Friedrich-Alexander-Universität
Technische Fakultät

Seminararbeit

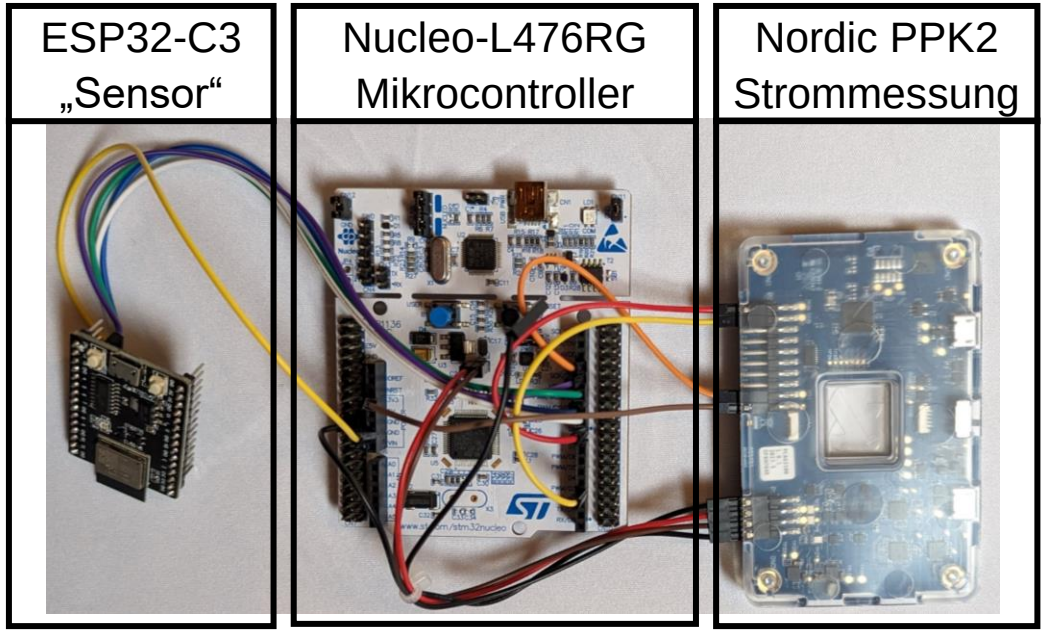
Dynamic Clock Control

Tobias Häberlein
10.01.2024

Motivation

Ausgangslage

- Energiebeschränkte Mikrocontroller (→ Batterien, Solarenergie)
- Anwendungsfall: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)



Ziel: Energieeffizienz

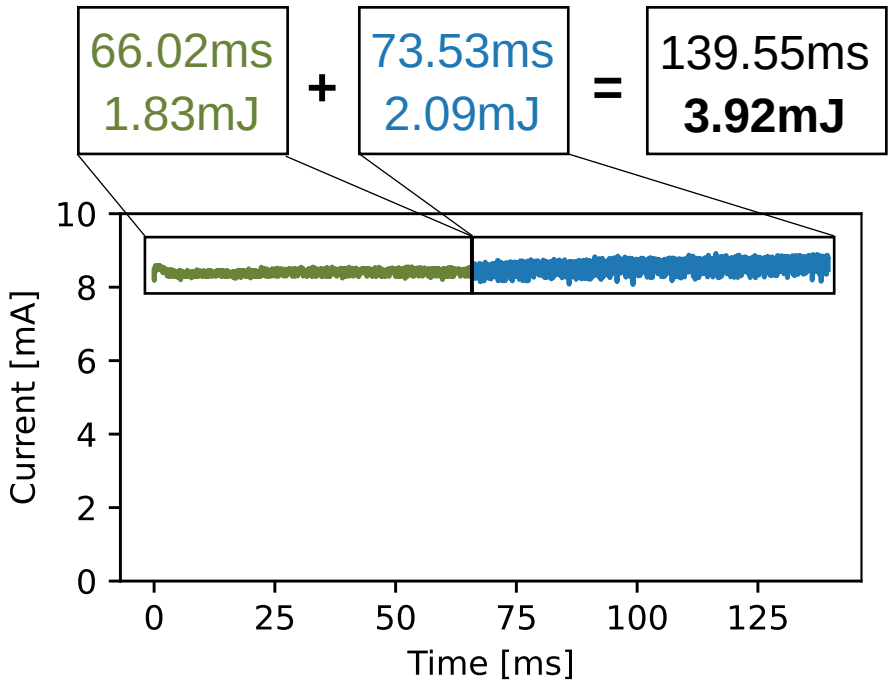
(ohne Berücksichtigung von Zeitschranken)

Motivation

Beispielanwendung

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)

Strategie 1: *race-to-idle (**48MHz**)*

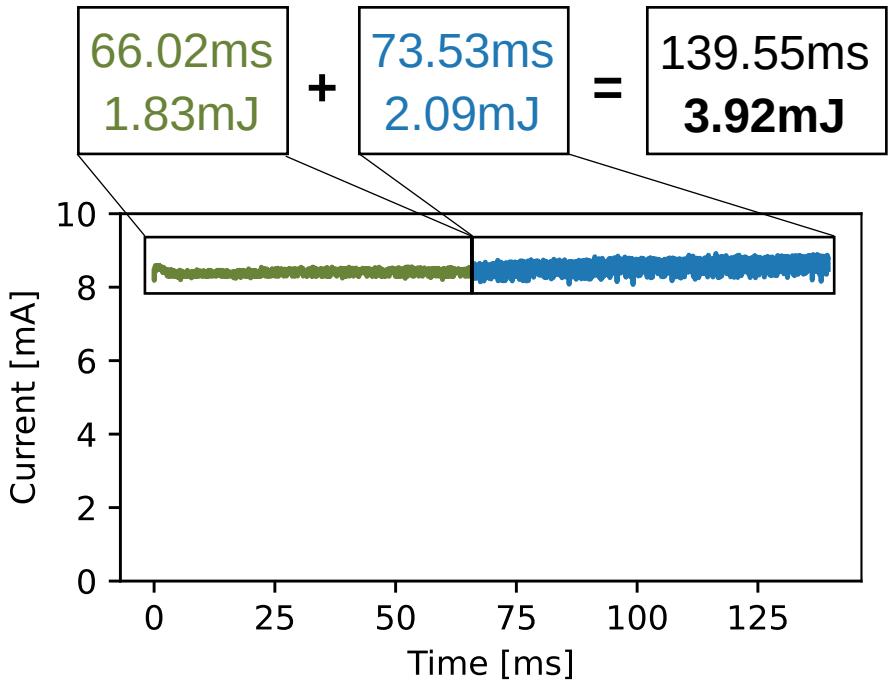


Motivation

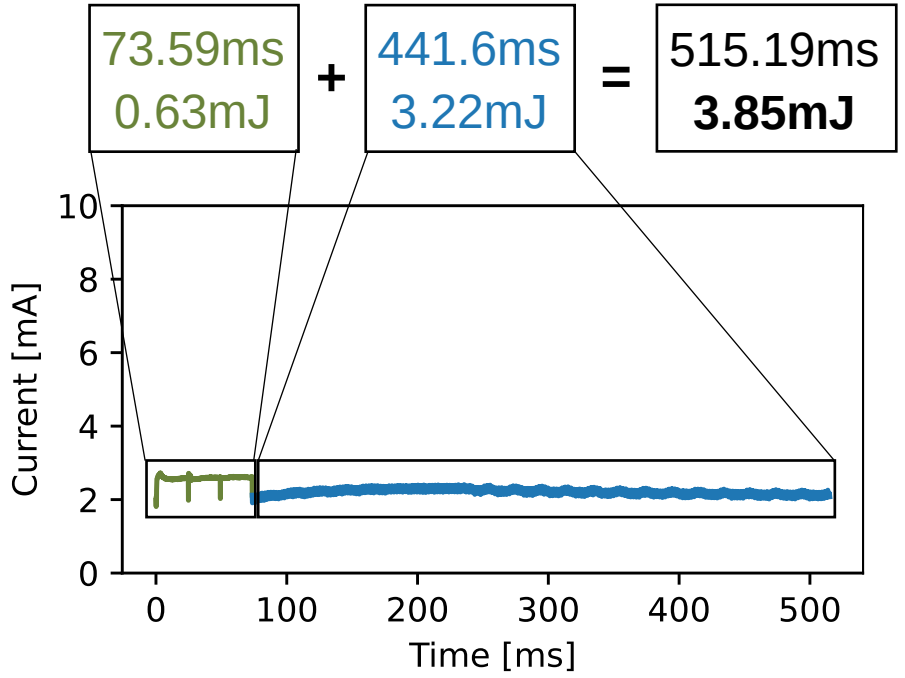
Beispielanwendung

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)

Strategie 1: *race-to-idle (**48MHz**)*



Strategie 2: *low-frequency (**8MHz**)*



Motivation

Beispielanwendung

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)

Strategie 1: *race-to-idle (**48MHz**)*

66.02ms
1.83mJ
+
73.53ms
2.09mJ
=
139.55ms
3.92mJ

Strategie 2: *low-frequency (**8MHz**)*

73.59ms
0.63mJ
+
441.6ms
3.22mJ
=
515.19ms
3.85mJ

Geht das nicht besser?

Motivation

Beispielanwendung

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)

Strategie 1: *race-to-idle (**48MHz**)*

66.02ms
1.83mJ
+
73.53ms
2.09mJ
=
139.55ms
3.92mJ

Strategie 2: *low-frequency (**8MHz**)*

73.59ms
0.63mJ
+
441.6ms
3.22mJ
=
515.19ms
3.85mJ

Geht das nicht besser?

→ Dynamic Frequency Scaling (DFS)

Gliederung

1 Motivation

2 Clock-Trees

Wie funktioniert der Wechsel der Taktfrequenz?

3 Dynamic Frequency Scaling (DFS)

3.1 *Power Clocks*

3.2 *ScaleClock*

Wann soll die Taktfrequenz geändert werden?

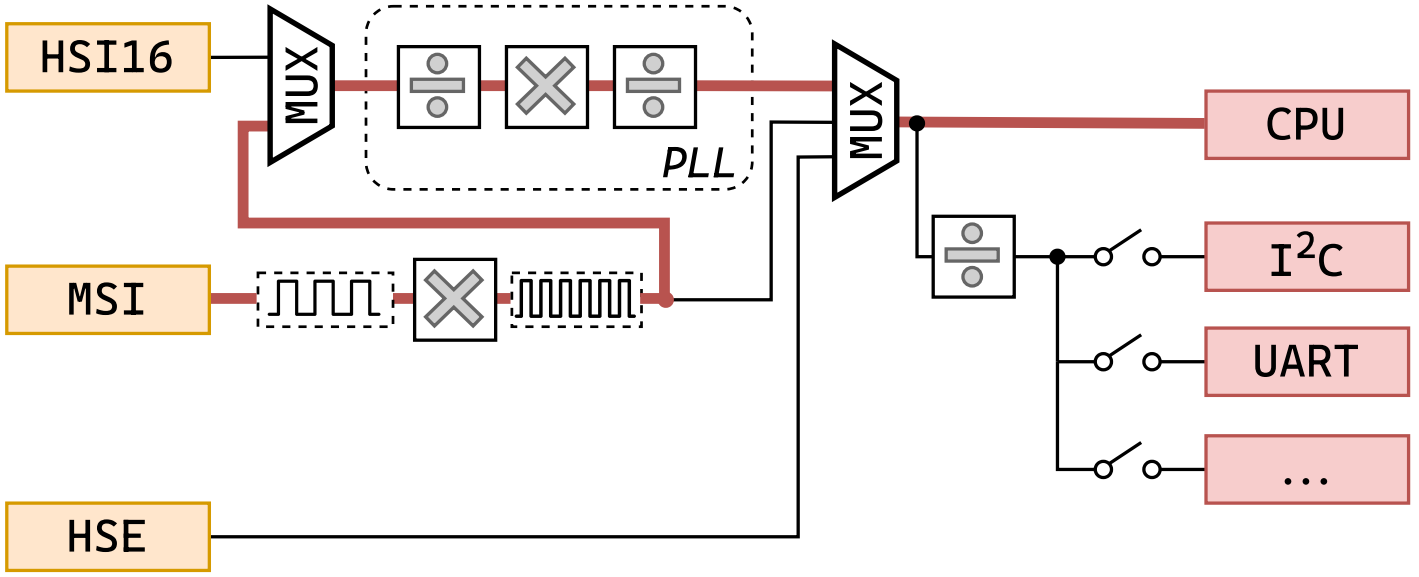
4 Zusammenfassung / Ausblick

Teil 1:
Wie funktioniert der Wechsel der Taktfrequenz?

Clock-Trees

Software-Unterstützung

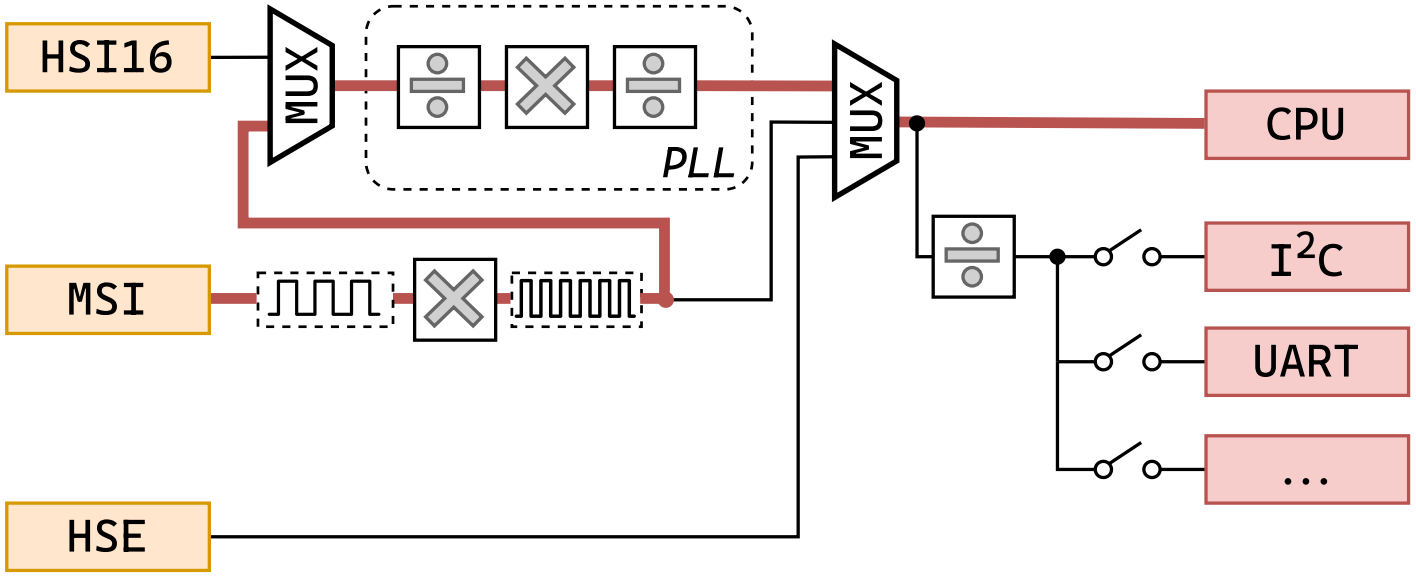
- Aktuell keine/wenig Unterstützung in IoT-Betriebssystemen für Clock-Konfiguration
  - Problem: Komplexe *Clock-Trees* je nach Hardware



Clock-Trees

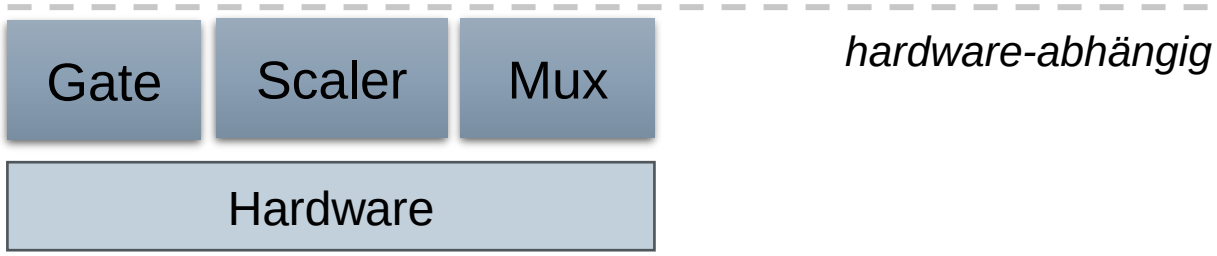
Software-Unterstützung

- Aktuell keine/wenig Unterstützung in IoT-Betriebssystemen für Clock-Konfiguration
  - Problem: Komplexe *Clock-Trees* je nach Hardware
  - Idee von ***ScaleClock***: Basisblöcke identifizieren und in Software abbilden



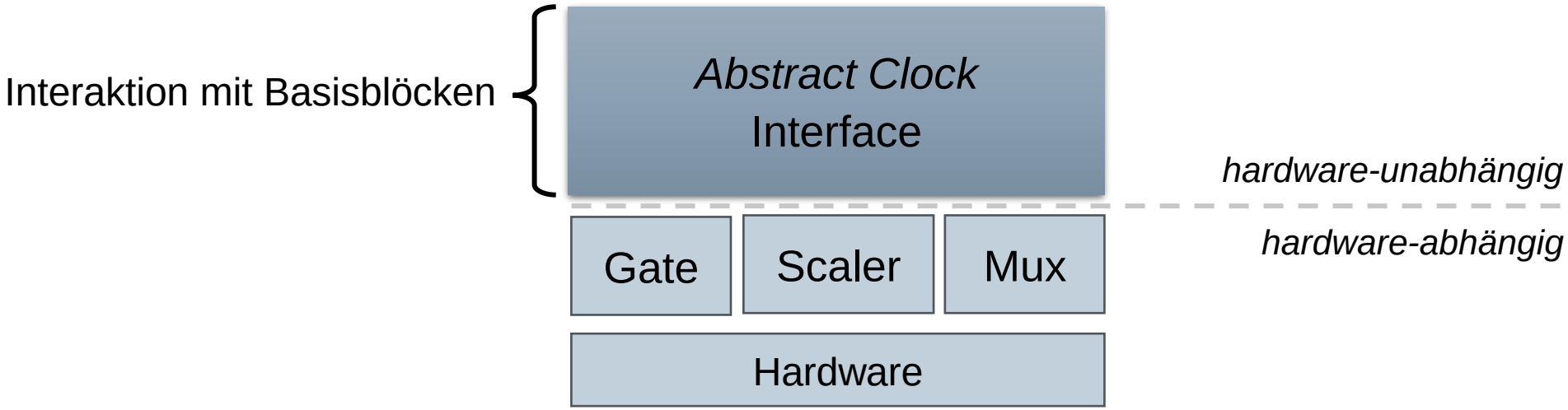
Clock-Trees

ScaleClock Architektur



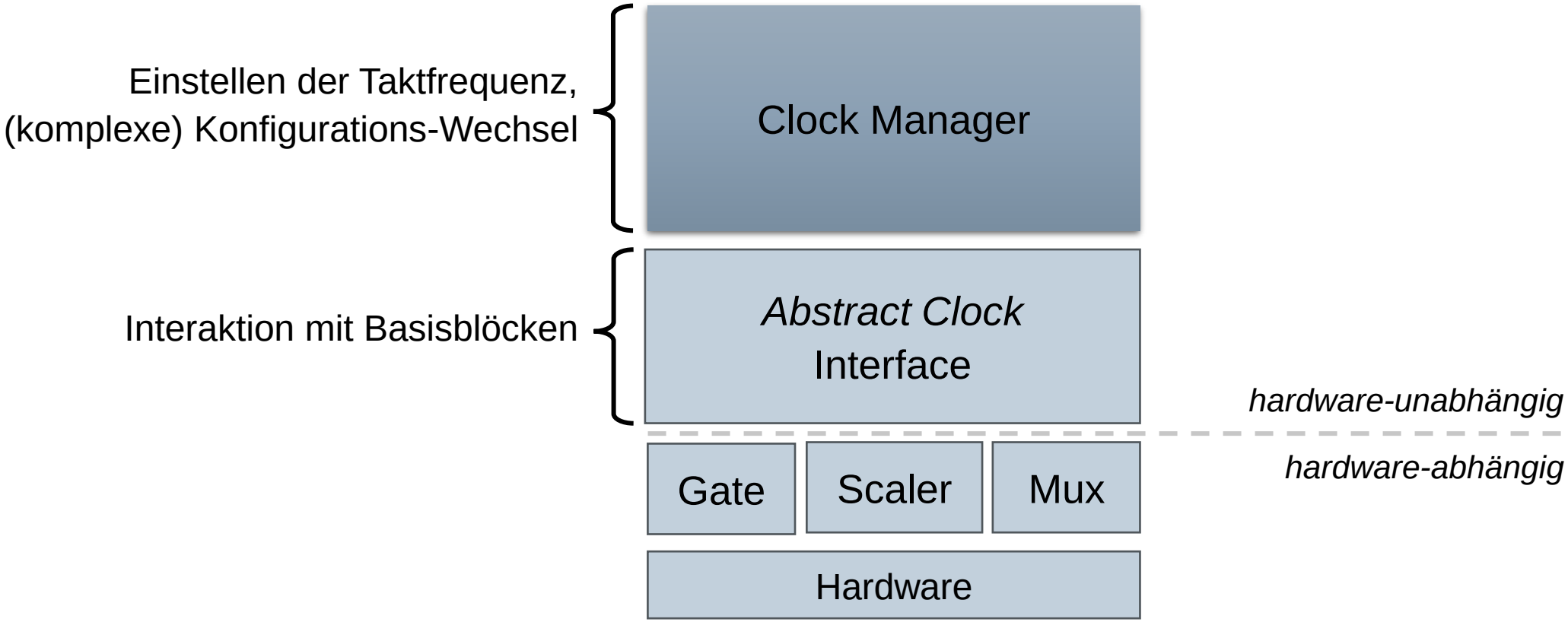
Clock-Trees

ScaleClock Architektur



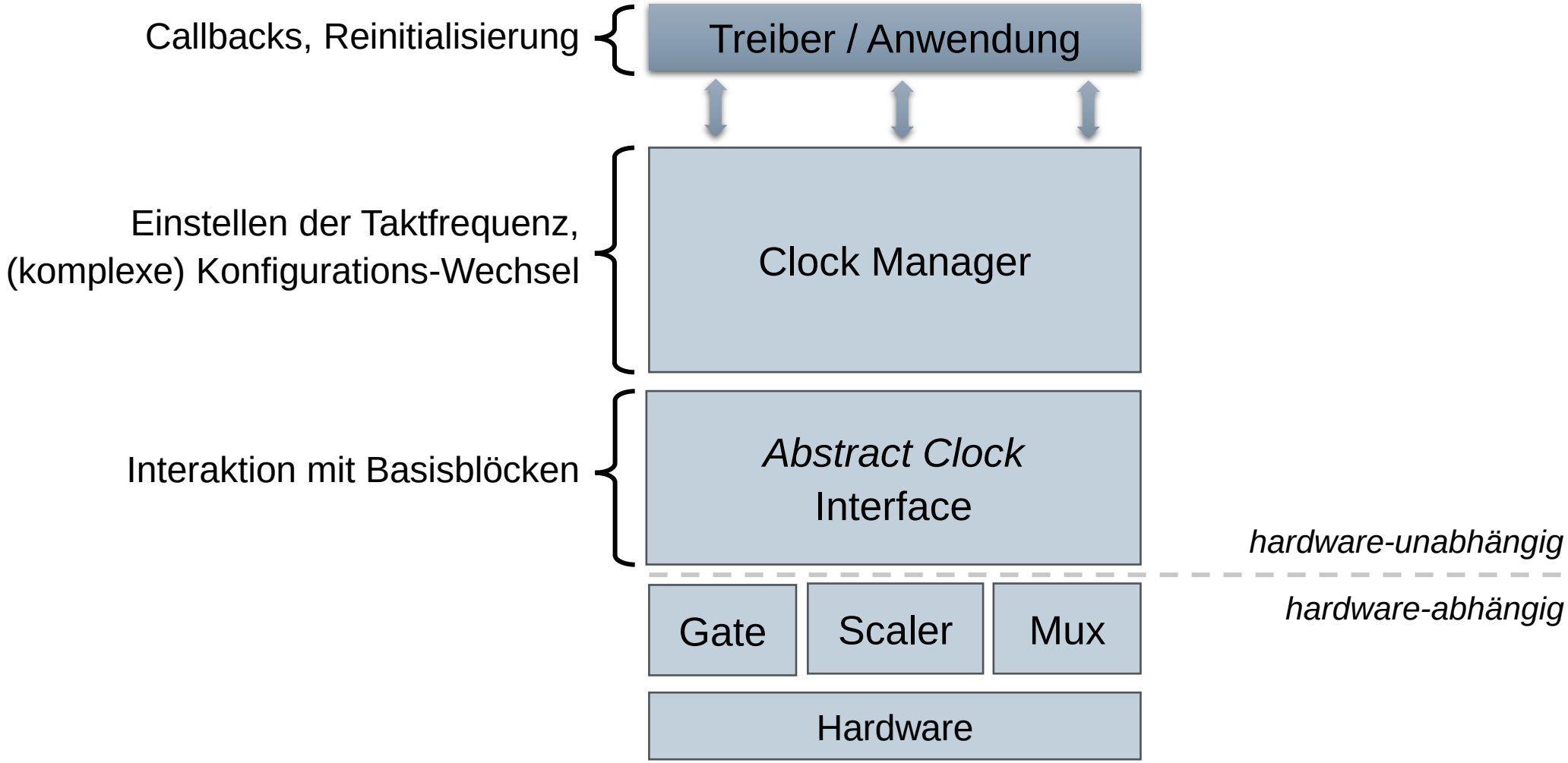
Clock-Trees

ScaleClock Architektur



Clock-Trees

ScaleClock Architektur



Teil 2:
Wann soll die Taktfrequenz geändert werden?

Dynamic Frequency Scaling (DFS)

Frequenzanpassung

- **Haupterkenntnis**
  - CPU-lastige Operationen energieeffizienter bei hoher Taktfrequenz
  - I/O-lastige Operationen energieeffizienter bei niedriger Taktfrequenz

→ Anpassen der Taktfrequenz je nach ausgeführten Operationen

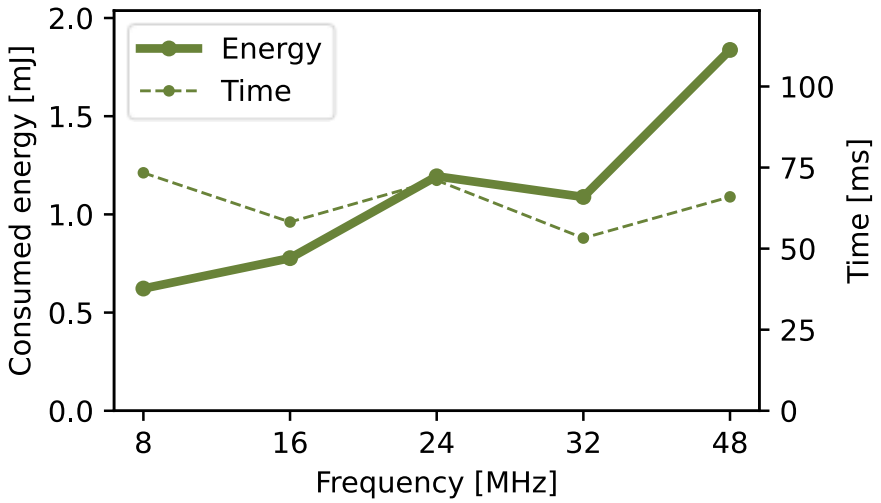
Dynamic Frequency Scaling (DFS)

Frequenzanpassung

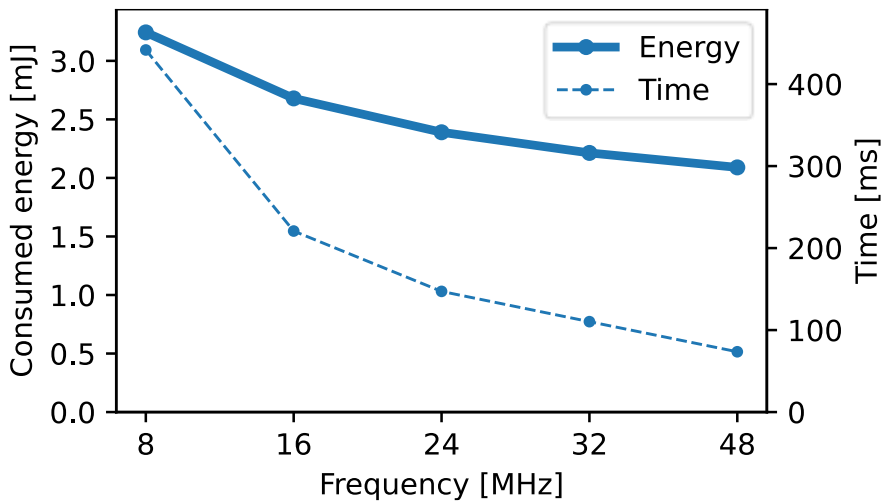
- **Haupterkenntnis**
  - CPU-lastige Operationen energieeffizienter bei hoher Taktfrequenz
  - I/O-lastige Operationen energieeffizienter bei niedriger Taktfrequenz

→ Anpassen der Taktfrequenz je nach ausgeführten Operationen

Daten von Sensor lesen



Daten verarbeiten



Dynamic Frequency Scaling (DFS)

Power Clocks

Holly Chiang, Hudson Ayers, Daniel Giffin, Amit Levy, and Philip Levis. 2021.
Power Clocks: Dynamic Multi-Clock Management for Embedded Systems.
In Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks. 139–150.

- Dynamischer Frequenzwechsel als Betriebssystemaufgabe
- Implementiert für das IoT-Betriebssystem *Tock*

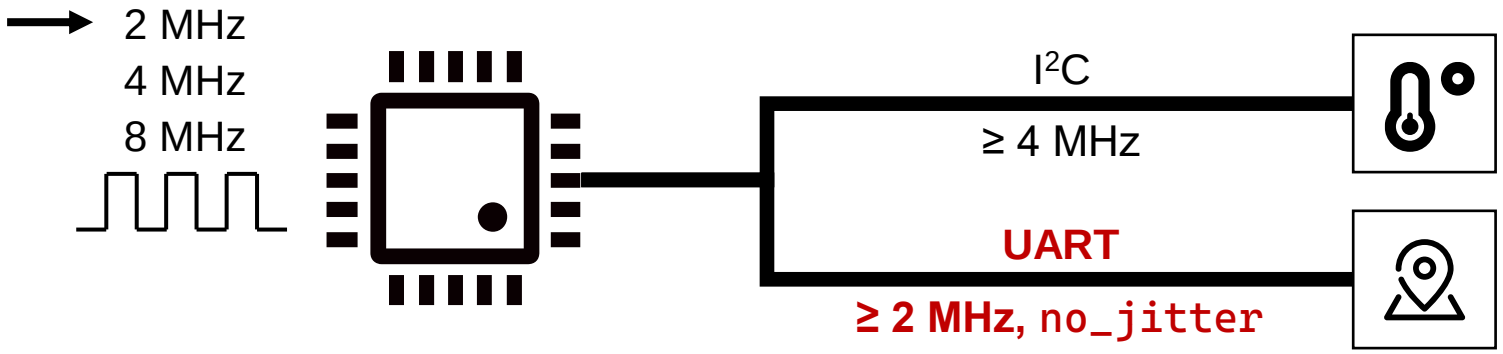
- **Idee**
  - mind. 1 I/O-Operation in Ausführung → wähle niedrigste **mögliche** Frequenz

    → Treiber rufen `make_clock_request`-Methode auf (Parameter: Mindesttaktfrequenz, `no_jitter`)

    → Betriebssystem ruft Callback-Funktion nach Wechsel der Taktfrequenz auf
  - keine I/O-Operation in Ausführung → wähle höchste Frequenz

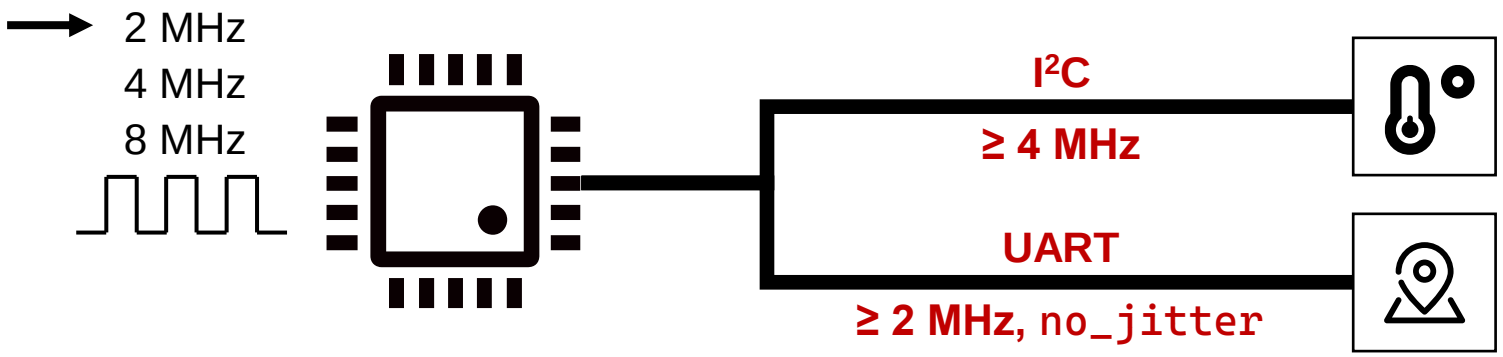
Dynamic Frequency Scaling (DFS)

Power Clocks



Dynamic Frequency Scaling (DFS)

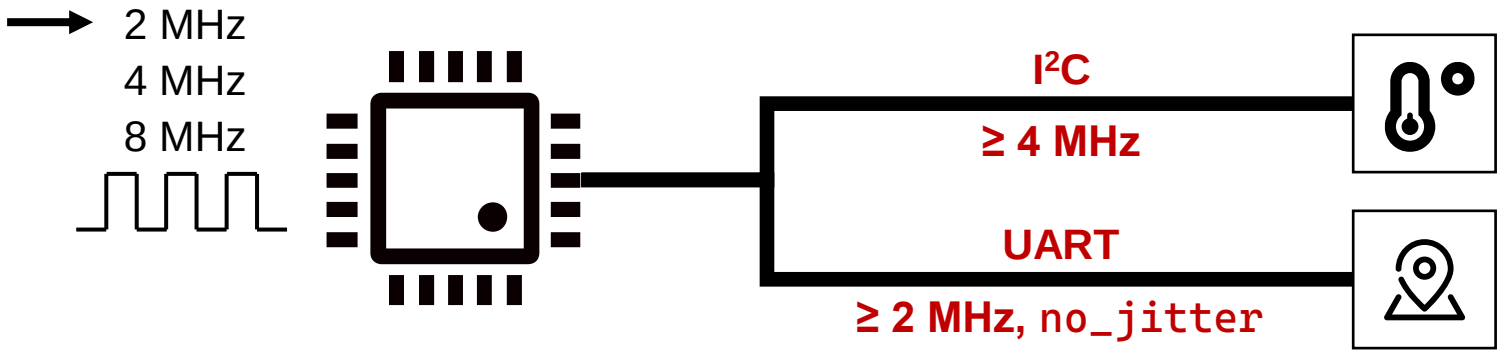
Power Clocks



- **Problem 1**: I²C muss warten bis UART-Übertragung abgeschlossen ist

Dynamic Frequency Scaling (DFS)

Power Clocks

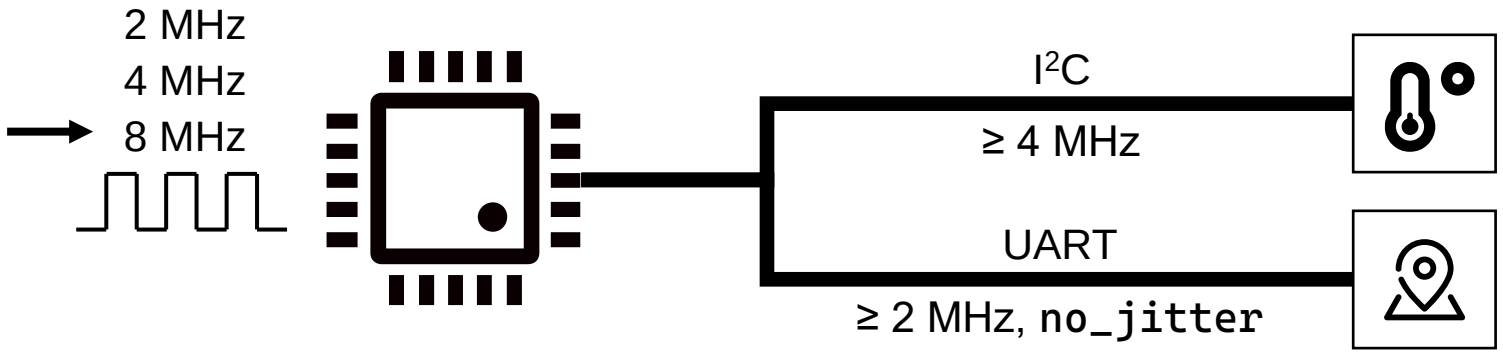


- **Problem 1**: I²C muss warten bis UART-Übertragung abgeschlossen ist

  → Verzögere Frequenzauswahl: Warte bis alle Threads `yield()` aufrufen

Dynamic Frequency Scaling (DFS)

Power Clocks



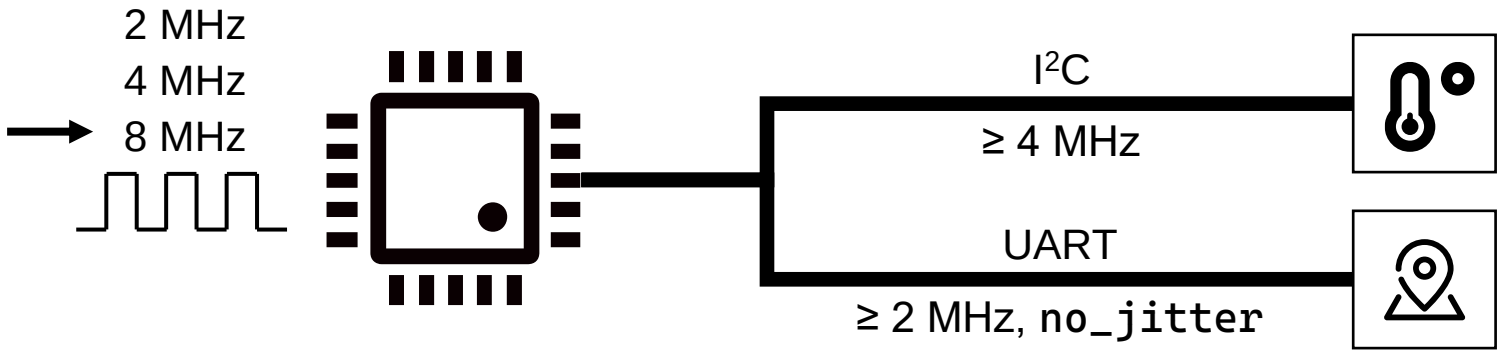
- **Problem 1**: I²C muss warten bis UART-Übertragung abgeschlossen ist

  → Verzögere Frequenzauswahl: Warte bis alle Threads `yield()` aufrufen

- **Problem 2**: *Thrashing* (Schneller Wechsel zwischen Taktfrequenzen)

Dynamic Frequency Scaling (DFS)

Power Clocks



- **Problem 1**: I²C muss warten bis UART-Übertragung abgeschlossen ist

  → Verzögere Frequenzauswahl: Warte bis alle Threads `yield()` aufrufen

- **Problem 2**: *Thrashing* (Schneller Wechsel zwischen Taktfrequenzen)

  → Verzögere Frequenzauswahl: Warte bei Wechsel auf hohe Taktfrequenz 10ms

Dynamic Frequency Scaling (DFS)

Power Clocks

- **Vorteile**
  - \+ Anwendungsentwickler muss keinen zusätzlichen Code schreiben
  - \+ Funktioniert in typischen IoT-Anwendungsfällen

- **Nachteile**
  - \- Schlecht, wenn Peripherie dauerhaft den Frequenzwechsel blockiert
  - \- Sollte nicht verwendet werden, wenn nur CPU-lastige Operationen ausgeführt werden
  - \- Im Papier: Ändern der Taktfrequenz nicht näher beschrieben

Dynamic Frequency Scaling (DFS)

ScaleClock

Michel Rottleuthner, Thomas C Schmidt, and Matthias Wahlisch. 2023. 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. 168–179.

- auch hier: Frequenzwechsel als Betriebssystemaufgabe
- implementiert für das IoT-Betriebssystem *RIOT* (→ **STM32**)

- **Idee:** Frequenzwechsel anhand einer Metrik pro Thread
- Performance Utilization (PU): Wie gut skaliert die Ausführungszeit eines Threads mit steigender Frequenz?
  - $PU = \frac{t_{busy}(F_1)}{t_{busy}(F_2)} \cdot \frac{F_1}{F_2}, \quad F_1 < F_2$

Dynamic Frequency Scaling (DFS)

ScaleClock

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)

I/O-Thread

8MHz **48MHz**

$$PU_{IO} = \frac{73.59ms}{66.02ms} \cdot \frac{8\ MHz}{48\ MHz} = \mathbf{0.186}$$

→ schlechte Skalierung

CPU-Thread

8MHz **48MHz**

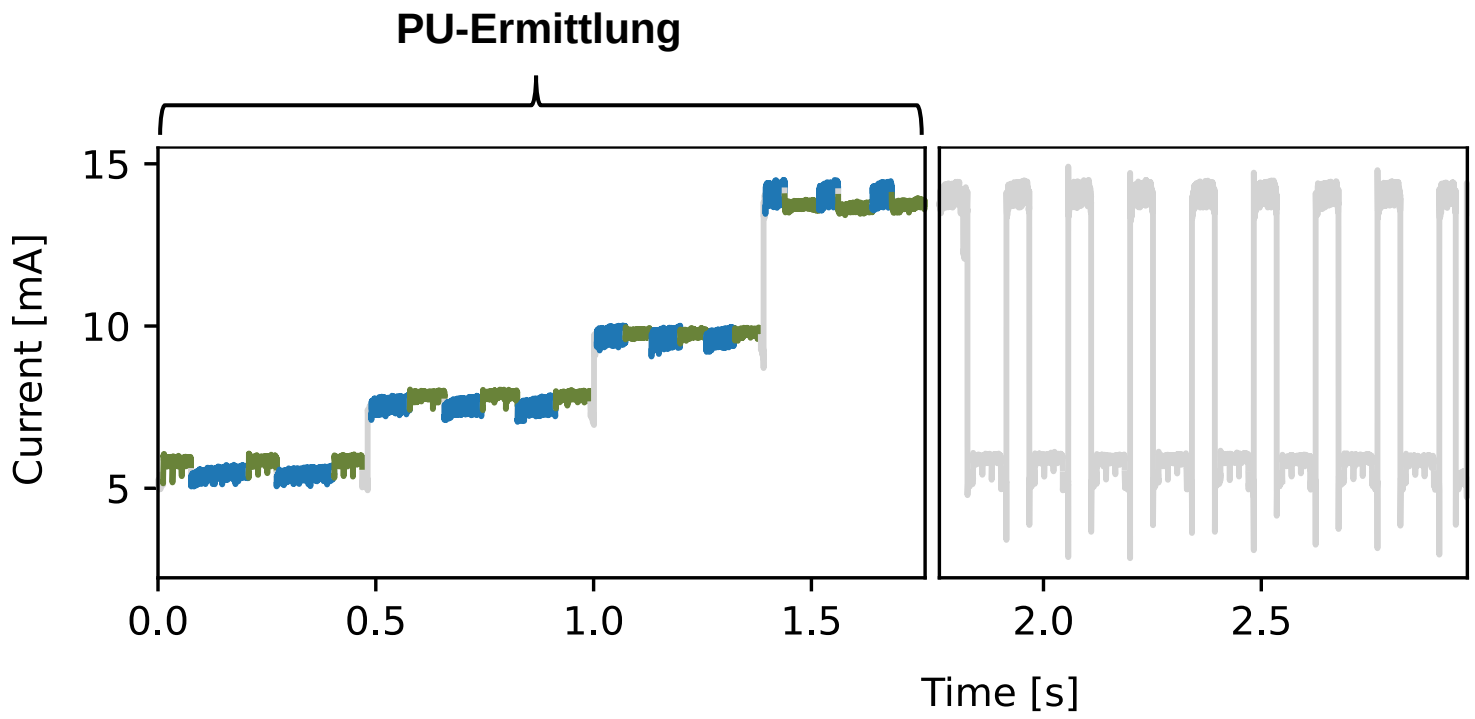
$$PU_{CPU} = \frac{441.6ms}{73.53ms} \cdot \frac{8\ MHz}{48\ MHz} = \mathbf{1.00}$$

→ optimale Skalierung

Dynamic Frequency Scaling (DFS)

ScaleClock

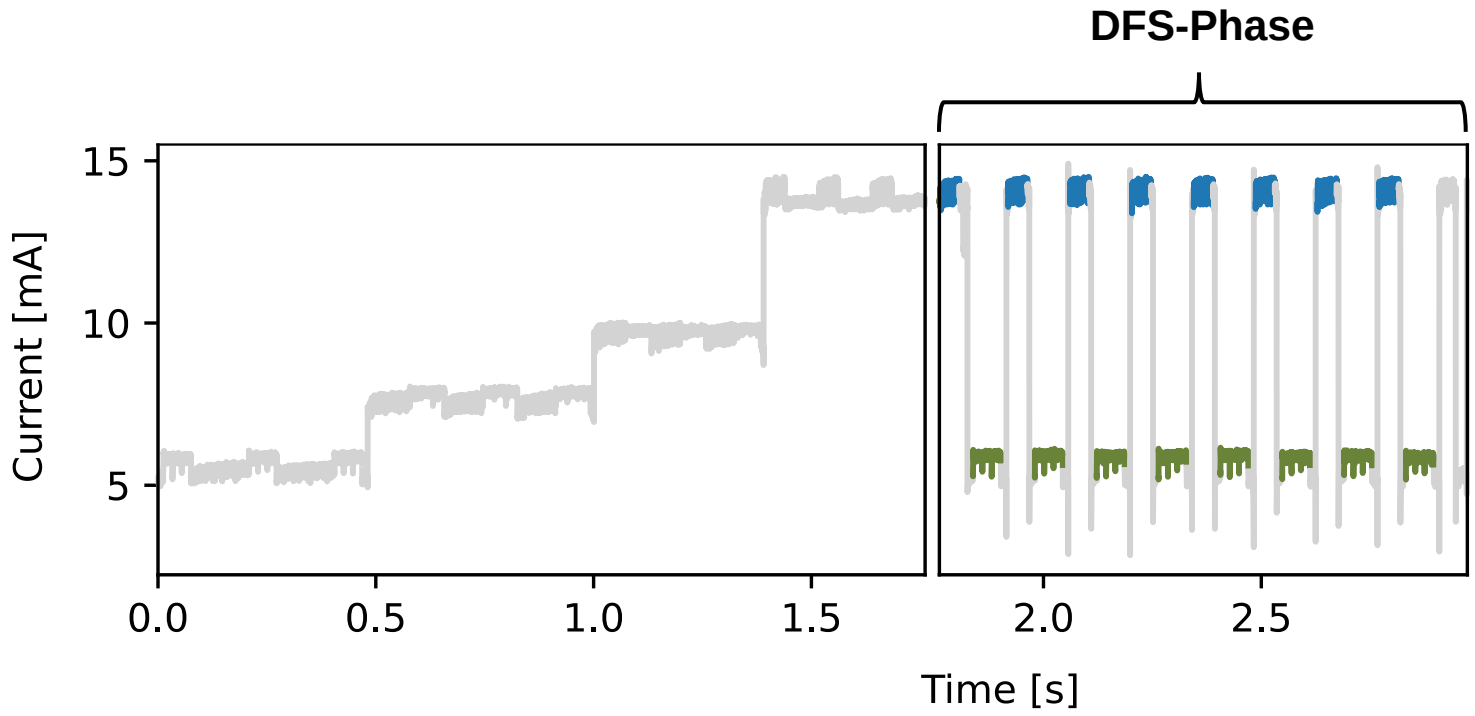
Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)



Dynamic Frequency Scaling (DFS)

ScaleClock

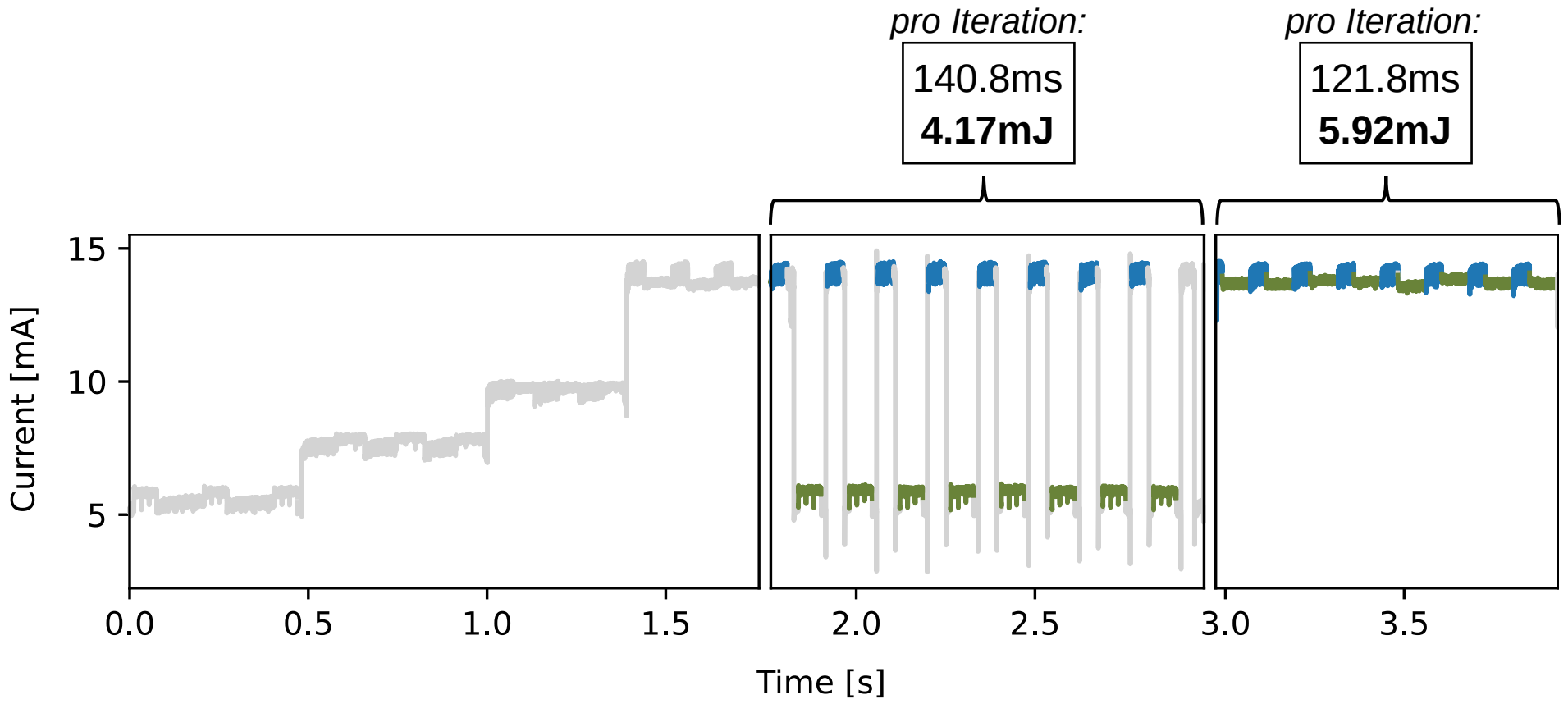
Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)



Dynamic Frequency Scaling (DFS)

ScaleClock

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)



Dynamic Frequency Scaling (DFS)

ScaleClock

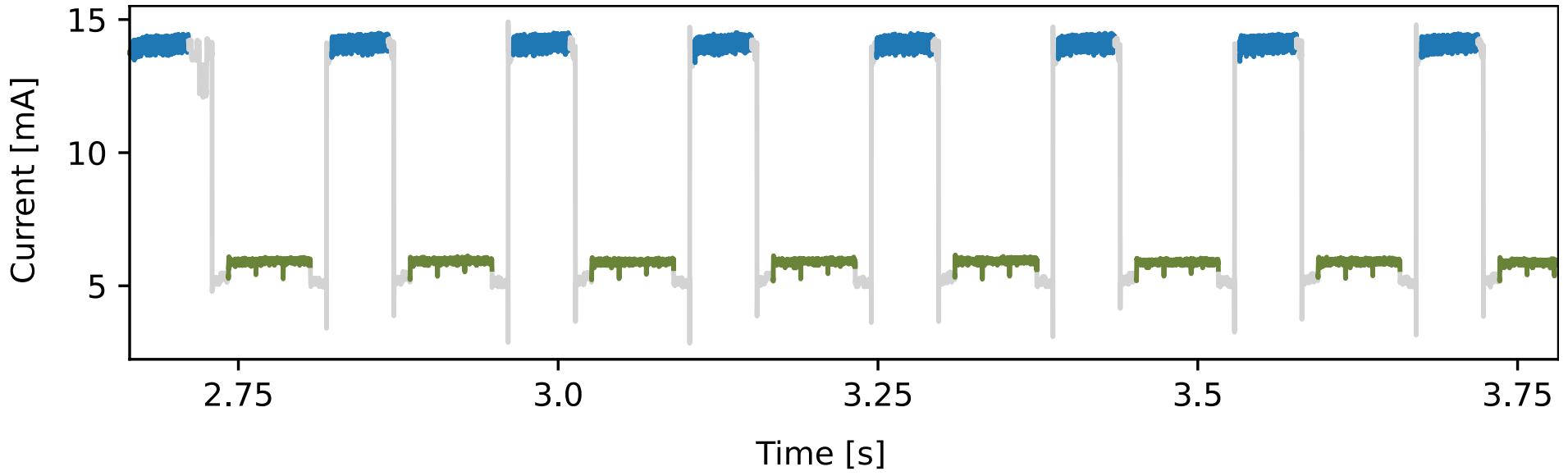
- **Vorteile**
  - \+ Anwendungsentwickler muss keinen zusätzlichen Code schreiben
  - \+ Sinnvolle Abbildung der Clock-Trees auf Betriebssystemebene

- **Nachteile**
  - \- Berechnung der Performance Utilization kostet Zeit
  - \- Treiberabhängigkeiten nicht berücksichtigt
  - \- Threads mit gemischten Operationen (CPU- und I/O-lastig) nicht berücksichtigt

Dynamic Frequency Scaling (DFS)

Probleme durch den Clock-Wechsel

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)



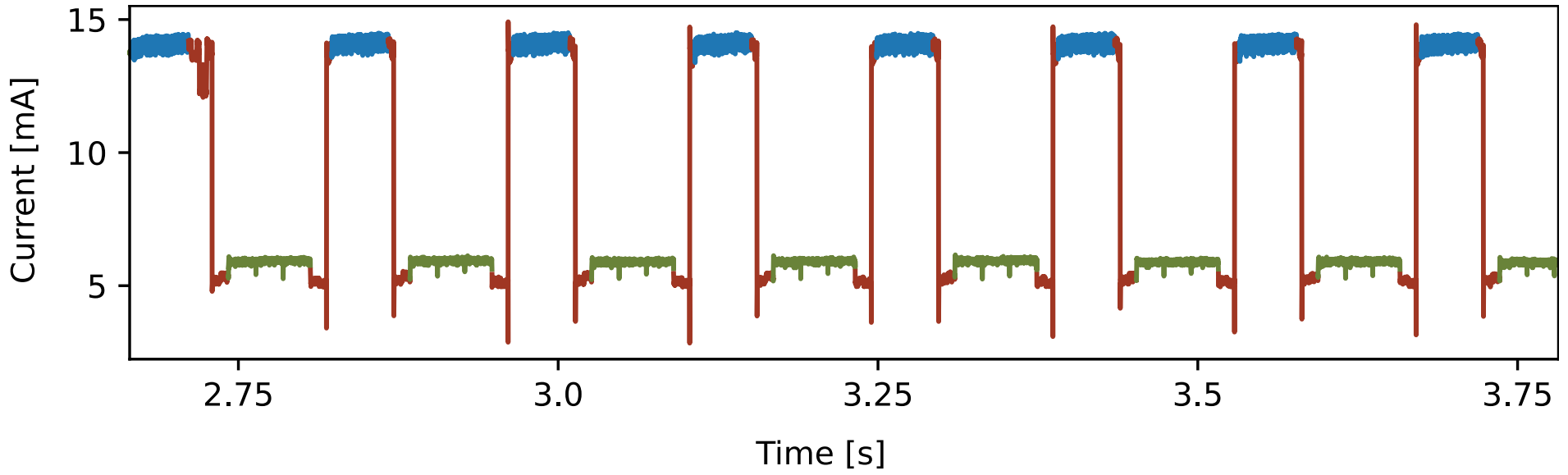
Dynamic Frequency Scaling (DFS)

Probleme durch den Clock-Wechsel

Beispiel: **Lese Daten von Sensor *(SPI)*** → **Verarbeite die Daten** (und speichere sie)

Problem: Frequenzänderung kostet (signifikant) Zeit

- hier: ca. 17ms **pro Frequenzwechsel**
- Grund: Treiber-Reinitialisierung (UART: 7.8ms / Timer: 4.2ms – 12.6ms)



Zusammenfassung

Vergleich

- ***Power Clocks*:**
  - \+ Frequenzentscheidung: Globale Sicht
  - \+ Berücksichtigt Treibervoraussetzungen
  - \- Frequenzwechsel von den Autoren nicht näher beschrieben

- ***ScaleClock*:**
  - \+ Abstraktion plattform-spezifischer Clock-Trees
  - \- Frequenzentscheidung: pro Thread
  - \- Berücksichtigt keine Treibervoraussetzungen

Zusammenfassung

Ausblick

- DFS als vielversprechende Methode, um Energie zu sparen

- *ScaleClocks* und *Power Clocks* mit unterschiedlichem Fokus
  - Kombination beider Ansätze in Zukunft?

- Herausforderungen
  - Zeitpunkt des Frequenzwechsels
  - Treiberoptimierungen

Vielen Dank
für Eure Aufmerksamkeit!