A RIOT Port for the Single-Chip Mote

Alexandre Abadie
alexandre.abadie@inria.fr
Inria
Paris, France

Koen Zandberg
koen.zandberg@fu-berlin.de
FU Berlin
Berlin, Germany

Thomas Watteyne
thomas.watteyne@inria.fr
Inria
Paris, France

Filip Maksimovic
filip.maksimovic@inria.fr
Inria
Paris, France

ABSTRACT

This paper presents a demonstration of the first port of a real-time operating system, RIOT, for the single-chip mote (SCuM). RIOT provides SCuM all its features to develop multithreaded embedded applications for the Internet of Things. This demonstration showcases RIOT real-time capabilities using threads, embedded AI to detect hand-written digits in an image, and some cryptographic primitives, all running on SCuM.

CCS CONCEPTS

• **Computer systems organization** → **Embedded software**; **System on a chip**; **Firmware**.

KEYWORDS

SCuM, RIOT, microcontroller, RTOS,

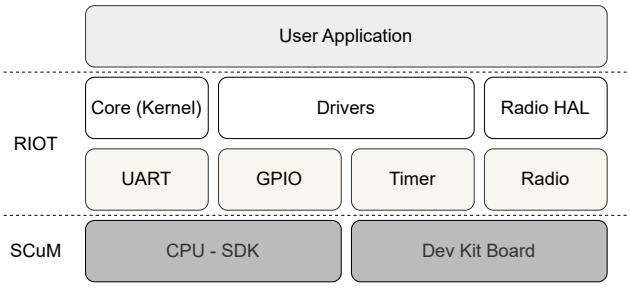

Figure 1: Architecture of the RIOT port on SCuM. The port consists in adapting the RIOT build system and implementing the drivers for the peripherals provided by SCuM.

1 INTRODUCTION

The development of Internet of Things (IoT) applications requiring increasing complexity usually necessitates the use of embedded Operating Systems (OS). On microcontroller-based devices, despite the highly constrained nature of such platforms in terms of RAM, ROM and CPU frequency, OSes provide real-time capabilities, multithreading, and advanced wireless communication stacks.

RIOT [2] is an operating system for microcontrollers designed around a tickless and real-time microkernel. It provides support for a wide range of microcontroller architectures, from 8-bit to 32-bit, including ARM, AVR, MSP430, Xtensa, and RISC-V. RIOT also comes with high-level libraries to make the developer's life easier such as a `shell` or `ztimer`. RIOT targets IoT applications with a focus on standard wireless protocols, such as IEEE 802.15.4, LoRa and BLE.

The Single-Chip Mote (SCuM) [5] is the first millimeter-scale system-on-chip which features a radio capable of interoperating with standard communication protocols without need for an external oscillator. This design requires calibration and compensation of the internal clocks in order to have functional communication over radio or UART.

We introduce the first port[1] of a real-time operating system (RTOS), RIOT, over SCuM. We choose RIOT because it is one of the main open-source RTOS available, with Contiki-NG, FreeRTOS, and Zephyr. This port allows developers to use any feature available in RIOT with SCuM provided that SCuM has the required peripherals and memory.

[1] As an online addition to this paper, the port is published on GitHub at https://github.com/aabadie/RIOT/tree/scum_support.

2 BACKGROUND

SCuM is a custom ARM Cortex-M0 based microcontroller developed as a research project at UC Berkeley in 2019. So far, the SCuM SDK [1] has been used to build, load, and boot a firmware for SCuM. SCuM is a constrained system with no off-chip components and only 128 kB of RAM and no non-volatile flash. The RAM is divided into 64 kB for volatile data and 64 kB for code instructions. Furthermore, it only provides 16 GPIOs, one single hardware timer (`RFTimer`), a single UART interface, and a reconfigurable 2.4 GHz radio compatible with the IEEE802.15.4 and BLE standards. Despite its limitations, Chang *et al.* have demonstrated that SCuM can be used in industrial applications of the IoT [3].

In the absence of an external oscillator, SCuM requires a calibration procedure to be performed for correct functioning, especially when using the radio. This is usually done by connecting an external board as a clock source to SCuM. Maksimovic *et al.* have proposed a radio-based calibration procedure where an oscillator-enabled radio device is used to provide a reliable clock source [4].

RIOT uses a modular approach to build application firmware. The philosophy is to only build what is needed to keep the generated binary as small as possible. The RIOT build system uses a large set of Makefiles for many purposes, including setting up toolchains, module dependency resolution for a given application, or integration of external tools such as programmers or serial terminals. To ensure portability across the various supported architectures, RIOT provides a hardware abstraction layer (HAL). Then, on top of the HAL, high-level libraries work transparently without requiring hardware-specific code. RIOT also has support for `newlib-nano` and `picolibc` C libraries, which provide partial compatibility with POSIX APIs.

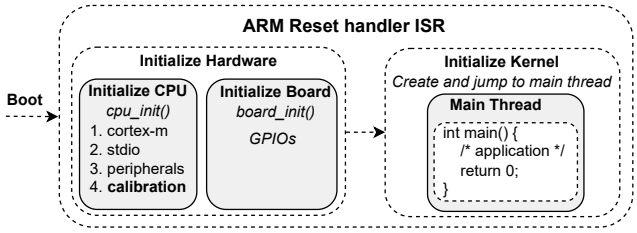

Figure 2: RIOT boot sequence on SCuM.

3 PORTING RIOT TO SCUM

Fig. 2 describes the RIOT boot sequence on an ARM core. In the case of SCuM, the additional and required calibration step is appended to the initialization of the CPU.

Build System Integration. The SCuM SDK provides a CMSIS header file that contains the definitions of the SCuM memory and the code used to perform the clock calibration of the chip. The port takes advantage of the RIOT external package mechanism to fetch the SCuM SDK from GitHub and to include the required files to the build. The SCuM Dev Kit provides the RIOT build system variables needed to call the `scum-programmer` script when the `flash` target is invoked.

HAL Drivers. The peripherals provided by SCuM have very basic features. For instance, the UART peripheral baudrate is fixed and hardwired to external pins, the RFTimer peripheral clock frequency is also kept at 500 kHz, GPIOs have fixed internal pull-down resistors. For these peripherals, the HAL driver implementations are thus very trivial to write, but enough to get support for the `ztimer` and `shell` system APIs of RIOT. Unfortunately, there is no hardware power management feature on SCuM so the RIOT `pm` HAL used to automatically switch between power modes cannot be adapted.

Memory Footprint. Table 1 shows the memory footprints of different applications built with RIOT and the SCuM SDK :

- `hello_world` is the usual basic application that prints a message to stdio,
- `ipc_pingpong` is a RIOT application that creates a thread and exchanges IPC messages between the *main* thread and the created thread,
- `aes128` provides a shell with commands to encrypt and decrypt a given message using the AES128 CTR mode,
- `emlearn` uses a random-forest model to detect an hand-written digit in an image.

All firmwares are compiled in *Release* mode without debugging symbols and optimized for size. Note that the memory footprint of the `hello_world` application given for RIOT and the SCuM SDK. Also, RIOT firmwares sizes are given when using newlib-nano and picolibc, two popular libc for embedded. When comparing the `hello_world` application, we see that the RIOT kernel adds a memory overhead of 1.86 kB compared to the SDK. But when using picolibc, the kernel overhead is largely compensated by the saved memory. In general, all RIOT firmwares built with picolibc have a much lower memory footprint than their equivalent built with newlib-nano.

Table 1: Memory footprint of some sample applications for different build configurations.

Case	libc	Application	ROM	RAM
SCuM SDK	newlib	hello_world	8,112 B	1,696 B
RIOT	newlib	hello_world	9,976 B	2,632 B
RIOT	picolibc	hello_world	6,920 B	2,532 B
RIOT	newlib	ipc_pingpong	10,740 B	4,168 B
RIOT	picolibc	ipc_pingpong	7,684 B	4,068 B
RIOT	newlib	aes128	18,000 B	2,924 B
RIOT	picolibc	aes128	14,164 B	2,824 B
RIOT	newlib	emlearn (AI)	43,508 B	2,716 B
RIOT	picolibc	emlearn (AI)	39,852 B	2,616 B

4 CONCLUSION & FUTURE WORK

In this paper, we present the first port of an RTOS that targets the SCuM chip. We also show that, when using `picolibc`, the RIOT kernel overhead becomes negligible and, compared to the SCuM SDK, the generated firmware for the `hello_world` application is 13.7% smaller. This port allows embedded application developers to use libraries provided by RIOT on SCuM, if they fit in memory or only use peripherals available on SCuM: `ztimer` or `shell` system libraries, machine learning, cryptography, and build any other RIOT demo applications.

Once the SCuM radio is adapted to the RIOT radio HAL, many communication stacks could then be used: RIOT internal GNRC, OpenThread, OpenWSN, LwIP, and NimBLE. Other future planned work will focus on providing ports for other existing RTOS such as FreeRTOS and Zephyr in C, or Ariel OS in Rust.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon Europe Framework Programme, project "OpenSwarm", under Grant Agreement No. 101093046.

REFERENCES

[1] A. Abadie. 2025. SCuM SDK. Retrieved 2025 from https://github.com/PisterLab/scum-sdk

[2] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S Lenders, H. Petersen, K. Schleiser, T. Schmidt, and M. Wählisch. 2018. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (2018), 4428–4440.

[3] T. Chang, T. Claeys, M. Vučinić, X. Vilajosana, T. Yuan, B. Wheeler, F. Maksimovic, D. Burnett, B. Kilberg, K. Pister, et al. 2020. Industrial IoT with Crystal-Free Mote-On-Chip. In *VLSI*. IEEE, New York, NY, USA, 1–2.

[4] F. Maksimovic, A. Patel, D. Burnett, T. Watteyne, and K. Pister. 2023. A Time Synchronized Multi-Hop Mesh Network with Crystal-Free Nodes. In *IEEE Global Communications Conference (GLOBECOM)* (Kuala Lumpur, Malaysia). IEEE, New York, NY, USA, 4158–4163.

[5] F. Maksimovic, B. Wheeler, D. Burnett, O. Khan, S. Mesri, I. Suciu, L. Lee, A. Moreno, A. Sundararajan, B. Zhou, et al. 2019. A Crystal-Free Single-Chip Micro Mote with Integrated 802.15.4 Compatible Transceiver, Sub-mW BLE Compatible Beacon Transmitter, and Cortex M0. In *VLSI* (Kyoto, Japan). IEEE, New York, NY, USA, C88–C89.