# Hardware Analysis and Low-level API Design Towards a Timer Subsystem for RIOT-OS

Niels Gandraß
<Niels.Gandrass@haw-hamburg.de>
Hamburg University of Applied Sciences

## 1 LITERATURE REVIEW

Assessment of related work on:

(a) timer hardware
(b) implementation techniques
(c) software modules

## 2 TIMER-HARDWARE ANALYSIS

43 MCU families from 8 different manufacturers

Base properties, channels, CLKs, low-power modes, chaining, ...

## 3 RIOT-OS TIMER MODULE REVIEW

Common design aspects and potential enhancements.

## 4 DERIVING API REQUIREMENTS

Demands on generic a low-level timer-API for RIOT-OS.

feature exposure, portability, HW capabilities, state, config management, resources, ...

## 5 LOW-LEVEL TIMER API DESIGN

Technical draft of aspired API.

Platform-independent as long as possible. Separation of HW- and user-facing APIs.

## 6 EVALUATION & BENCHMARKING

Prototypical implementation on STM32 platform.

Micro-benchmarking of different design decisions.

## 7 IMPLEMENTATION & VERIFICATION

Provide support for various RIOT-OS compatible MCUs.

Backed by extensive HiL tests.

Scan to **download poster** and get additional information.

## – TIMER-HARDWARE ANALYSIS — 2

*Gaining detailed insight into target hardware, underpinning the API design.*

| | |
|---|---|
| **PLATFORM SELECTION** | Determination of all MCU device families currently supported by RIOT-OS. Acquisition of datasheets, reference manuals, application notes and others. |
| **ANALYSIS CRITERIA** | Definition of criteria and properties to at least be extracted from available information sources. Based on literature analysis and extended by us. |
| **INFORMATION EXTRACTION** | Extraction of timer data into mind-map structure. If documentation was unclear at some point, additional data sources (e.g. SDKs) were used. |
| **PLATFORM RESULTS** | Consolidation of results into *Timer Comparison Matrices* (TCMs) for every analyzed platform. |
| **INTER-PLATFORM FINDINGS** | Generation of inter-MCU-platform insights based on all TCMs. Includes basic properties as well as availability of advanced features. |

## – RIOT-OS TIMER MODULE REVIEW — 3

*Review of existing low-level timer implementations and their limitations.*

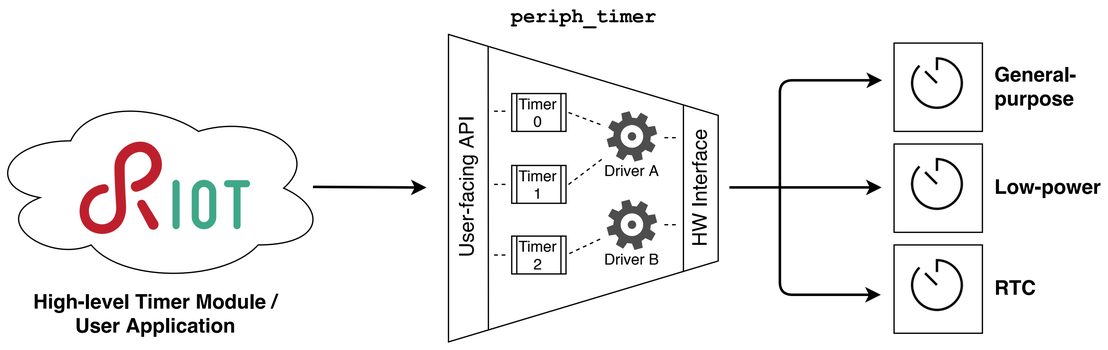
**LL-Modules**

```
periph/
 -timer ( )
 -rtc ( ), rtt ( )
 -pwm ( )
 -wdt ( )
```

- Reduced to minimal common functions
- Functionality often overlaps between modules
- Neither exposing all hardware timers nor all their basic features (e.g. compare channels)
- Support of advanced features is solely left to the application developer (e.g. low-power modes)

## – LOW-LEVEL TIMER API DESIGN — 5

*Timers shall be usable transparently interchangeable via the same unified API.*




**User-facing API**

Timer type abstracted functions exposed to user or high-level timer module.

**Hardware Interface**

Compact and reusable timer drivers, directly interfacing the various hardware peripherals.

## – EVALUATION — 6

*Prototypical implementation of the proposed API for STM32 MCUs.*

- Integration of currently unsupported timer types, all usable through a unified and MCU-independent API.
- Exposure of advanced features commonly found on mid- to high-end MCUs.
- Providing (runtime-) information on timer capabilities and properties.
- Introducing flexibility through driver based design. Timers are modeled as uniform objects in the form of standardized timer instance structs.
- Widening of runtime configuration possibilities (e.g. clock selection).
- Combining various hardware timers into a virtual instance (e.g. for chaining).