Dynamic Clock Reconfiguration for the Constrained IoT and its Application to Energy-efficient Networking

Michel Rottleuthner - HAW Hamburg
Thomas C. Schmidt - HAW Hamburg
Matthias Wählisch - Freie Universität Berlin
Motivation

- NOx & VOC values high
- Structural instability
- Escalator must be greased
- Turn on Ventilation
- Crossover is stuck
- Door doesn’t open
- Stop Escalator
- Light bulb broken
- High Blood Pressure
- Replace Filter!
- You made 10,000 steps!
- Motion detected
- Trash Bin Full
- Heart rate high
- Low glucose Level
- Train operator asleep
- Sync Time
- NOx & VOC values high
- Needs maintenance
- Train operator asleep
- Motion detected
- Heart rate high
- Low glucose Level
- Replace Filter!
- You made 10,000 steps!
IoT Firmware Development

- Agile development
- Faster time to market
- Better interoperability
- Improved software support and updates (better security?)

Unified tooling

ARM mbed

RIOT

RTOS

Zephyr

Contiki

NEXT GENERATION
IoT Firmware Development

- Agile development
- Faster time to market
- Better interoperability
- Improved software support and updates (better security ?)

- Limited access to very hardware specific features
  → Needed for low-power optimizations
  → Must be added to the HAL to employ it cross-platform
Outline

• Motivation
  − A Catch with Modern IoT Firmware Development

• Energy in the Constrained IoT

• Dynamic Clock Configuration

• Our Approach: ScaleClock
  − Evaluation
    − Application: Energy-efficient Networking

• Conclusion & Future Work

• Q&A
Dynamic Clock Reconfiguration for the Constrained IoT and its Application to Energy-efficient Networking

In the Constrained IoT
IoT Power Management

- Duty Cycling
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep
IoT Power Management

- Duty Cycling
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep
IoT Power Management

- **Duty Cycling**
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep

- **Power Gating**
  - Disable unneeded peripherals
  - External and MCU-internal peripherals
IoT Power Management

- Duty Cycling
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep

- Power Gating
  - Disable unneeded peripherals
  - External and MCU-internal peripherals
IoT Power Management

- **Duty Cycling**
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep

- **Power Gating**
  - Disable unneeded peripherals
  - External and MCU-internal peripherals
IoT Power Management

• Duty Cycling
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep

• Power Gating
  - Disable unneeded peripherals
  - External and MCU-internal peripherals
IoT Power Management

- **Duty Cycling**
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep

- **Power Gating**
  - Disable unneeded peripherals
  - External and MCU-internal peripherals
IoT Power Management

- **Duty Cycling**
  - Put system to sleep on idle
  - Wakeup via timers or interrupts
  - No processing during sleep

- **Power Gating**
  - Disable unneeded peripherals
  - External and MCU-internal peripherals

- **Dynamic Voltage and Frequency Scaling**
  - Fine grained performance control
  - Low-power processing
IoT Power Management

- Duty Cycling
  - Set CPU to sleep on Idle
  - Wakeup on event
  - No processing during sleep

- Power Gating
  - Turn off unused subsystems

- Dynamic Voltage and Frequency Scaling (DVFS)
  - Precise performance control
  - Requires access to the clock configuration subsystem

Can be used complementary!

Not part of unified tooling
IoT Power Management

- Duty Cycling
  - Set CPU to sleep on Idle
- Power Gating
  - Turn off unused subsystems
- Dynamic Voltage and Frequency Scaling (DVFS)
  - Precise performance control
  - Requires access to the clock configuration subsystem

How to provide dynamic clock configuration with unified tooling?

- Power Gating
- Dynamic Voltage and Frequency Scaling (DVFS)
- Not part of unified tooling
Dynamic Clock Reconfiguration for the Constrained IoT and its Application to Energy-efficient Networking

Problem Overview
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- **Capacitance**
- **Switching activity**
- **Voltage**
- **Frequency**

[6], [7]
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- Capacitance
- Switching activity
- Voltage
- Frequency

“Objective Function”

\[ \text{Objective Function} \]
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

"Objective Function"

Capacitance

Switching activity

Voltage

Frequency

"Hardware-specific"
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- **Capacitance**
- **Switching activity**
- **Voltage**
- **Frequency**

- **Power**
- **Objective Function**

- ** Hardware-specific**
- **Task-specific**

[6], [7]
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- Power
- Capacitance
- Switching activity
- Voltage
- Frequency

"Objective Function"
"Hardware-specific"
"Task-specific"
"Adjustable"

[6], [7]
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

\[ W = P \cdot t \]

"Objective Function!"
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- range(frequency) >> range(voltage) and voltage \( \propto \) frequency

Energy

\[ W = P \cdot t \]

1 \( \rightarrow \) 1

1 \( \rightarrow \) 0.8
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- range(frequency) >> range(voltage) and voltage \( \ll \) frequency
- Operations not always scale well with frequency (memory and peripheral access, radio communication, ...)
  - Scalability bottlenecks waste energy

\[ W = P \cdot t \]

1 \( \rightarrow \) 0.47

1 \( \rightarrow \) 0.8

"Objective Function!"
Dynamic Voltage and Frequency Scaling

\[ P = C \cdot \alpha \cdot V^2 \cdot f \]

- range(frequency) >> range(voltage) and voltage \( \notin \) frequency

How to detect those bottlenecks to save energy?

\[ W = P \cdot t \]

1 \( \rightarrow \) 0.47

1 \( \rightarrow \) 0.8

"Objective Function!"
Clock Trees

(from STM32L476RG reference manual RM0351)
Dynamic Clock Reconfiguration for the Constrained IoT and its Application to Energy-efficient Networking

Clock Trees

(from STM32L476RG reference manual RM0351)
Clock Trees

(from STM32L476RG reference manual RM0351)
Clock Trees

STM32L476RG

EFM32PG12B
Dynamic Clock Reconfiguration for the Constrained IoT
and its Application to Energy-efficient Networking

Our Approach: ScaleClock
Clock Tree Abstraction in ScaleClock
Clock Tree Abstraction in ScaleClock
ScaleClock Architecture

Application

Utilization Monitor

Monitor

OS / Scheduler

Transition Manager

Generic Module

Query & Notify

Clock Configurator

Unified Interface

Abstract Clock

Static Model & Configuration

Properties, Constraints, Flags

Gate

Mux

Scaler

... Hardware

Core Voltage Module

Callback Interaction

Voltage & Frequency Range Constraints
ScaleClock Architecture

Application

Utilization Monitor

Monitor

Query & Notify

Generic Module

OS / Scheduler

Transition Manager

Clock Configurator

Abstract Clock

Core Voltage Module

Callback Interaction

Unified Interface

Static Model & Configuration

Properties, Constraints, Flags

Gate

Mux

Scaler

...
ScaleClock Architecture

Application

Utilization Monitor

Monitor

Query & Notify

OS / Scheduler

Transition Manager

Clock Configurator

Abstract Clock

Callback Interaction

Core Voltage Module

Generic Module

Unified Interface

HW-specific

Static Model & Configuration

Properties, Constraints, Flags

Gate

Mux

Scaler

... Hardware

Voltage & Frequency Range Constraints
How to provide dynamic clock configuration with unified tooling?
ScaleClock Architecture

How to detect bottlenecks to save energy?
Performance Utilization Metric

Load = \frac{t_{\text{busy}}}{t_{\text{busy}} + t_{\text{idle}}} \times \mathbf{X}

PU = \frac{t_{\text{busy}}}{t_{\text{busy}}} \left( \frac{F_1}{F_2} \right) \cdot \frac{F_1}{F_2} \mathbf{✓}

How to detect bottlenecks to save energy?
Dynamic Clock Reconfiguration for the Constrained IoT and its Application to Energy-efficient Networking

Experiments & Evaluation Results
Power Reduction

![Graphs showing power reduction for different operations (add, divide, multiply) with two different datasets (nucleo-l476rg, slstk3402a) at varying core frequencies.](image-url)
Power Reduction

→ Static power offset and slope differ between platforms
→ Instruction types impact consumption considerably
Topoogy Impact

(same task at different clock topology and frequency; compared to default)
→ Explicit topology control is needed due to its significant impact
→ Scaling closer to the source is preferrable
Task Scalability

Task A

- Low Frequency
- High Frequency

Task B

- Low Frequency
- High Frequency

→ nicely scalable

→ bottle-necked
Task Scalability

Task A

Low Frequency

High Frequency

→ nicely scalable

Task B

Low Frequency

High Frequency

→ bottle-necked
Task-specific Performance Utilization

![Graph showing performance utilization vs frequency for two different devices: `nucleo-1476rg` and `slstk3402a`. The graph indicates that `nucleo-1476rg` has better performance utilization for bottle-necked tasks compared to nicely scalable tasks, while `slstk3402a` shows the opposite trend.](image-url)
Task-specific Performance Utilization

\[ PU = \frac{t_{\text{busy}}(F_1)}{t_{\text{busy}}(F_2)} \cdot \frac{F_1}{F_2} \]

→ Online assessed PU metric tracks energy-optimal frequency
PU-based DFS Control of Multithreaded Applications

Two tasks, one with low PU value (acquisition) and one with high PU value (processing)
PU-based DFS Control of Multithreaded Applications

Two tasks, one with low PU value (acquisition) and one with high PU value (processing)

→ Performance Utilization metric serves as viable frequency control input
Dynamic Clock Reconfiguration for the Constrained IoT and its Application to Energy-efficient Networking

Case Study Results
Energy-efficient Networking

(Frequency Impact)

TX, 64 Bytes UDP

slow down clock

![Graph showing the relationship between core frequency and energy consumption/time for a 64 Bytes UDP transmission. The graph indicates a decrease in energy as the core frequency decreases, suggesting that slowing down the clock can result in energy savings.]
Energy-efficient Networking

(Frequency Impact)

TX, 64 Bytes UDP

Energy: -42%  Time: +1%

Energy: -82%  Time: +14%
Energy-efficient Networking

(Frequency Impact)

TX, 64 Bytes UDP

Energy: -42%  Time: +1%

RX, 64 Bytes UDP

Energy: -82%  Time: +14%

![Graphs showing energy and time impact across different core frequencies for TX and RX transmissions.](image-url)
Energy-efficient Networking
(Frequency Impact)

TX, 64 Bytes UDP

Energy: -42% Time: +1%

Energy: -82% Time: +14%

RX, 64 Bytes UDP

Energy: -40% Time: +2%

Energy: -69% Time: +44%
Energy-efficient Networking
(Payload Size Impact; Energy-optimized vs. default; Normalized to max payload)
Energy-efficient Networking

(Payload Size Impact; Energy-optimized vs. default; Normalized to max payload)

TX

Energy [Normalized]

Time [ms]

Payload Size [Byte]

Energy Reduction

0

0

20

40

60

80

100

120

140

160

180

0

2

4

6

8

10

12

14

16

18

TX Energy (default) Energy (optimized) Time (default) Time (optimized)
Energy-efficient Networking

(Payload Size Impact; Energy-optimized vs. default; Normalized to max payload)
Energy-efficient Networking

(Payload Size Impact; Energy-optimized vs. default; Normalized to max payload)
Energy-efficient Networking

(Payload Size Impact; Energy-optimized vs. default; Normalized to max payload)

→ Optimization beneficial for all UDP payload sizes
→ Temporal performance impact higher for RX
Conclusion & Future Work

What did we learn and what’s next?
Conclusion

- Generic clock configuration feasible for common IoT platforms
  - Enables self-optimization for energy-aware systems
Conclusion

- Generic clock configuration feasible for common IoT platforms
  - Enables self-optimization for energy-aware systems

- Frequency scaling is not enough
  - Voltage and topology control offer significant benefits
  - Online PU-assessment for task specific performance
Conclusion

• Generic clock configuration feasible for common IoT platforms
  - Enables self-optimization for energy-aware systems

• Frequency scaling is not enough
  - Voltage and topology control offer significant benefits
  - Online PU-assessment for task specific performance

• For energy-efficient networking
  ... 40% energy can be saved without noticeable performance impact
  ... 80% energy can be saved in case of non-critical timing
Future Work

• More platforms & applications
Future Work

- More platforms & applications
- Parametric power model for the clock subsystem
  - Determine parameters automatically
Future Work

- More platforms & applications
- Parametric power model for the clock subsystem
  - Determine parameters automatically
- Optimize task characterization
Future Work

• More platforms & applications
• Parametric power model for the clock subsystem
  – Determine parameters automatically
• Optimize task characterization
• Integration of different control mechanisms
  – Threshold selection, PID, AI, ...
Questions & Discussion

ScaleClock Sources
https://github.com/inetrg/RIOT/tree/ScaleClock

Related Websites
Internet Technologies research group | https://inet.haw-hamburg.de/
RIOT OS | https://www.riot-os.org/

Contact
Michel Rottleuthner | michel.rottleuthner@haw-hamburg.de
Thomas C. Schmidt | t.schmidt@haw-hamburg.de
Matthias Wählisch | m.waehlisch@fu-berlin.de