Work-in-Progress: Large-scale Timer Hardware Analysis for a Flexible Low-Level Timer-API Design

Niels Gandraß, Michel Rottleuthner, Thomas C. Schmidt
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 of all five existing APIs.

4 DERIVING API REQUIREMENTS

Demands upon a uniform 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 the novel API.

Two-layered design, separating hardware- and user-facing APIs. Platform-independence maintained whenever possible.

6 EVALUATION & BENCHMARKING

First prototypical implementation on the STM32 platform.

Validation and benchmarking on a representative set of MCUs.

7 IMPLEMENTATION & VERIFICATION

Provide support for various RIOT-OS compatible MCUs.

Backed by extensive HiL tests with CI integration.

Scan to **download paper** 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 each analyzed platform.
INTER-PLATFORM FINDINGS	Determination 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.

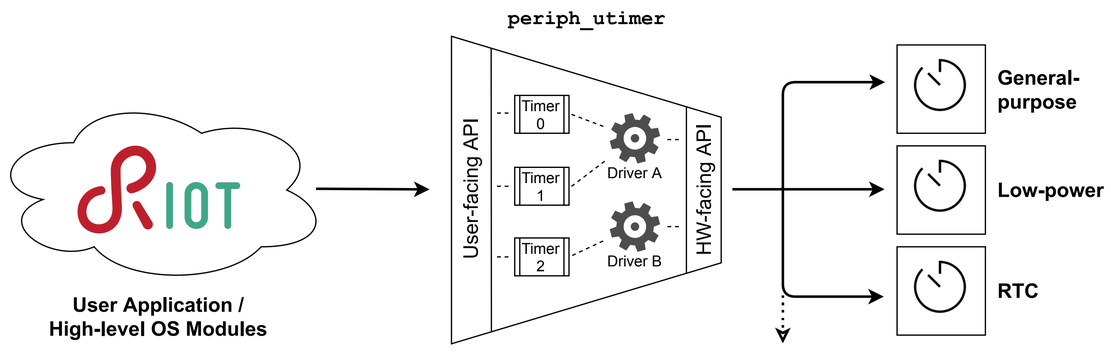
- Reduced to minimal common function set
- Functionality often overlaps between modules
- Neither exposing all hardware timers nor all their basic features (e.g. compare channels)
- Peripheral allocation conflicts between modules
- Application developers are often required to write driver code for features (e.g. low-power modes)
- Timer selection and configuration highly heterogeneous and requires changes to OS header files

RIOT-OS Modules

`periph/`
- `timer`
- `rtc`
- `rtt`
- `pwm`
- `wdt`

– LOW-LEVEL TIMER API DESIGN — 5

Streamlining existing APIs into a uniform interface, fostering flexible use of available timers and features while preserving application portability whenever possible.



User-facing API (uAPI)
Single set of timer type abstracted functions, exposed to the user application and high-level system modules

Hardware-facing API (hAPI)
Compact and reusable timer drivers for each used timer type, directly interfacing the various hardware peripherals.

- Timers interactively configured via KConfig
- Static properties and run-time status information provided to application and OS modules
- Similar functions are bundled within the hAPI
- Separate handling of CMP and OVF interrupts
- Run-time (re-)configuration of clock sources
- Virtual drivers allow representation of chained timers as atomic instances, extending existing driver code