

# A Clock-Distribution Abstraction for Resource-Constrained Real-Time Systems in Zephyr

1. International Conference on Zephyr in Science and Education

---

23-24 September 2025

Eva Dengler, Tobias Häberlein, Phillip Raffeck, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



Friedrich-Alexander-Universität  
Technische Fakultät

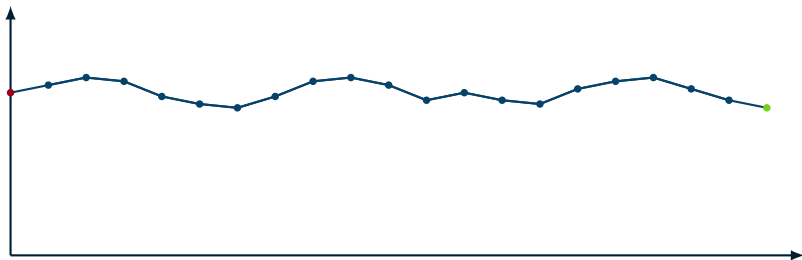
# Application Scenarios with Constraints



# Application Scenarios with Constraints

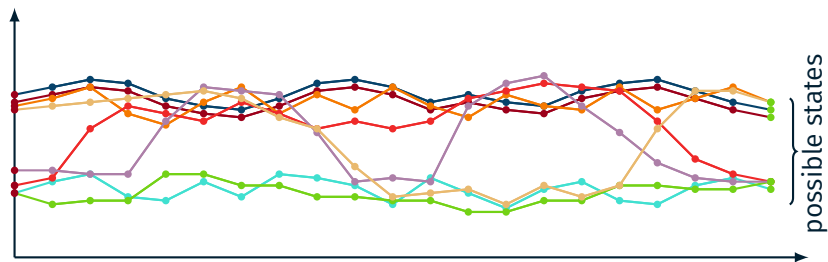


# Abstracting System Behavior for Runtime Guarantees

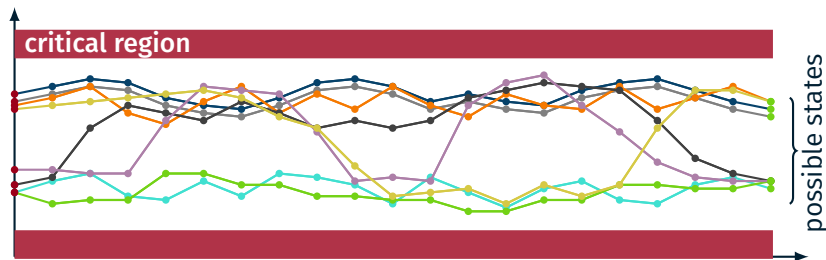




# Abstracting System Behavior for Runtime Guarantees

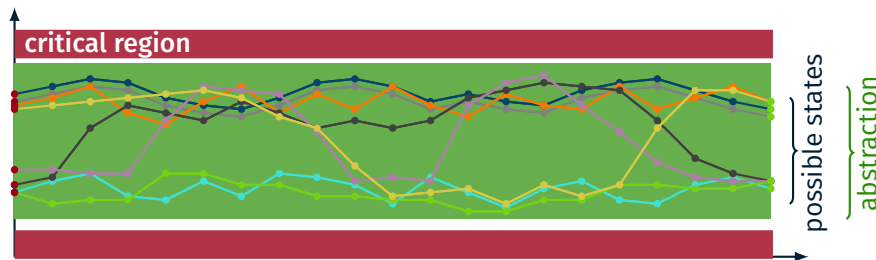


# Abstracting System Behavior for Runtime Guarantees



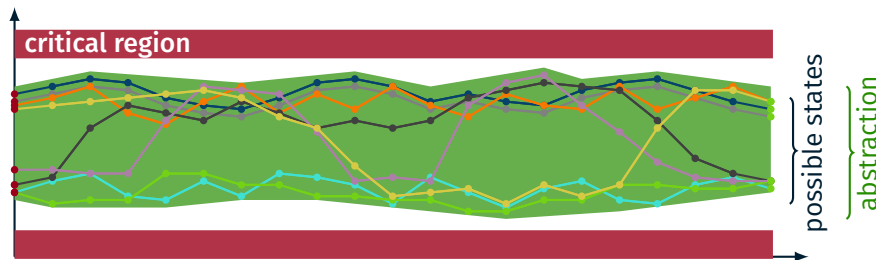
- **Critical region:** timing deadlines, energy budgets, memory bound

# Abstracting System Behavior for Runtime Guarantees



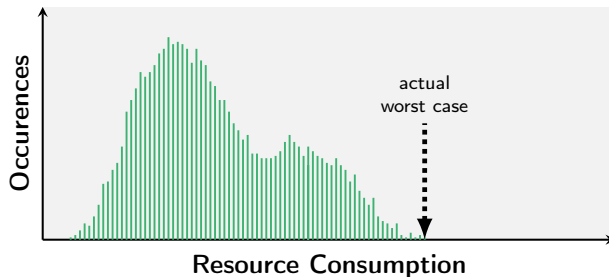
- **Critical region:** timing deadlines, energy budgets, memory bound
- **Soundness** of program analysis: encapsulate all **possible system states**

# Abstracting System Behavior for Runtime Guarantees



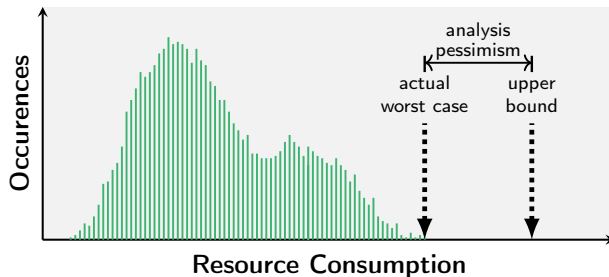
- **Critical region:** timing deadlines, energy budgets, memory bound
- **Soundness** of program analysis: encapsulate all **possible system states**
- **Accuracy:** avoid overestimations

# Requirement of Worst-Case Behavior



- **Time & Energy:** worst-case execution time (WCET) & energy consumption (WCEC)
- Static program code analysis techniques
- Upper resource-consumption bounds  $\leadsto$  runtime guarantees
- Static program analysis to meet safety standards (e.g., ISO 26262, DO-178)
- Contrast to testing 📁 **verify the absence of runtime errors**

# Requirement of Worst-Case Behavior






- **Time & Energy:** worst-case execution time (WCET) & energy consumption (WCEC)
- Static program code analysis techniques
- Upper resource-consumption bounds  $\leadsto$  runtime guarantees
- Static program analysis to meet safety standards (e.g., ISO 26262, DO-178)
- Contrast to testing 📁 **verify the absence of runtime errors**

# Embedded Real-Time Systems

- Portable, energy-efficient computing



# Embedded Real-Time Systems

- Portable, energy-efficient computing
- Equipped with devices   





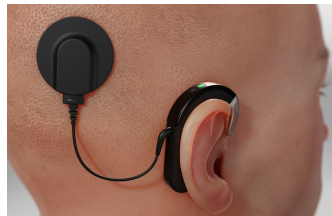
# Embedded Real-Time Systems

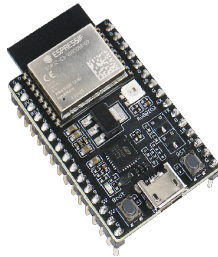
- Portable, energy-efficient computing
- Equipped with devices   , which require energy



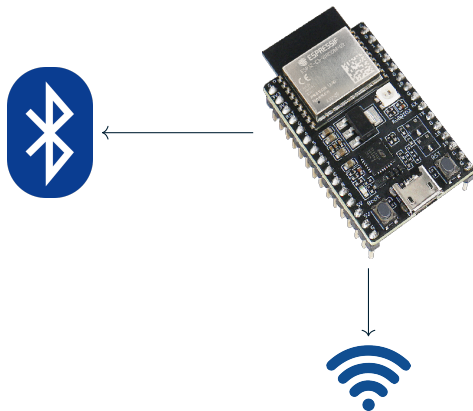
# Embedded Real-Time Systems

- Portable, energy-efficient computing
- Equipped with devices ⚙️ 📶 📶, which require energy
- How to *give timing guarantees*?
- How to *save energy*, to operate as long as possible?

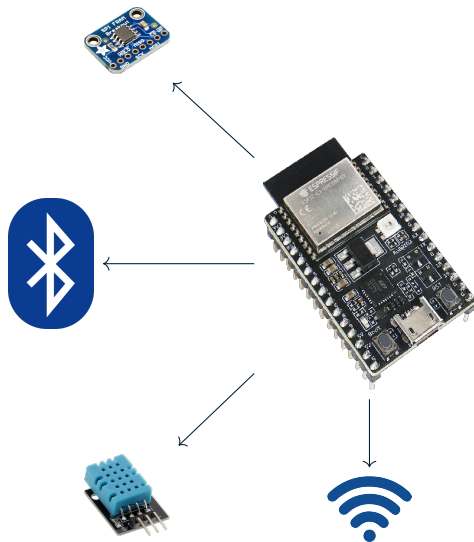




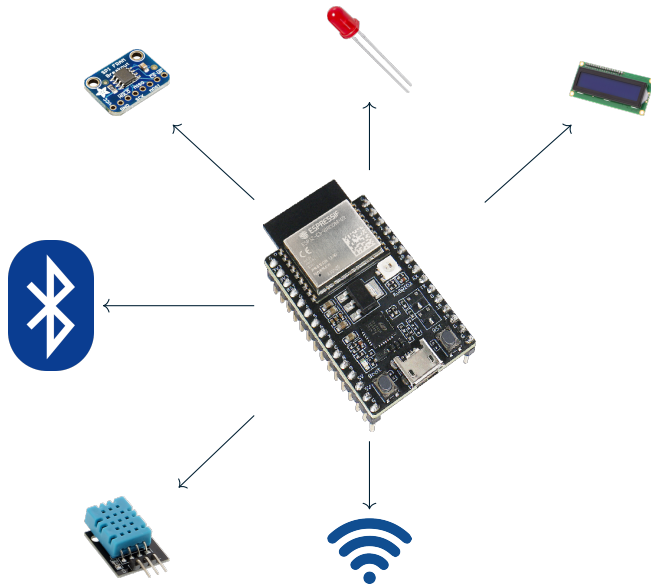
# Devices in Embedded Systems



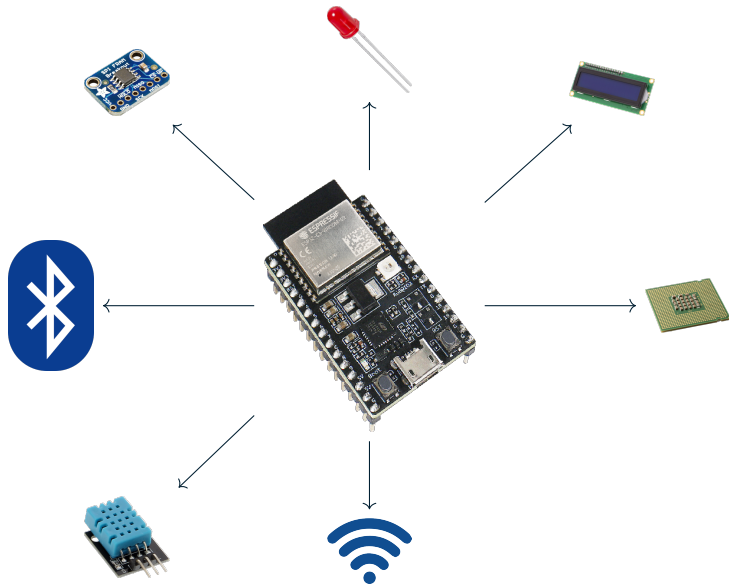
# Devices in Embedded Systems



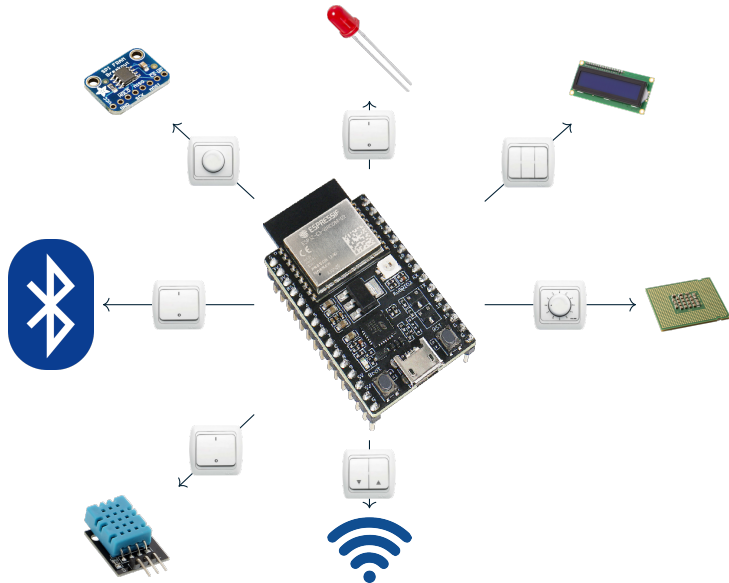
# Devices in Embedded Systems



# Devices in Embedded Systems

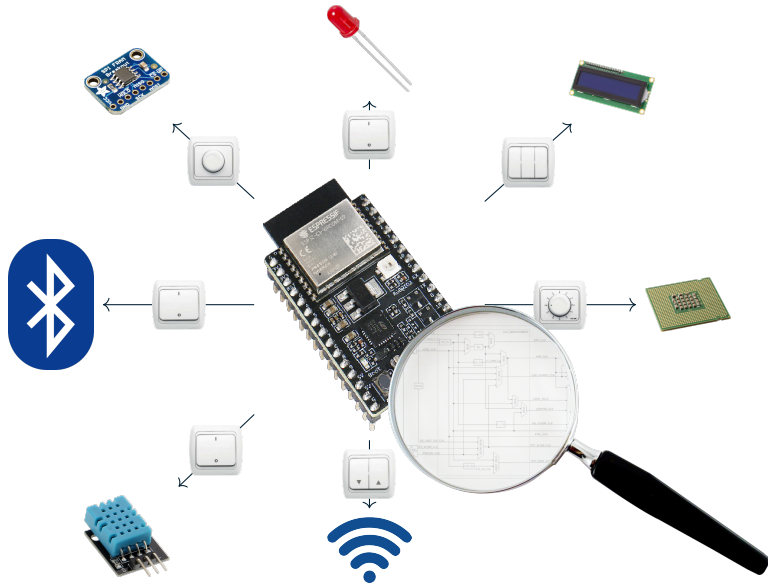


# Devices in Embedded Systems

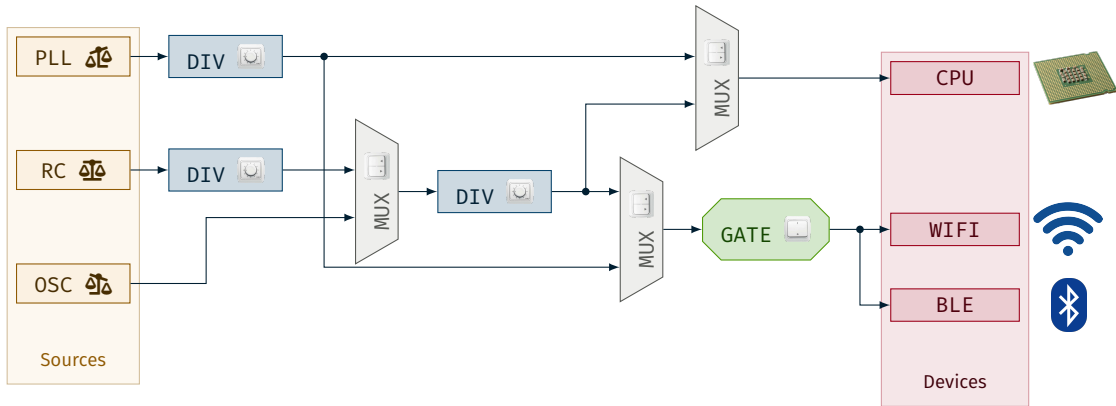




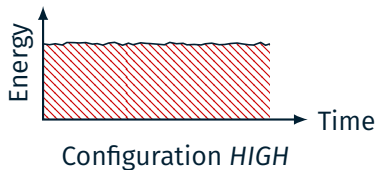
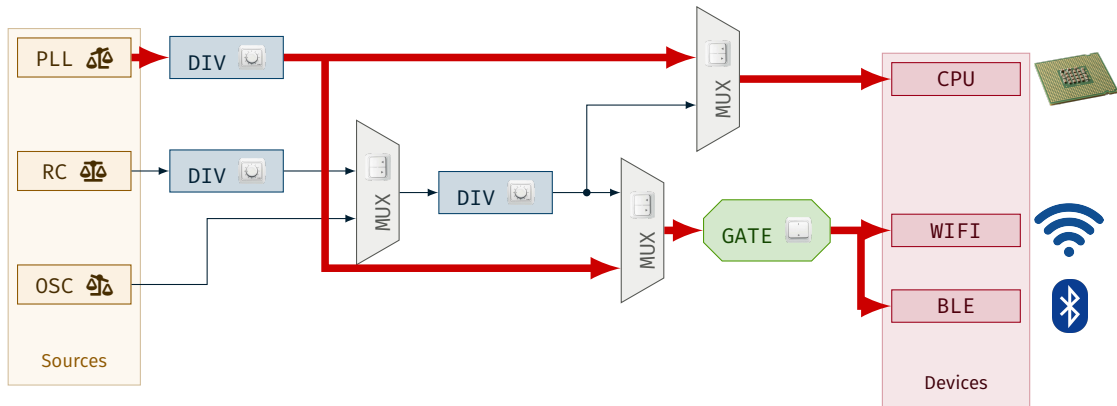
# Devices in Embedded Systems



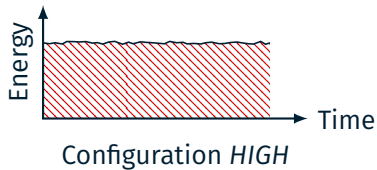
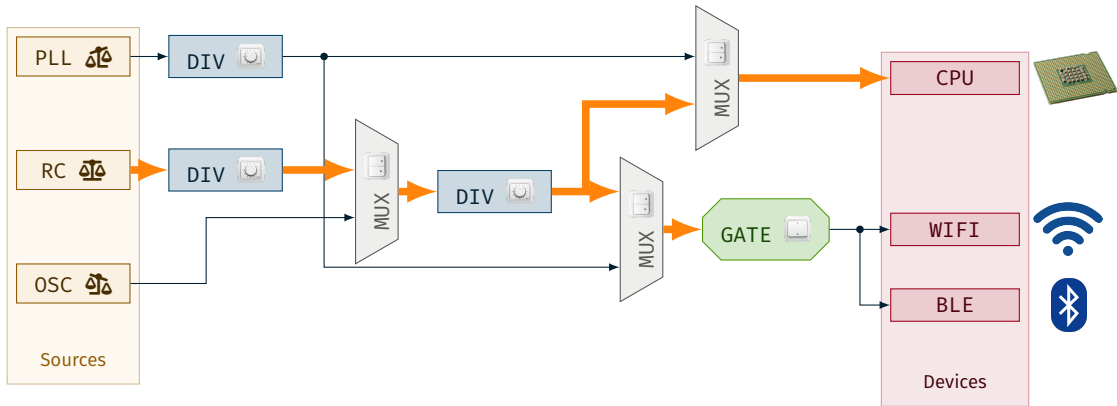
# The Clock Tree



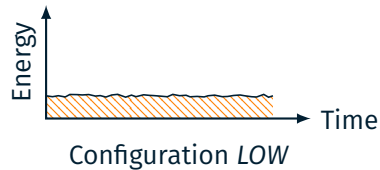
# The Clock Tree



## The Clock Tree



V.S.



- Detecting Clock-Subsystem Configurations
- Integration into Kernel
- Real-Time Observations for Clock Subsystems

- Detecting Clock-Subsystem Configurations:  
⇒ ScaleClock [RSW22] for *RIOT* OS
- Integration into Kernel
- Real-Time Observations for Clock Subsystems

- Detecting Clock-Subsystem Configurations:
  - ⇒ ScaleClock [RSW22] for *RIOT* OS
- Integration into Kernel:
  - ⇒ Power Clocks [Chi+21] for *Tock* OS
- Real-Time Observations for Clock Subsystems

- Detecting Clock-Subsystem Configurations:
  - ⇒ ScaleClock [RSW22] for *RIOT* OS
- Integration into Kernel:
  - ⇒ Power Clocks [Chi+21] for *Tock* OS
- Real-Time Observations for Clock Subsystems:
  - ⇒ FusionClock [Den+23], Crêpe [DW24], WatwaOS [Häb+25]

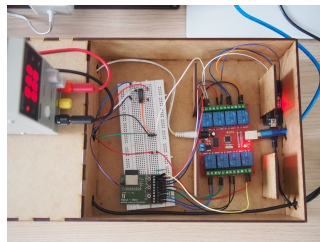


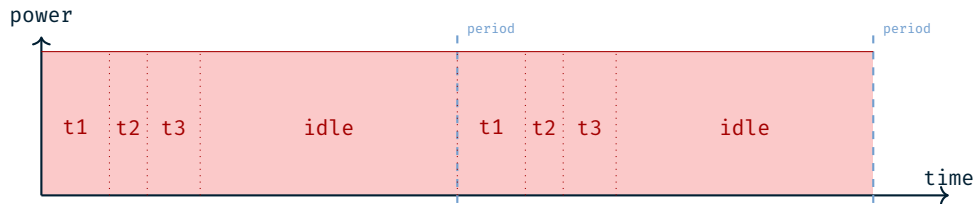
- Detecting Clock-Subsystem Configurations:  
⇒ ScaleClock [RSW22] for *RIOT* OS
- Integration into Kernel:  
⇒ Power Clocks [Chi+21] for *Tock* OS
- **Real-Time Observations for Clock Subsystems:**  
⇒ FusionClock [Den+23], Crêpe [DW24], WatwaOS [Häb+25]

# Real-Time Observations for Clock Subsystems

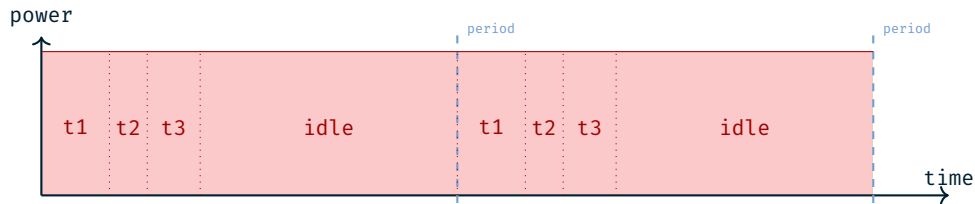
FusionClock [Den+23], Crêpe [DW24],  
Watwa-OS [Häb+25]:

- focus on energy-constrained real-time systems
- embedded platforms
- controllable with clock tree
- guarantees with static analysis
  - ⇒ compliance to real-time constraints
- energy optimization:
  - ⇒ achieve lowest possible energy consumption
- evaluation on real hardware



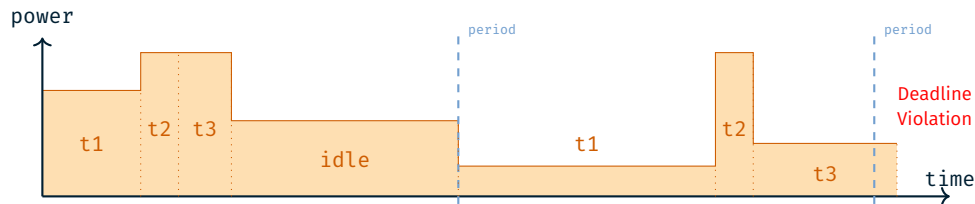


*all-always-on* approach



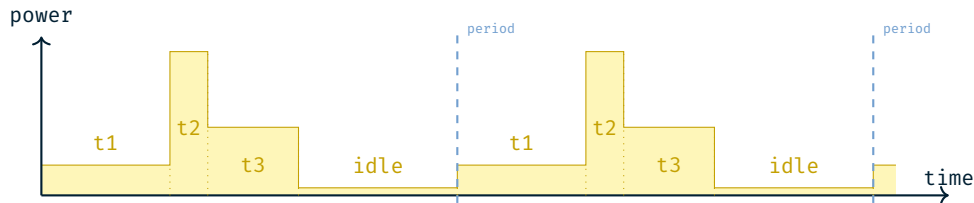
*all-always-on* approach

× minimization of energy consumption



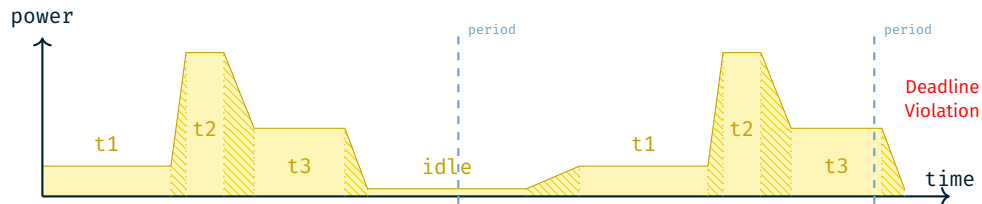
**feedback-based approach:** reconfigurations during execution

- minimization of energy consumption
- × real-time guarantees



**static approach:** analysis before execution

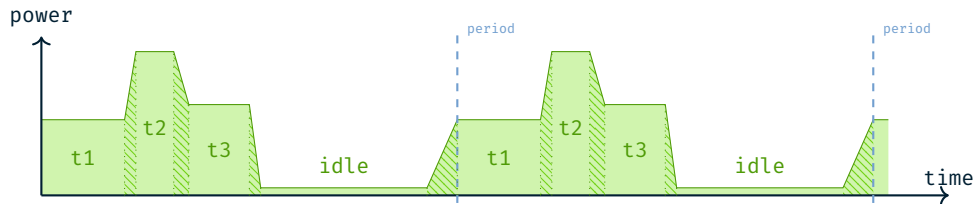
- minimization of energy consumption
- real-time guarantees



static approach without reconfiguration penalties

- minimization of energy consumption
- × real-time guarantees
- × consideration of reconfiguration costs

# Concept of FUSIONCLOCK and CRÊPE



static approach with reconfiguration penalties

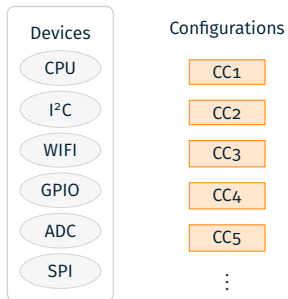
- ✓ minimization of energy consumption
- ✓ real-time guarantees
- ✓ consideration of reconfiguration costs



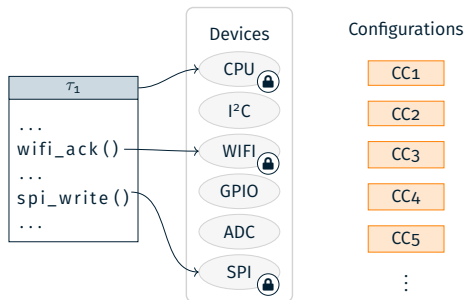
# Clock-Tree Reconfiguration Graph



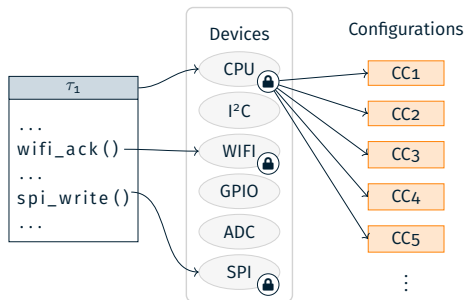
# Clock-Tree Reconfiguration Graph



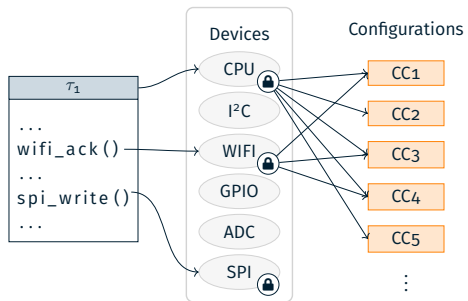
# Clock-Tree Reconfiguration Graph



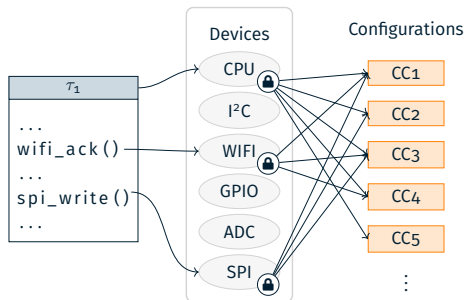
# Clock-Tree Reconfiguration Graph



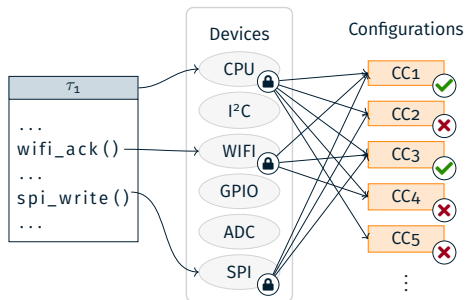
# Clock-Tree Reconfiguration Graph



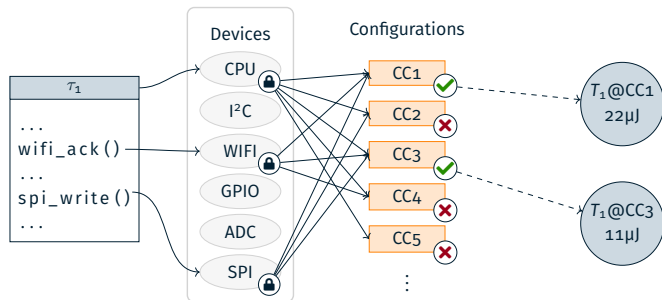
# Clock-Tree Reconfiguration Graph



# Clock-Tree Reconfiguration Graph

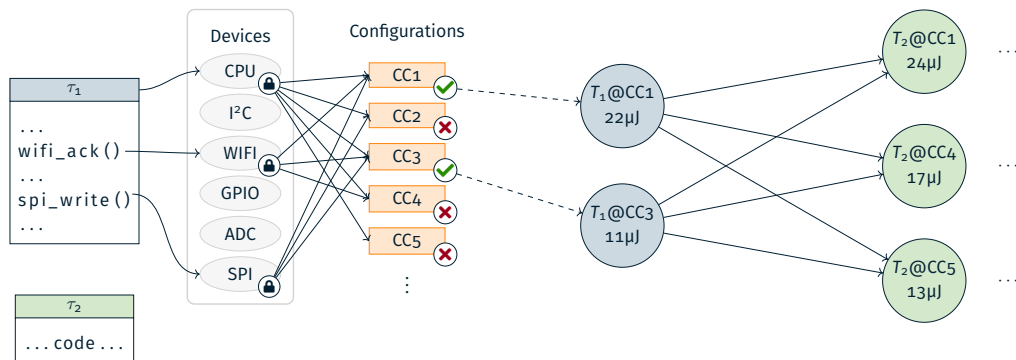


# Clock-Tree Reconfiguration Graph

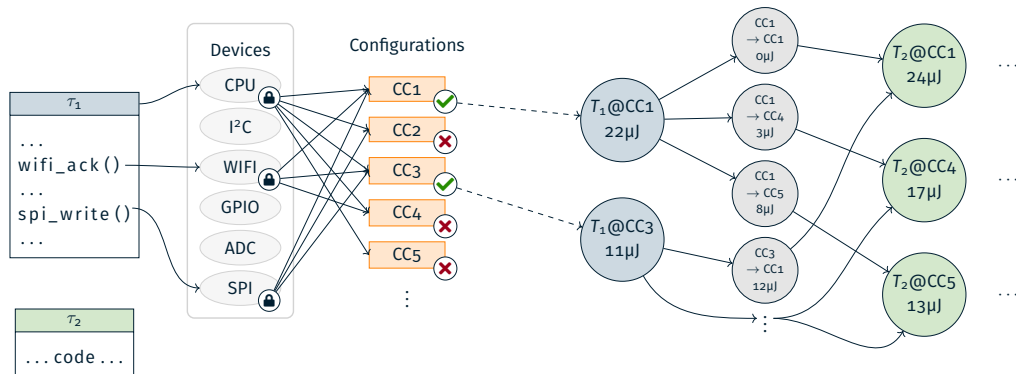




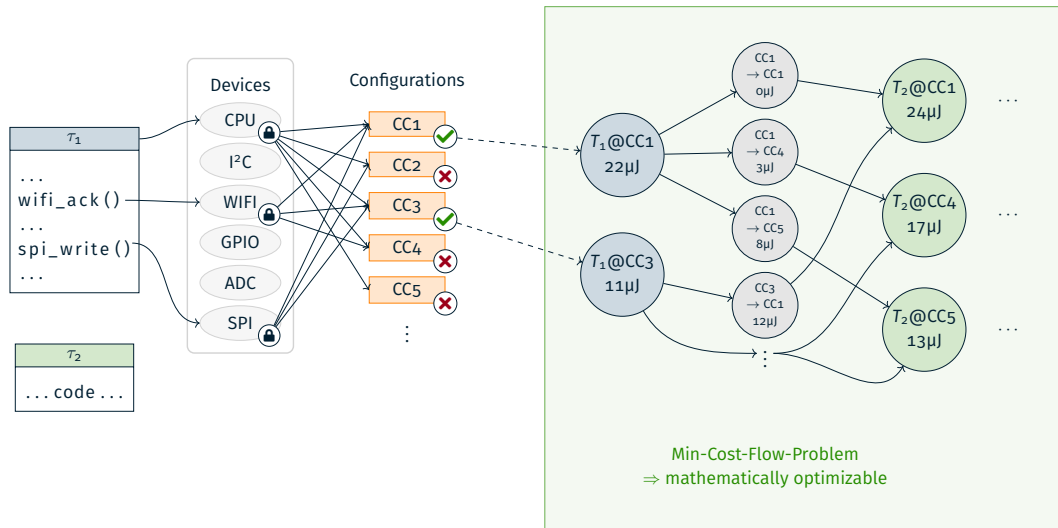
# Clock-Tree Reconfiguration Graph



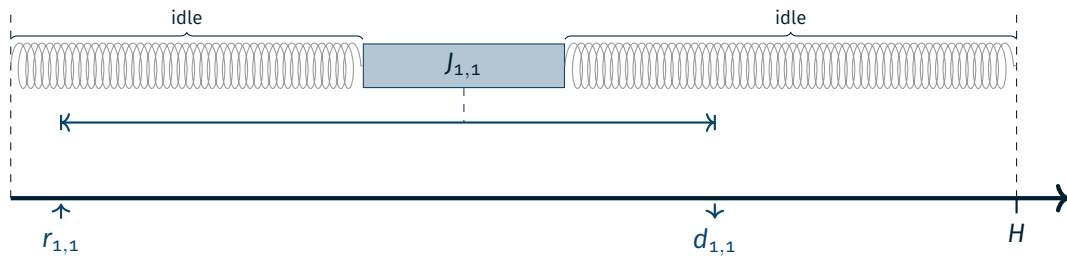
# Clock-Tree Reconfiguration Graph



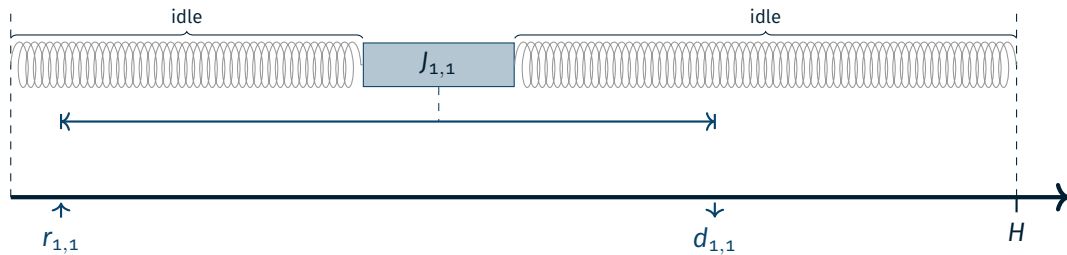
# Clock-Tree Reconfiguration Graph



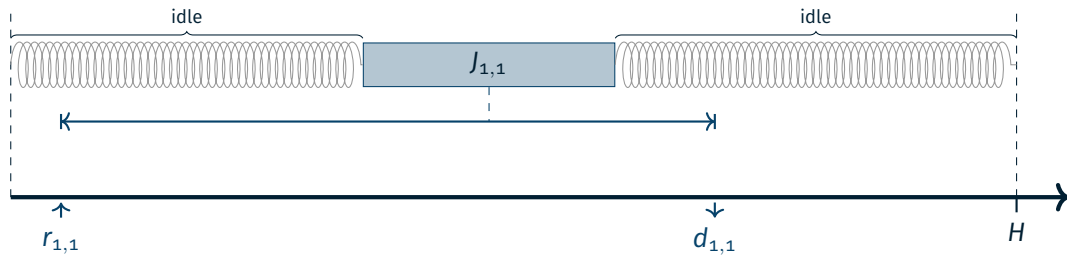
# Incorporation of a Time-Triggered Schedule



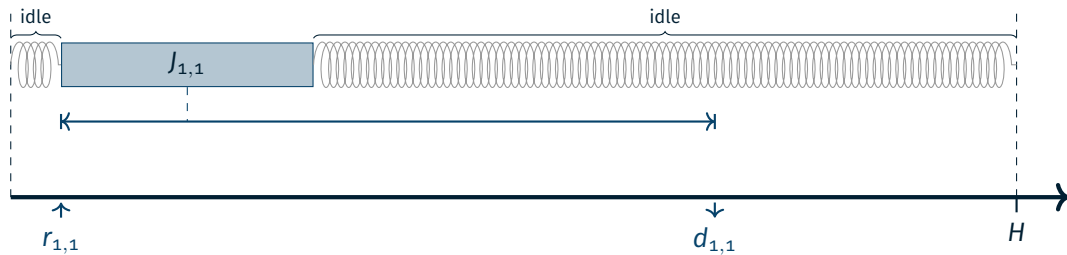
# Incorporation of a Time-Triggered Schedule



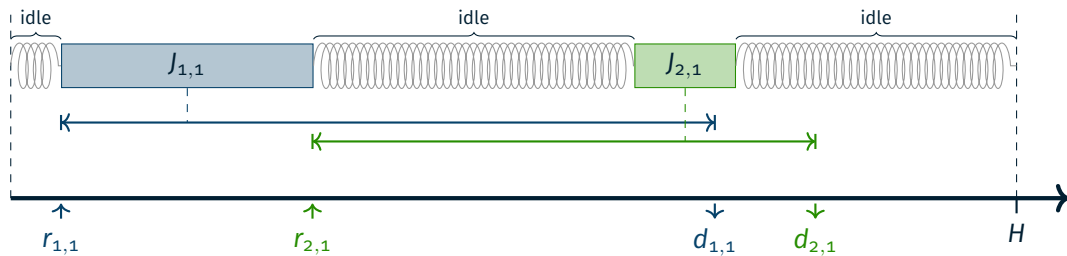
# Incorporation of a Time-Triggered Schedule



# Incorporation of a Time-Triggered Schedule

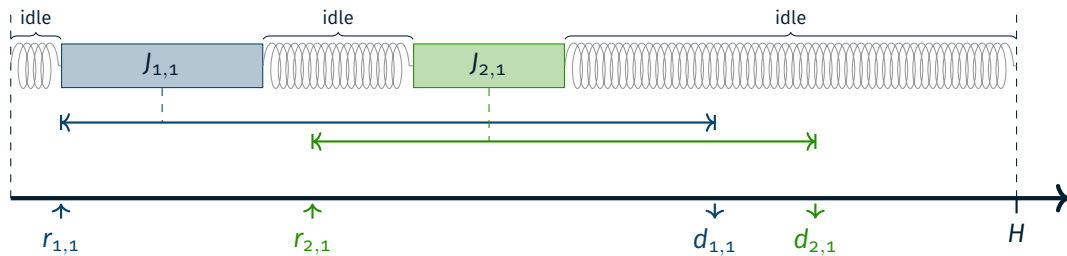


# Incorporation of a Time-Triggered Schedule

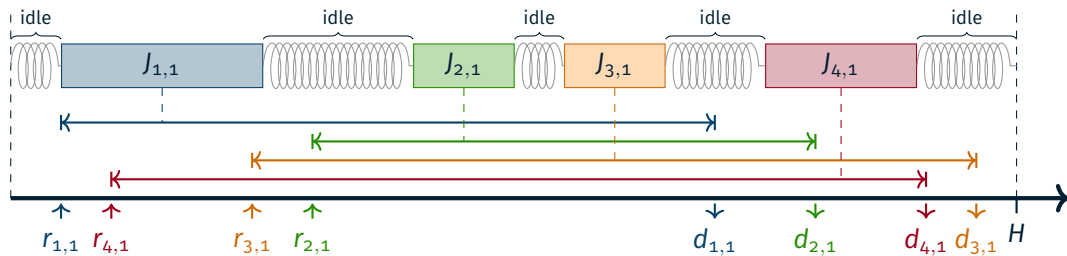




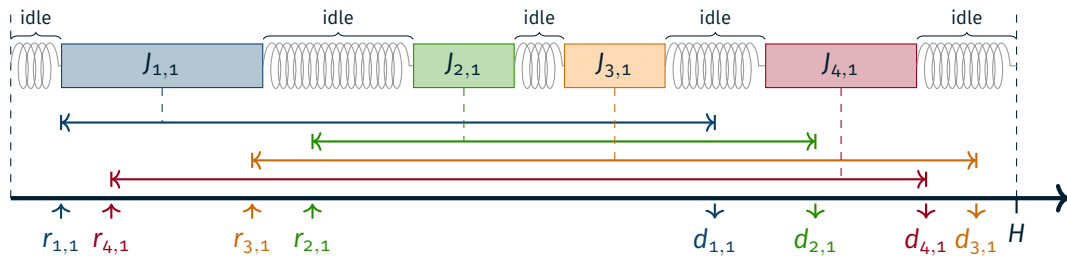
# Incorporation of a Time-Triggered Schedule



# Incorporation of a Time-Triggered Schedule



# Incorporation of a Time-Triggered Schedule



Distributing the slack:

- **reconfiguration penalties**
- **idling**: start times, durations, and configurations

min **energy costs** of jobs and idling options  
+ **energy penalty** for reconfiguration

w.r.t.

constraints in the **clock-tree reconfiguration graph**

all times **sum up** to **hyperperiod**

each job **starts** at or after its **release time**

each job **finishes** before or at its **deadline**

min **energy costs** of jobs and idling options  
+ **energy penalty** for reconfiguration

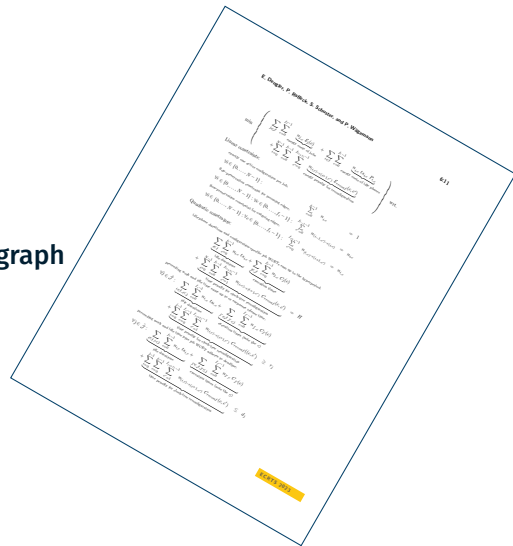
w.r.t.

constraints in the **clock-tree reconfiguration graph**

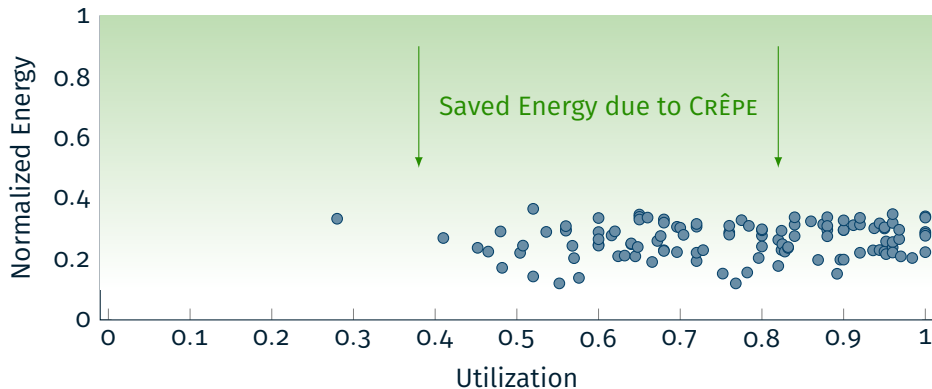
all times **sum up** to **hyperperiod**

each job **starts** at or after its **release time**

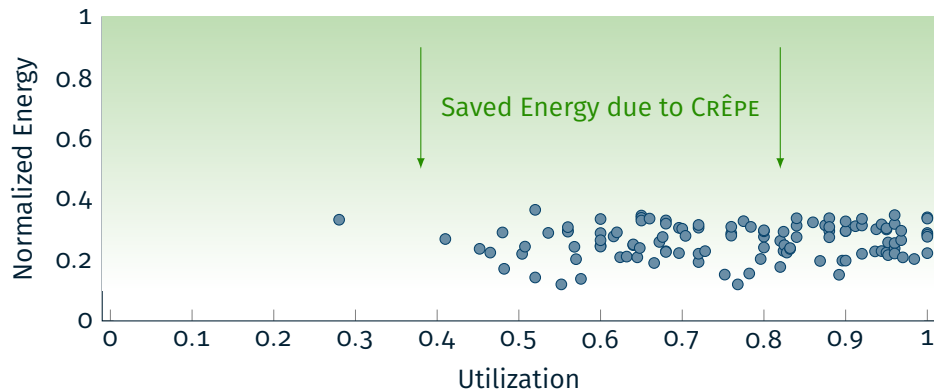
each job **finishes** before or at its **deadline**



## Energy Savings compared to Clock-Agnostic Approach



# Energy Savings compared to Clock-Agnostic Approach



⇒ 70% energy savings

# Clock Subsystems for Zephyr

---



## Device-Tree Specifications:

- specify available power and sleep states
- reconfiguration penalties (with `min-residency-us`, `exit-latency-us`)
- processor clock sources and frequencies
- power management states for devices
- ...

⇒ Zephyr already provides access to much information required for optimization

## Goal 1: Fine-Granular Clock-Tree Modeling

We need...

- ... a modular system that supports multiple clock domains
- ... the resource demands of possible clock-tree configurations and transitions
- ... a greater level of detail for device tree:
  - effects and ...
  - power consumption of clock-tree configurations and transitions

⇒ This knowledge enables us to ...

## **Goal 2: Clock-Tree-Aware Resource Management**

- ... help applications to manage their resource demand
- ... ensure minimal invasive interventions, preserving resources for applications

⇒ This knowledge enables us to ...

## **Goal 2: Clock-Tree-Aware Resource Management**

- ... help applications to manage their resource demand
- ... ensure minimal invasive interventions, preserving resources for applications

## **Goal 3: Workload-Specific Clock-Configuration Optimization**

- ... give bounds on resource consumption with static analysis
- ... determine the best clock configurations for different application phases
- ... use CCs that still adhere to application constraints, but minimize resource demand

⇒ energy-optimized applications!

Current standings:

- Existing work provides energy optimization methods
- Static approaches to guarantee runtime requirements: FUSIONCLOCK and CRÊPE
- Zephyr provides fundamentals for clock-tree-dependent optimizations

Current standings:

- Existing work provides energy optimization methods
- Static approaches to guarantee runtime requirements: FUSIONCLOCK and CRÊPE
- Zephyr provides fundamentals for clock-tree-dependent optimizations

Next steps...?

- Thoughts about our suggestions?
- Enable more fine-grained clock trees - how?
- Improve the clock domains - how?
- Enable Zephyr to include statically determined energy optimizations for applications - how?
  - Determine Application Requirements
  - Integration into compiler
  - ...?

## References

---

- [Chi+21] Holly Chiang et al. “**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.
- [RSW22] 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)*. 2022.
- [Den+23] Eva Dengler et al. “**FusionClock: Energy-Optimal Clock-Tree Reconfigurations for Energy-Constrained Real-Time Systems**”. In: *ECRTS '23*. 2023.
- [DW24] Eva Dengler and Peter Wägemann. “**Crêpe: Clock-Reconfiguration-Aware Preemption Control in Real-Time Systems with Devices**”. In: *ECRTS '24*. 2024.
- [Häb+25] Tobias Häberlein et al. “**WatwaOS: A Framework for Worst-Case-Aware Tailoring and Whole-System Analysis of Energy-Constrained Real-Time Systems**”. In: *RTSS '25*. 2025.