# WATWAOS: A Framework for Worst-Case-Aware Tailoring and Whole-System Analysis of Energy-Constrained Real-Time Systems

December 3, 2025
46th IEEE Real-Time Systems Symposium, Boston, MA, USA

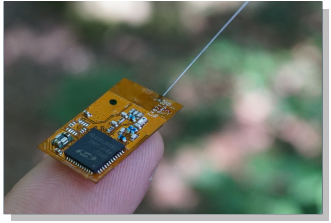
Tobias Häberlein, Eva Dengler, Phillip Raffeck, Peter Wägemann

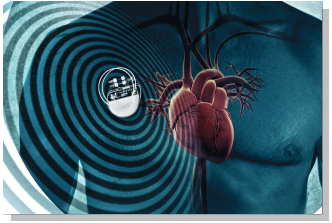
Friedrich-Alexander-Universität Erlangen-Nürnberg
Supported by the DFG under the grant WA 5186/1-1 (Watwa)

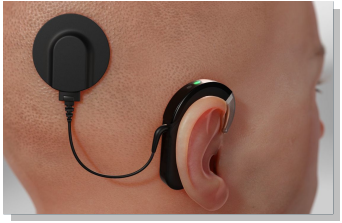
Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Faculty of Engineering

# Embedded Real-Time Systems

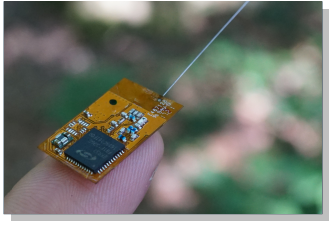


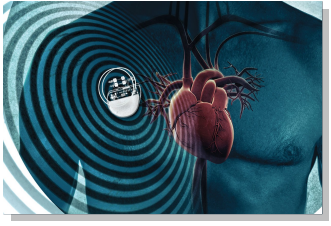


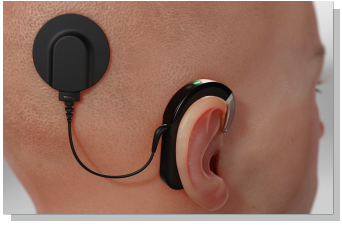


## Requirements

- Safe worst-case timing and energy consumption bounds (WCET/WCEC)
- Energy-efficient operation







# Energy-Constrained Real-Time Systems

- Energy efficiency of embedded systems depends on
  - The clock frequency
  - The type of performed operation

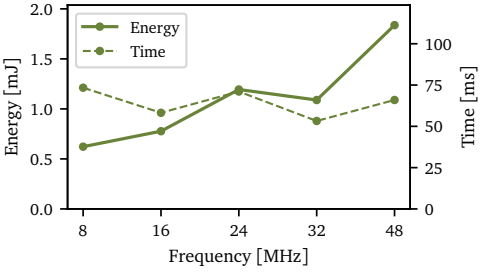
# Energy-Constrained Real-Time Systems

- Energy efficiency of embedded systems depends on
  - The clock frequency
  - The type of performed operation
- Typical use case: read sensor data $\rightarrow$ process the obtained data

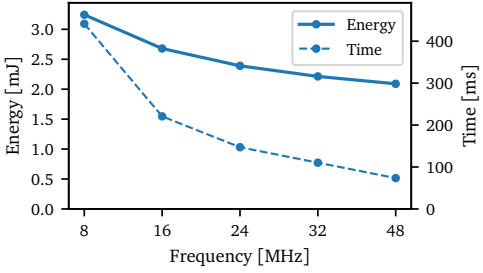
# Energy-Constrained Real-Time Systems

- Energy efficiency of embedded systems depends on
  - The clock frequency
  - The type of performed operation
- Typical use case: read sensor data → process the obtained data

reading sensor data




processing sensor data



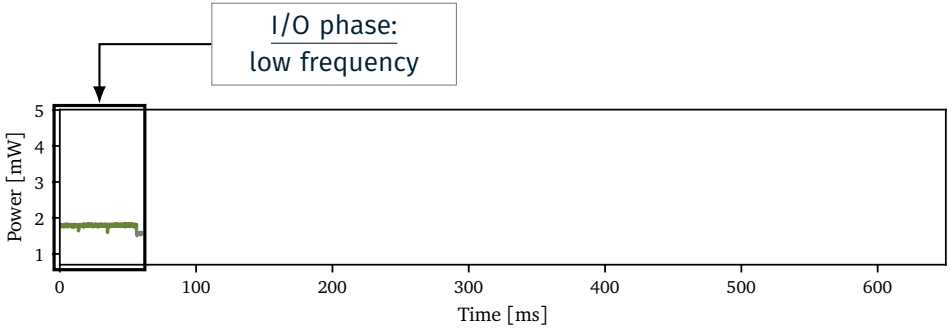
# Dynamic Frequency Scaling Approaches

## Dynamic Approaches

- Detect I/O and compute phases and change clock frequency accordingly

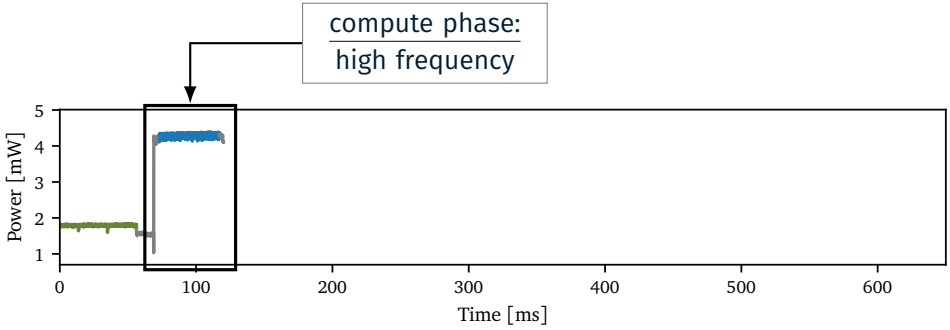
## Dynamic Approaches

- Detect I/O and compute phases and change clock frequency accordingly



## Dynamic Approaches

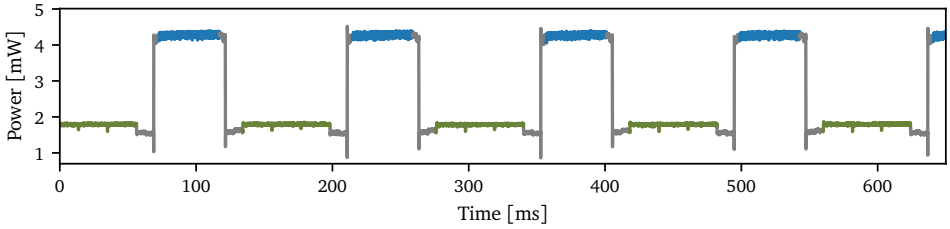
- Detect I/O and compute phases and change clock frequency accordingly



# Dynamic Frequency Scaling Approaches

## Dynamic Approaches

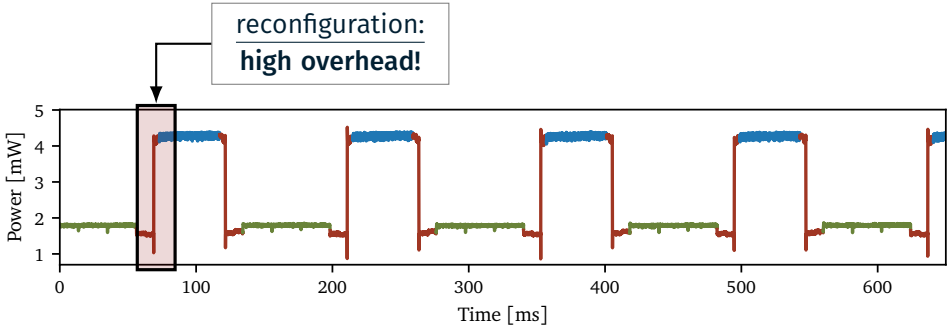
- Detect I/O and compute phases and change clock frequency accordingly



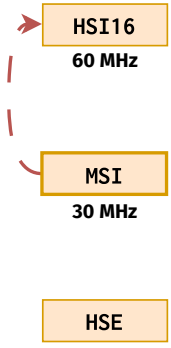
# Dynamic Frequency Scaling Approaches

## Dynamic Approaches

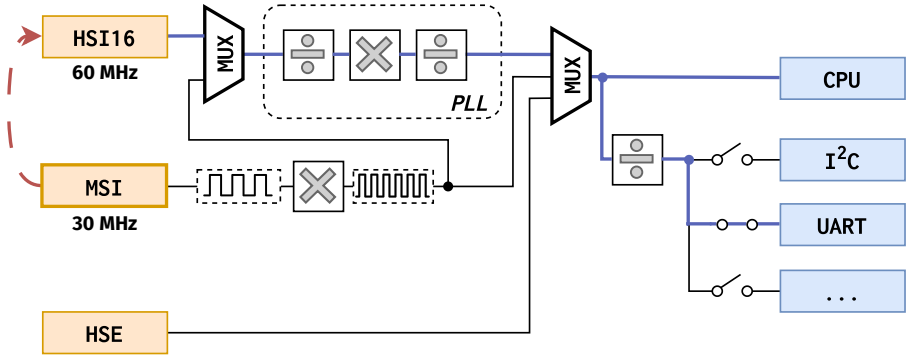
- Detect I/O and compute phases and change clock frequency accordingly



# Clock Trees



# Clock Trees



# Problem Statement

How can we **save energy** to operate as long as possible,
while still being able to give **real-time guarantees**?

# Problem Statement

How can we **save energy** to operate as long as possible,
while still being able to give **real-time guarantees**?

- Minimize energy consumption while providing safe WCET/WCEC bounds
- Keep analysis and optimization times reasonable

# Approach - Outline

How can we **save energy** to operate as long as possible,
while still being able to give **real-time guarantees**?

**In a Nutshell:**

- Minimize energy consumption while providing safe WCET/WCEC bounds
  - Construct global Power-State–Transition Graph [ECRTS '18] → perform IPET to get WCET/WCEC
- Keep analysis and optimization times reasonable

# Approach - Outline

How can we **save energy** to operate as long as possible,
while still being able to give **real-time guarantees**?

**In a Nutshell:**

- Minimize energy consumption while providing safe WCET/WCEC bounds
  - Construct global Power-State–Transition Graph [ECRTS '18] → perform IPET to get WCET/WCEC
  - Introduce clock reconfiguration points → solve multiple ILPs
- Keep analysis and optimization times reasonable

# Approach - Outline

How can we **save energy** to operate as long as possible,
while still being able to give **real-time guarantees**?

**In a Nutshell:**

- Minimize energy consumption while providing safe WCET/WCEC bounds
  - Construct global Power-State–Transition Graph [ECRTS '18] → perform IPET to get WCET/WCEC
  - Introduce clock reconfiguration points → solve multiple ILPs
- Keep analysis and optimization times reasonable
  - Reduce ILP complexity with *node merging* techniques
  - Make use of special solver features

# System Model

## Hardware

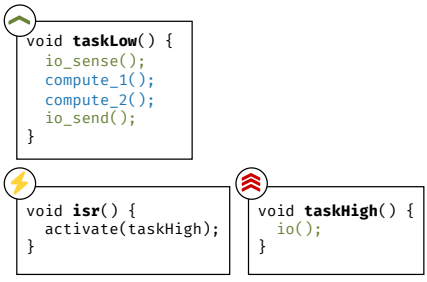
- Resource-constrained system-on-chip platform
- Predictable architecture, no caches
- Complex clock-configuration networks

## Operating System

- Fixed-priority, preemptive scheduler
- Statically known task set
- Independent tasks
- Any number of sporadic task activations ($\rightarrow$ interrupts) allowed

# The WatwaOS Approach (1)

Application:



# The WATWAOS Approach (1)

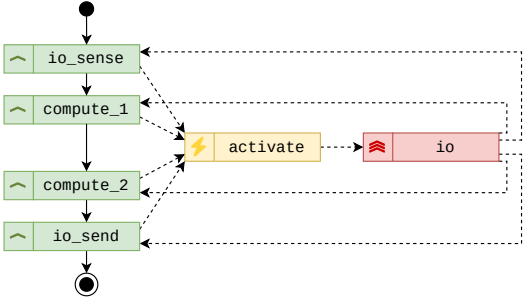
## Application:

```
void taskLow() {
  io_sense();
  compute_1();
  compute_2();
  io_send();
}
```

```
void isr() {
  activate(taskHigh);
}
```

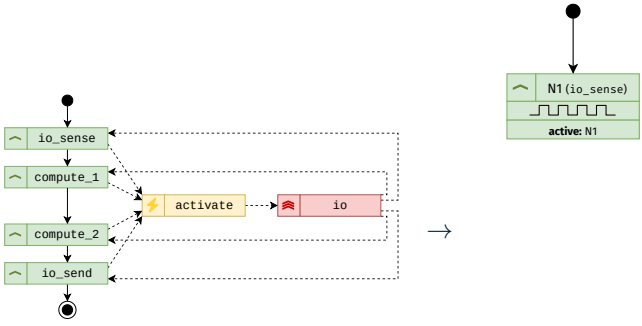
```
void taskHigh() {
  io();
}
```

## Control-Flow Graph:



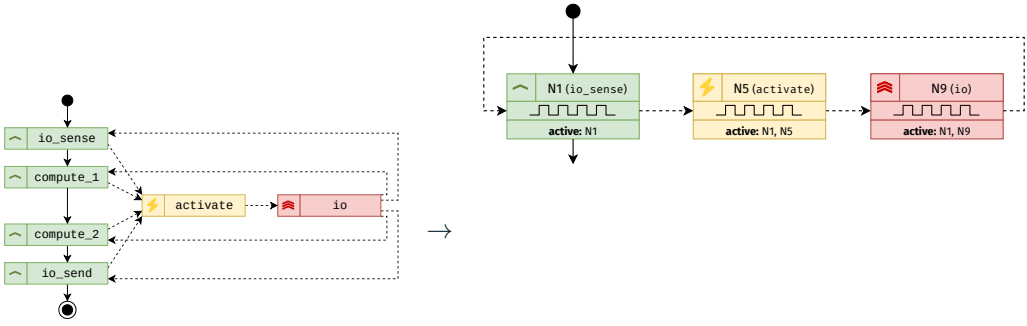
# The WATWAOS Approach (2)

Control-Flow Graph → Power-State–Transition Graph (PSTG):



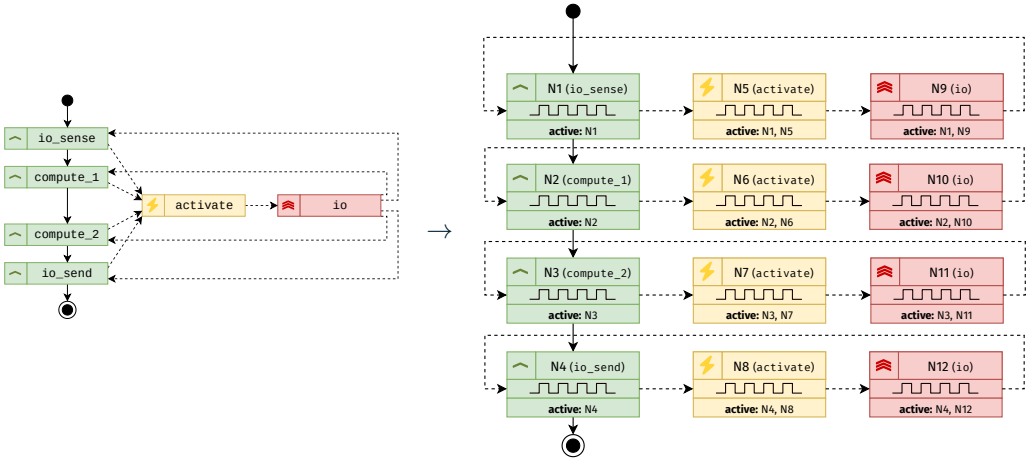
# The WatwaOS Approach (2)

Control-Flow Graph → Power-State–Transition Graph (PSTG):



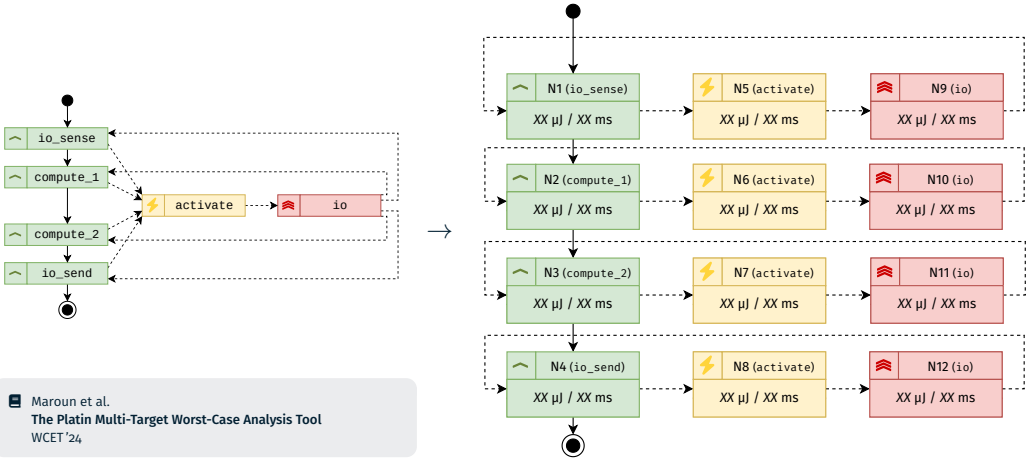
# The WatwaOS Approach (2)

Control-Flow Graph → Power-State–Transition Graph (PSTG):



# The WATWAOS Approach (2)

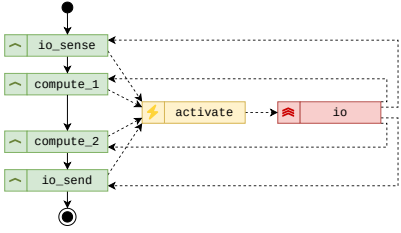
Control-Flow Graph → Power-State–Transition Graph (PSTG):



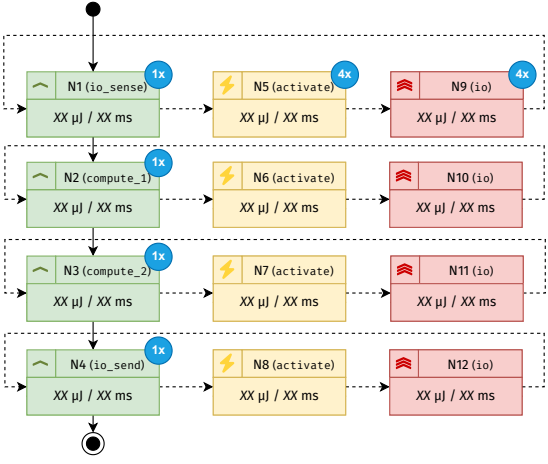

Maroun et al.
**The Platin Multi-Target Worst-Case Analysis Tool**
WCET '24

# The WATWAOS Approach (2)

Control-Flow Graph → Power-State–Transition Graph (PSTG) → ILP

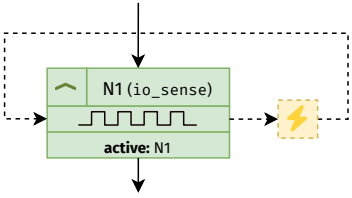



→




Maroun et al.
**The Platin Multi-Target Worst-Case Analysis Tool**
WCET '24

# Integer Linear Program



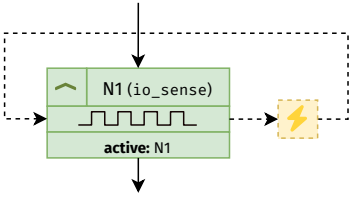

**Constraints per node:**

- $\sum f(\text{Input Edge}) = \sum f(\text{Output Edge})$
- $f(\text{Node}) = \sum f(\text{Input Edges}_{\neq \text{ISR}})$
- ...

**Other important constraints**

- Loops: derive upper bound from source code
- ISRs: determine frequency using minimum inter-arrival time

# Integer Linear Program




**Constraints per node:**

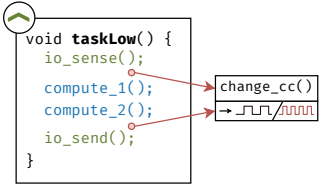
- $\sum f(\text{Input Edge}) = \sum f(\text{Output Edge})$
- $f(\text{Node}) = \sum f(\text{Input Edges}_{\neq \text{ISR}})$
- ...

**Other important constraints**

- Loops: derive upper bound from source code
- ISRs: determine frequency using minimum inter-arrival time

✔ Provide safe WCET/WCEC bounds

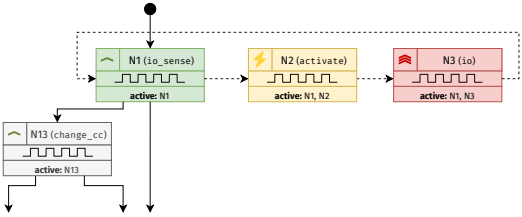
# Adding Clock Reconfiguration Points



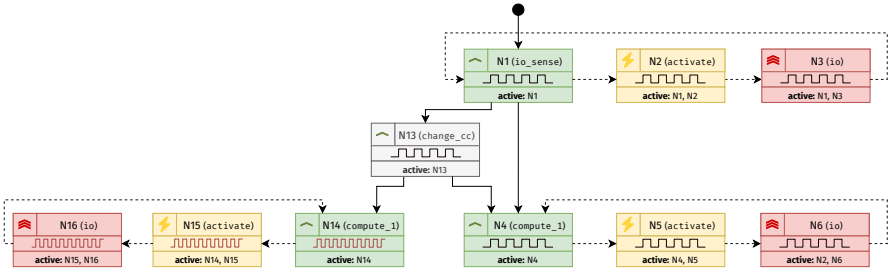

**New:** add clock reconfiguration nodes

- before/after single I/O operations
- before/after a loop with I/O operations
- at the start/end of a task

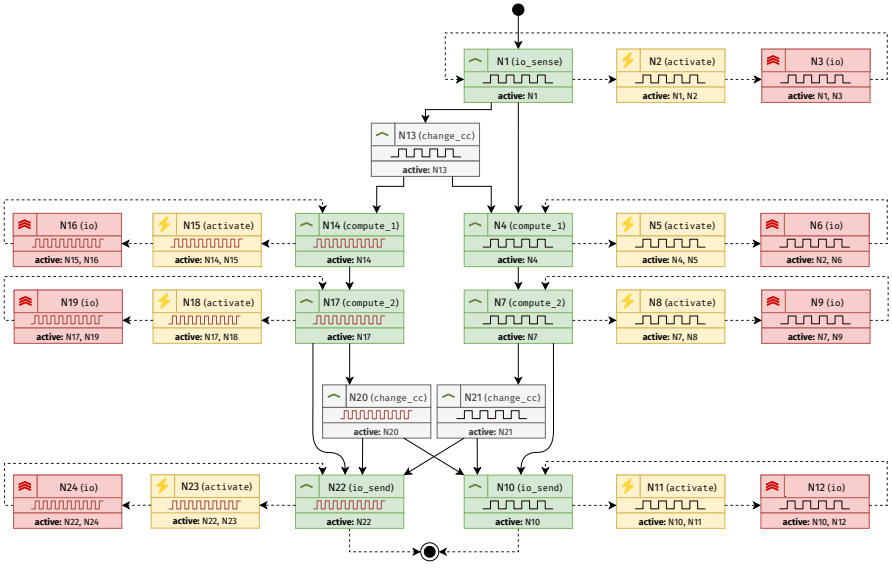
# Adding Reconfiguration Points to the PSTG



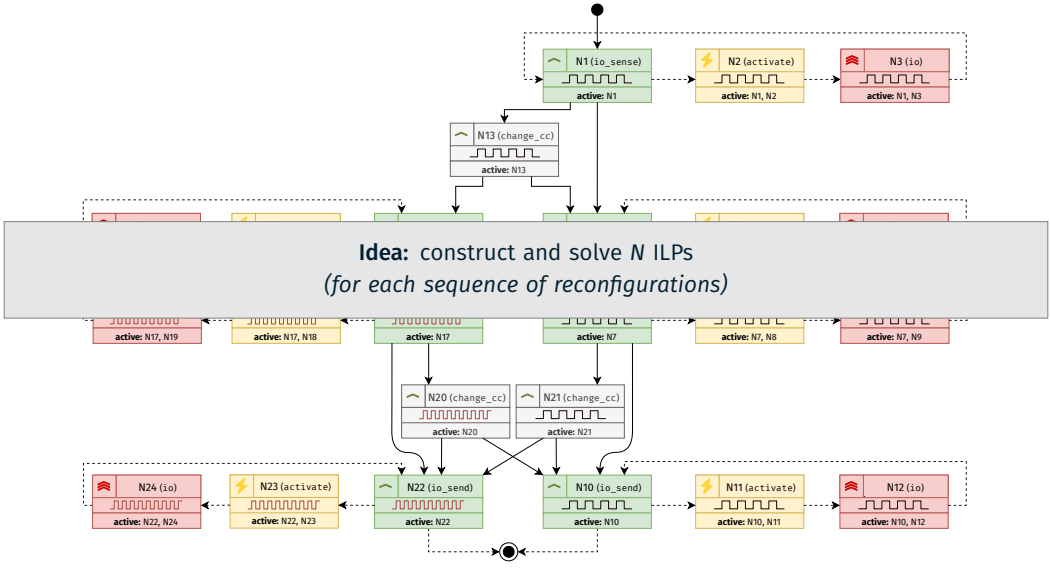
# Adding Reconfiguration Points to the PSTG



# Adding Reconfiguration Points to the PSTG

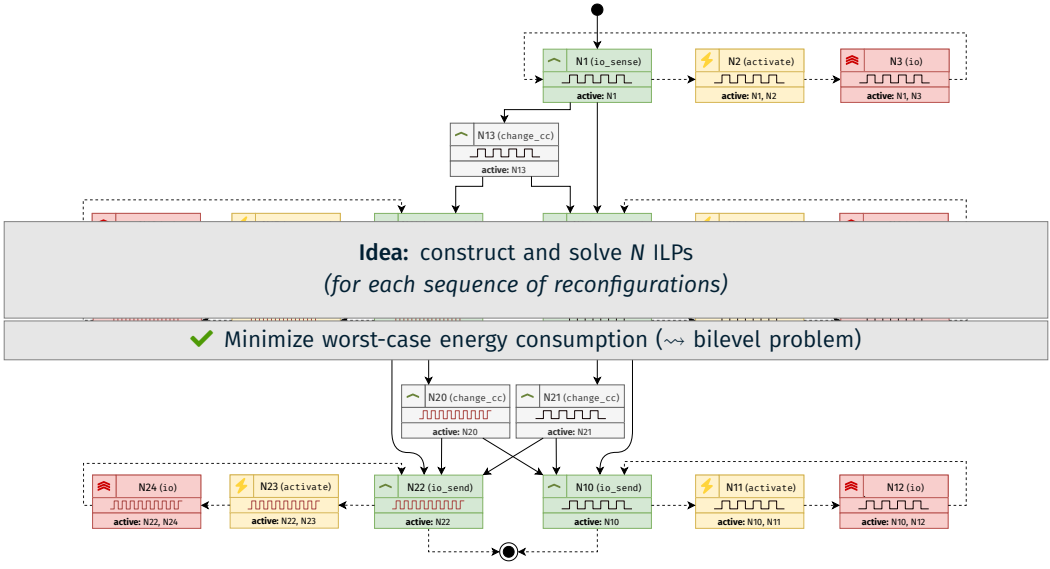


# Adding Reconfiguration Points to the PSTG

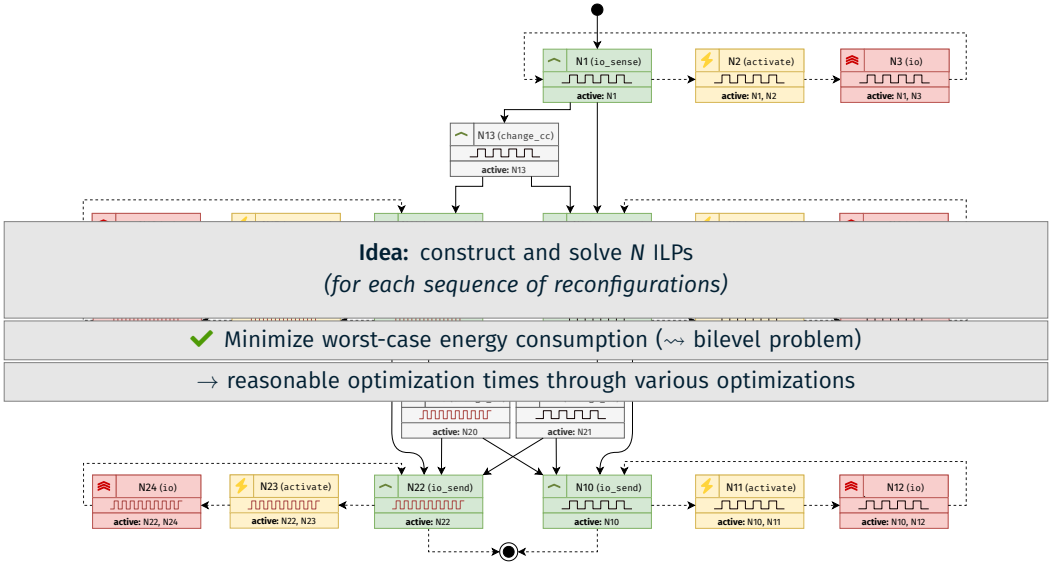



**Idea:** construct and solve *N* ILPs
*(for each sequence of reconfigurations)*

# Adding Reconfiguration Points to the PSTG



# Adding Reconfiguration Points to the PSTG

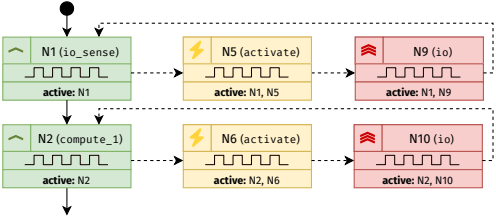



**Idea:** construct and solve *N* ILPs
*(for each sequence of reconfigurations)*

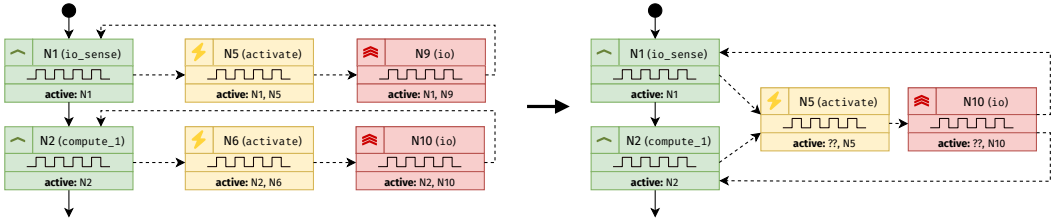
✔ Minimize worst-case energy consumption ($\rightsquigarrow$ bilevel problem)

$\rightarrow$ reasonable optimization times through various optimizations

# Optimization: Exploiting Similarities in the Graph Construction



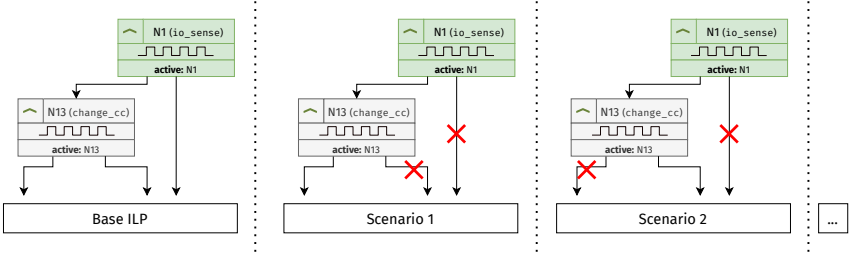
# Optimization: Exploiting Similarities in the Graph Construction




- Idea: merge nodes and edges whenever possible
  - Reduces the amount of constraints and variables in the ILP
  - New constraints necessary to retain context-sensitive information

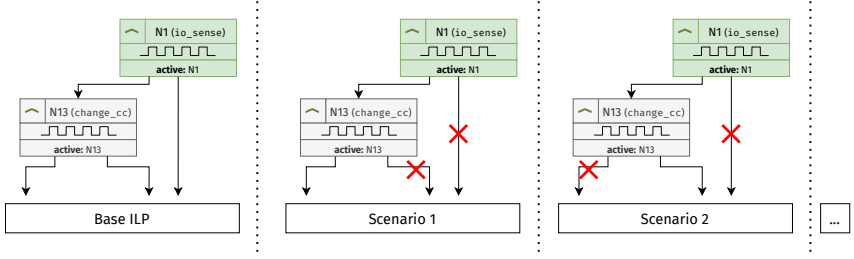
# Optimization: Exploiting Similarities in the ILP Formulation

- Previously: construct *N* ILPs
- Now: construct 1 base ILP with multiple *scenarios*
  - Feature of the *Gurobi* solver



# Optimization: Exploiting Similarities in the ILP Formulation

- Previously: construct *N* ILPs
- Now: construct 1 base ILP with multiple *scenarios*
  - Feature of the *Gurobi* solver

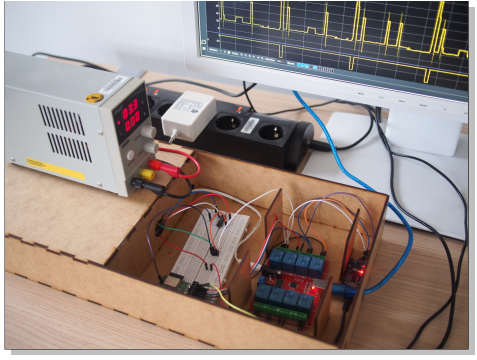


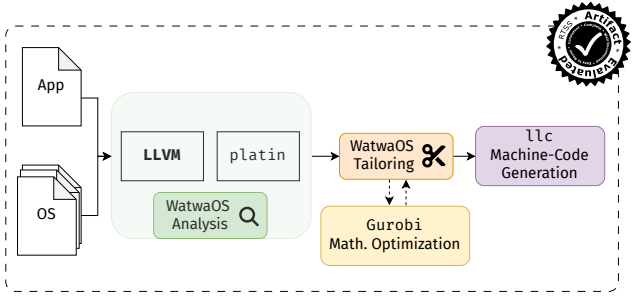

✔ Keep analysis and optimization times reasonable

# Evaluation

# Setup

**ESP32-C3**: 32-bit RISC-V CPU, max. 160 MHz, 400 kB SRAM
**Energy measurement device**: Joulescope JS220





# Evaluation: Optimization Time (1)

| Interrupts | Merging | Variables | Constraints | Solving time *(first 3000 scenarios)* |
|---|---|---|---|---|
| 0 | Off | 98 | 87 | 0.10 s |
| | On | 98 | 87 | 0.10 s |

# Evaluation: Optimization Time (1)

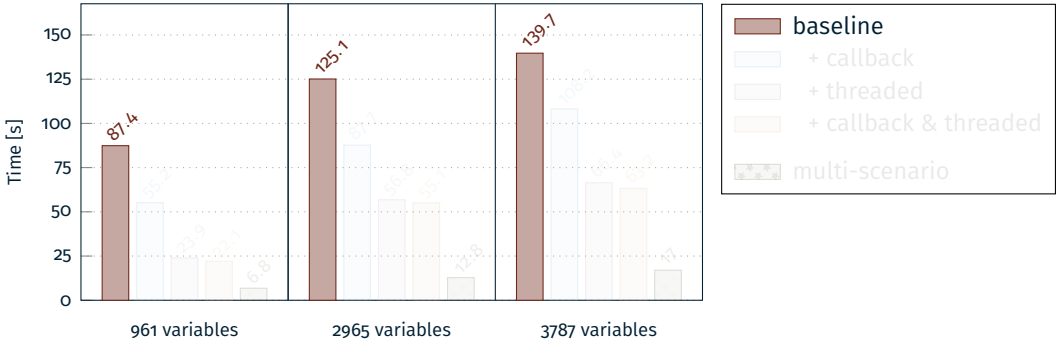
| Interrupts | Merging | Variables | Constraints | Solving time *(first 3000 scenarios)* |
|---|---|---|---|---|
| 0 | Off | 98 | 87 | 0.10 s |
| | On | 98 | 87 | 0.10 s |
| 1 | Off | 984 | 663 | 2.74 s |
| | On | 270 | 238 | 2.29 s **-16.4 %** |
| 2 | Off | 1854 | 1229 | 9.65 s |
| | On | 430 | 386 | 5.26 s **-45.5 %** |
| 3 | Off | 2724 | 1795 | 13.34 s |
| | On | 590 | 534 | 6.56 s **-50.8 %** |
| 4 | Off | 3594 | 2361 | 15.68 s |
| | On | 750 | 682 | 9.24 s **-41.1 %** |

# Evaluation: Optimization Time (1)

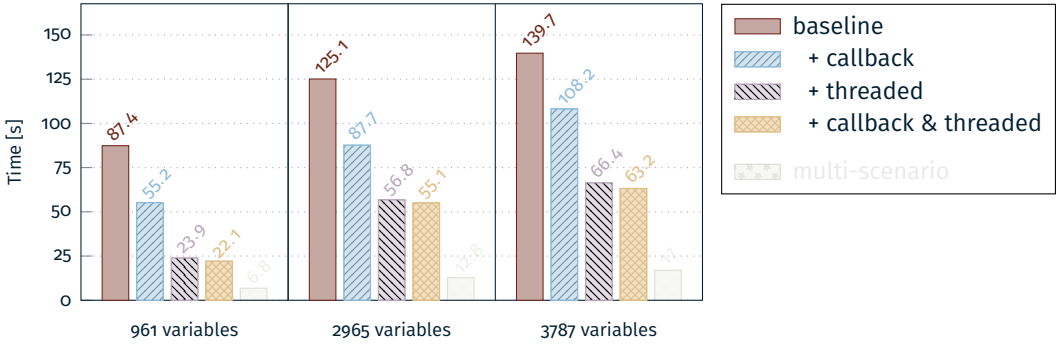
| Interrupts | Merging | Variables | Constraints | Solving time *(first 3000 scenarios)* |
|---|---|---|---|---|
| 0 | Off | 98 | 87 | 0.10 s |
| | On | 98 | 87 | 0.10 s |
| 1 | Off | 984 | 663 | 2.74 s |
| | On | 270 | 238 | 2.29 s **-16.4 %** |
| 2 | Off | 1854 | 1229 | 9.65 s |
| | On | 430 | 386 | 5.26 s **-45.5 %** |
| 3 | Off | 2724 | 1795 | 13.34 s |
| | On | 590 | 534 | 6.56 s **-50.8 %** |
| 4 | Off | 3594 | 2361 | 15.68 s |
| | On | 750 | 682 | 9.24 s **-41.1 %** |

✔ Up to 50 % solving time reduction by exploiting similarities in the PSTG

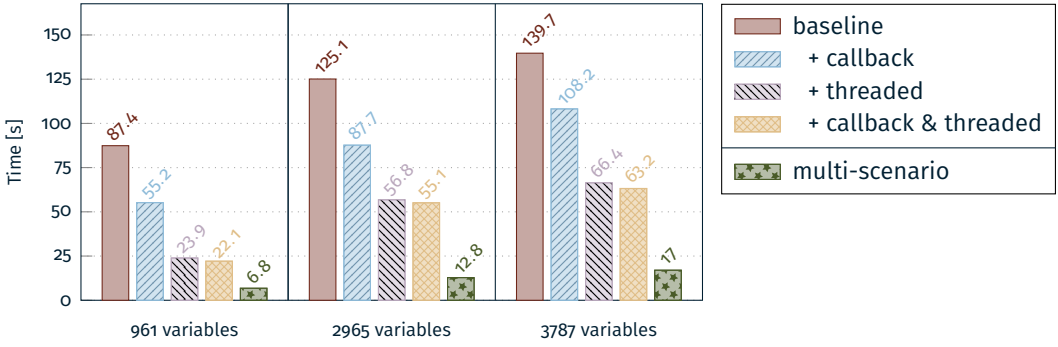
# Evaluation: Optimization Time (2)



# Evaluation: Optimization Time (2)

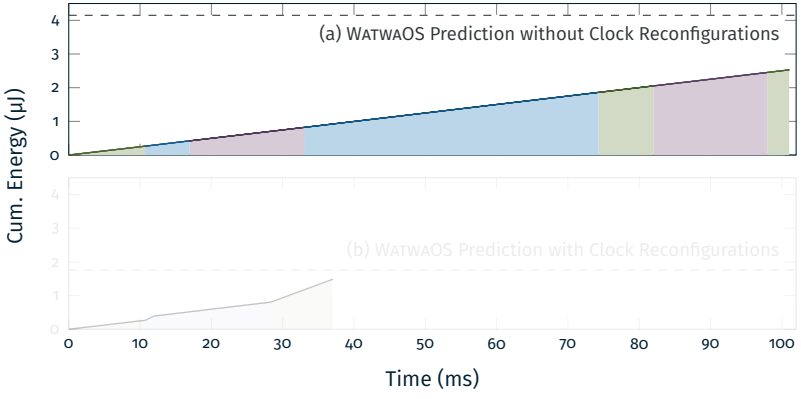


# Evaluation: Optimization Time (2)



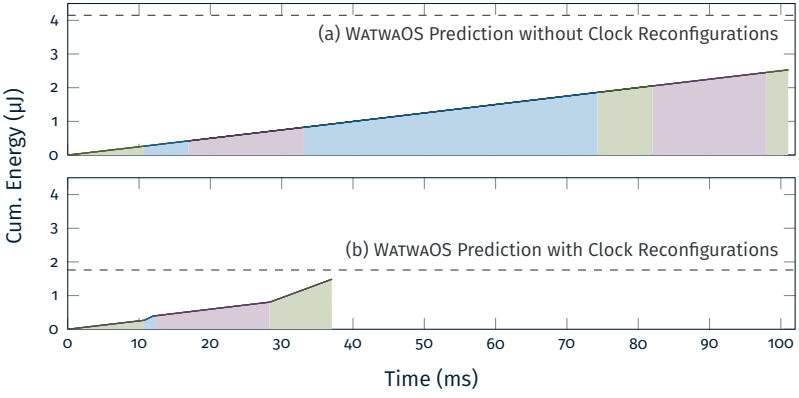

✔ Significant optimization time reduction by optimizing the ILP solving process

# Evaluation: Application Example




| | Predicted | Measured |
|---|---|---|
| **(a)** | 4.15 µJ | 2.52 µJ |

# Evaluation: Application Example




| | Predicted | Measured |
|---|---|---|
| **(a)** | 4.15 µJ | 2.52 µJ |
| **(b)** | 1.76 µJ | 1.48 µJ |

✔ Improved energy efficiency compared to static clock configuration

# Conclusion

# Conclusion

## WatwaOS

- ✔ Whole-system analysis and optimization framework
  - → Integrated into the LLVM compiler framework

# Conclusion

## WatwaOS

- ✔ Whole-system analysis and optimization framework
  - → Integrated into the LLVM compiler framework
- ✔ Minimizes energy consumption while providing safe WCET/WCEC bounds
  - → Construction of a Power-State–Transition Graph with reconfiguration nodes

# Conclusion

## WatwaOS

- ✔ Whole-system analysis and optimization framework
  - → Integrated into the LLVM compiler framework
- ✔ Minimizes energy consumption while providing safe WCET/WCEC bounds
  - → Construction of a Power-State–Transition Graph with reconfiguration nodes
- ✔ Achieves reasonable analysis and optimization times
  - → Node merging and multi-scenario ILPs

# Conclusion

## WATWAOS

- ✔ Whole-system analysis and optimization framework
  - → Integrated into the LLVM compiler framework
- ✔ Minimizes energy consumption while providing safe WCET/WCEC bounds
  - → Construction of a Power-State–Transition Graph with reconfiguration nodes
- ✔ Achieves reasonable analysis and optimization times
  - → Node merging and multi-scenario ILPs

WATWAOS Source Code and Artifact: