-latency-us = <105>;
        };
};
```

Listing 2: Excerpt of the device tree for the ESP32-C3 showing the processor clock description [18].

```
cpus {
        cpu@0 {
                cpu-power-states = <&light_sleep &deep_sleep>;
                clock-source = <ESP32_CPU_CLK_SRC_PLL>;
                clock-frequency = <DT_FREQ_M(160)>;
                xtal-freq = <DT_FREQ_M(40)>;
        };
}
```

Listing 3: Excerpt of the device tree for the ESP32-C3 showing the clock specification [18].

```
clock: clock {
        compatible = "espressif,esp32-clock";
        fast-clk-src = <ESP32_RTC_FAST_CLK_SRC_RC_FAST>;
        slow-clk-src = <ESP32_RTC_SLOW_CLK_SRC_RC_SLOW>;
        #clock-cells = <1>;
};
```

device-specific device-tree schemes provide further custom clock information. A schema for Espressif RISC-V CPUs [16], for example, requires specifying available processor clock sources and frequencies. Platforms compatible with the ESP32 family must further provide the properties `fast-clk-src` and `slow-clk-src` as specific clock sources [17].

We illustrate the current state with the example of the device tree for the ESP32-C3 [18]. Listing 1 shows the definition of power states, including their reconfiguration penalties. Listing 2 shows the part of the CPU description that features information about the processor clocks while Listing 3 shows an excerpt of additional clock configuration. A header file ultimately provides macro definitions for the clock sources and supported frequencies [19].

In summary, Zephyr's support for device trees already provides access to much of the information about the clock subsystem required for optimizations, as depicted in Fig. 2. The status quo provides a foundation for a fine-granular modeling of clock trees and the resulting optimization potential to achieve minimal resource consumption.

3.2. Goal 1: Fine-Granular Clock-Tree Modeling

An existing RFC proposes an architecture for a precision clock subsystem in Zephyr, catering to the diverse clocks found in embedded platforms [20]. Among other points, the RFC emphasizes the need for a modular system that supports multiple clock domains and highlights the importance of low-power optimizations in the target domain. The RFC correctly states that optimizations for resource usage in constrained devices require operating-system support for switching between clock sources and clock domains to utilize the configurability of the clock tree fully. We further believe that minimizing resource usage is only possible with a fine-granular modeling of the resource demands of possible clock-tree configurations and transitions between configurations.

While the status quo of clock-tree modeling provides a solid and valuable foundation, we argue that a greater level of detail regarding clock-subsystem information is beneficial. That begins with enriching the clock model with explicit information about the effects on power consumption both when the clock configuration is active and for transitions between clock configurations.

Approaches like FusionClock [8] and Crêpe [12] provide a holistic state enumeration for applications to derive optimal, application-tailored clock configurations. Their underlying clock-tree models, for example, not only subsume the global worst-case for reconfiguration penalties, but also consider context-specific penalties for all possible transitions between clock domains. This knowledge enables the determination of optimal clock configurations for all phases of entire applications.

Ultimately, fine-granular clock-tree specifications in the device trees are not an end in themselves but enable the reconstruction of the clock tree into a resource-consumption–aware model. Such a powerful model paves the way towards resource monitoring and optimizing the system's resource demand.

3.3. Goal 2: Clock-Tree–Aware Resource Management

The resource-constrained characteristics of IoT devices necessiate a deliberate approach to resource management and strict monitoring of resource budgets. Fine-grained clock-tree models enriched with resource-consumption data enable Zephyr to be an operating system that helps applications manage their resource demand and operate efficiently in resource-constrained environments. Knowledge about the effects of clock reconfigurations and the associated overheads positions Zephyr to ensure minimal invasive interventions by the operating system, thereby preserving resources for the applications.

3.4. Goal 3: Workload-Specific Clock-Configuration Optimization

With the help of existing static analysis approaches that provide bounds on resource consumption, the operating system can minimize the resource consumption of applications. Clock-configuration–aware resource bounds allow the determination of alternative clock configurations that still adhere to application constraints but minimize the resource demand. Fine-granular clock models provide the necessary knowledge about reconfiguration overheads to ensure that switching clock configurations between application phases does not nullify the obtained resource savings. The ability to find the optimal clock configuration that leads to the minimal resource demand for different application phases, considering operating-system overheads, would enable Zephyr to provide an efficient execution environment suitable for low-power operation to applications.

4. Conclusion

Advances in battery-free operation and the growing Internet of Things require operating systems like Zephyr to provide environments for execution under resource constraints. Zephyr's existing device-tree support already provides the ability to express comprehensive information about the clock subsystem and power domains. In this positional paper, we argue for advancing the clock-subsystem abstraction to enable Zephyr to cater to the needs of resource-constrained systems. A more fine-granular understanding of the heterogeneous clock subsystems provided by embedded platforms including information about the resource consumption of clock configurations would benefit Zephyr's role as an embedded, real-time–ready operating system.

Fine-grained, resource-aware clock-tree models and clock-tree–aware resource-consumption models are powerful tools for finding application-specific clock configurations that minimize overall resource demand. Integrating such a clock-system abstraction would enable Zephyr to configure systems optimally for the respective applications and further enhance the usability of Zephyr.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] P. Sparks, White paper: The economics of a trillion connected devices, 2017.

[2] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, P. Stenström, The worst-case execution-time problem - overview of methods and survey of tools, ACM Transactions on Embedded Computing Systems (ACM TECS) 7 (2008) 36:1–36:53. doi:10.1145/1347375.1347389.

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

[4] R. Jayaseelan, T. Mitra, X. 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), 2006, pp. 81–90. doi:10.1109/RTAS.2006.17.

[5] D. Trilla, C. Hernández, J. Abella, F. J. Cazorla, Worst-case energy consumption: A new challenge for battery-powered critical devices, IEEE Transactions on Sustainable Computing 6 (2021) 522–530. doi:10.1109/TSUSC.2019.2943142.

[6] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, W. 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, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 24:1–24:25. doi:10.4230/LIPIcs.ECRTS.2018.24.

[7] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, W. 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), 2015, pp. 105–114. doi:10.1109/ECRTS.2015.17.

[8] E. Dengler, P. Raffeck, S. Schuster, P. 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), 2023. doi:10.4230/LIPIcs.ECRTS.2023.6.

[9] Espressif Systems Co., Ltd., ESP32-C3 Technical Reference Manual, 2022. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf, pre-release v0.7.

[10] M. Rottleuthner, T. C. Schmidt, M. 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), 2022, pp. 168–179. URL: https://dl.acm.org/doi/abs/10.5555/3578948.3578964.

[11] H. Chiang, H. Ayers, D. B. Giffin, A. Levy, P. A. 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), 2021, pp. 139–150. URL: https://dl.acm.org/doi/10.5555/3451271.3451284.

[12] E. Dengler, P. Wägemann, Crêpe: Clock-reconfiguration–aware preemption control in real-time systems with devices, in: Proceedings of the 36th Euromicro Conference on Real-Time Systems (ECRTS '24), 2023. doi:10.4230/LIPIcs.ECRTS.2024.10.

[13] The Zephyr Project, Source code, 2025. URL: https://github.com/zephyrproject-rtos/zephyr/blob/main/dts/bindings/power/zephyr%2Cpower-state.yaml.

[14] The Zephyr Project, Zephyr project documentation: Device power management, 2025. URL: https://docs.zephyrproject.org/latest/services/pm/device.html#device-power-management-states.

[15] Devicetree schema tools, 2025. URL: https://github.com/devicetree-org/dt-schema/blob/main/dtschema/schemas/clock/clock.yaml.

[16] The Zephyr Project, Source code, 2025. URL: https://github.com/zephyrproject-rtos/zephyr/blob/main/dts/bindings/cpu/espressif%2Criscv.yaml.
[17] The Zephyr Project, Source code, 2025. URL: https://github.com/zephyrproject-rtos/zephyr/blob/main/dts/bindings/clock/espressif%2Cesp32-clock.yaml.
[18] The Zephyr Project, Source code, 2025. URL: https://github.com/zephyrproject-rtos/zephyr/blob/main/dts/riscv/espressif/esp32c3/esp32c3_common.dtsi.
[19] The Zephyr Project, Source code, 2025. URL: https://github.com/zephyrproject-rtos/zephyr/blob/main/include/zephyr/dt-bindings/clock/esp32c3_clock.h.
[20] Proposal for a precision clock subsystem architecture, 2024. URL: https://github.com/zephyrproject-rtos/zephyr/issues/76335.