Porting RIOT-OS to the RIoT-Board

Jakob Lennart Dührsen
Institute of Computer Science
Freie Universität Berlin
Takustrasse 9, 14195 Berlin, Germany
lennart.duehrsen@fu-berlin.de

Leon Martin George
Institute of Computer Science
Freie Universität Berlin
Takustrasse 9, 14195 Berlin, Germany
leon@georgemail.eu

In early 2014, Freescale introduced a new board targeting the Internet of Things, the **RIoT-Board**. RIoT is an abbreviation for »Revolutionizing the Internet of Things«.
Compared to other typical devices used in IoT it is quite powerful, featuring an ARM(R) Cortex A9 processor running at 1GHz, 4GB MMC, Gigabit-Ethernet and a graphics processor that allows playback of 1080p videos and supports OpenGL ES 2.0 .
RIOT OS on the other hand is an operating system for constrained devices, as found frequently in IoT context. It is optimized for low power and memory consumption.
To enable the operation of RIOT OS on the RIoT-Board, we will port RIOT OS to the ARM Cortex A9 architecture as a software project at Freie Universität Berlin.

Most parts of an OS can be written independently from the underlying hardware, which allows for easier portability. Some parts, however, have to be rewritten for every new architecture, such as the bootloader and the context switch.

As we have the boards in our hands now, the question is where to start. The following will give a compendium on what we have already done and are planning to do in the near future to let RIOT OS run on the RIoT-Board.
Our first step was to get familiar with the board itself. It ships with Android pre-installed, so we could boot it immediately and check out some of its features. A linux derivative based on Ubuntu is also available, and was testet right after Android. Unfortunately, linux can not be booted from the SD card, so we had to find out how to flash the board. If you have a Windows computer, this is easier than expected. Freescale provides a tool to flash the MMC using a mini USB cable. With Linux running, we were able to blink the user LEDs.
For further inspection of the RIoT-Board's inner workings, we installed U-Boot, a widely used bootloader for embedded systems. U-Boot offers a shell, to which you can connect using UART. This set up, we were able to run our own bare-metal code.

Before coding can start, the hardware-specific parts of RIOT OS have to be identified. In the core functionality of RIOT OS, these are the two functions `board_init()` and `kernel_init()`, which are the first to be called during booting and supposed to set up frequencies, voltages, stacks and other necessary things to run properly.

As we are not ARM experts, the first step here is to understand the target architecture, which means reading lots of documentation. This work is still in progress and probably will be for quite some time.
Further things we need to learn include Makefiles and ARM assembly.

As soon as we are skilled enough to start working, the plan is to make I/O via UART work, to enable comfortable debugging and later access to the shell. After that, code to initialize the timers is needed, because the RIOT OS kernel depends on these. The lasts steps will then be the `board_init()` and `kernel_init()`.

For that, we have agreed on the following (soft) deadlines:

- 31.05. UART and timers working
- 31.05. `board_init()` working
- 31.06. `kernel_init()` working
- 07.07. more realistic deadline for the above

As of today, we are not able to tell how the work will be split, because the exact approach for each milestone is not clear yet. So far, our attitude was to identify the next small task or problem, respectively, start reading and hacking and whoever finds a solution first explains it to his teammate and we start again. This has worked pretty good, so we decided to keep it that way until we know what we're doing by splitting work in a planned way.

All documentation, by us and by vendors, can be found in this `git` repository:

`https://github.com/backenklee/swp-report`