Porting RIOT OS to the ARM Cortex™-A9

Jakob Lennart Dührsen and Leon Martin George
Institute of Computer Science
Freie Universität Berlin
Takustrasse 9, 14195 Berlin, Germany
Email: {lennart.duehrsen, leon.george}@fu-berlin.de

***Abstract*—This is a report on the process of doping a basic port of RIoT-OS to the RIoTboard as part of the software-project 2014 of the working-group "Computer Systems & Telematics" at Freie Universität Berlin. The port is based on the i.MX6 Platform SDK 1.1 and at the end of the project it is in a state where the UART, timers and interrupts have interfaces to RIoT-OS.**

I. Introduction

In Douglas Hofstaedter applies Godel's seminal contribution to modern mathematics to the study of the human mind and the development of artificial intelligence.

II. Preparing For The Port

In order to complete the project successfully, we agreed on these goals with the teaching-staff:

- The RIoT-OS-wiki has pages on working with the hardware
- RIoT-OS compiles for the RIoTboard
- Interfaces to timers, UART and interrupts is implemented

As we had no idea how hard it was going to be, we were very generous with the curfews for the different steps.

A. The Hardware

The RIoTboard is a rather unusual board for the RIoT-OS to run on. It is based on the "i.MX6Solo" freescale-architecture, has an ARM cortex-A9, many different interfaces and supposed to be used mainly by developers.

B. Software Running on the RIoTboard

On the boards website the manufacturer, embest-tech, offers binary-images and instructions for running android or ubuntu, of which android is installed by default. Both operating systems rely on u-boot to do low-level hardware-initialisation. The source-code of those ports is available in moderately-hard-to-find repositories on the internet and the code itself could somehow be re-useable.

Due to their structure, the linux-code and the u-boot-fork were useful in different ways: u-boot requires the seperation of syscalls, the "flash-header" - which helped understand the way the i.MX6 boots and linker script. So with u-boot embest-tech provides three files for those and one giant source-file for all the rest. The linux-port seems better structured but due to us lacking the knowledge about the kernel it was harder to use any parts of the source.. Also, as far as initialisation goes, linux seems to repeat some of the steps that have already been done by u-boot and does some differently.

The focus on developers is somehow restricted to those two operating systems. There is no official support apart from discussions on a channel of an IoT/embedded-focused developer forum.

III. Running our own Software

The simplest thing we could get to run on the RIoTboard was u-boot: Either via the serial connection, where the freshly transferred u-boot and an initramfs could be used to receive files or binary that could then be stored on the board, or by writing the binary onto a SD-card. From within the running u-boot it was possible to execute a method from a cross-compiled object-file that simply returned an integer value.

We were trying to figure out how to use u-boot for booting and supplying the standard library but all the files were spread all over the u-boot-source-directory. As we were realising how difficult it was going to be to not only have a functional binary based on our own code but also integrate the existing structure into the make-system of RIoT-OS, we didn't think we would be able to meet any goal beyond supplying a rudimentary framework so that coding can start in the RIoT-OS-codebase.

Luckily, freescale supplies a SDK for the i.MX6-platform that is able to run on their reference-boards. We agreed with the teaching-staff to rather use the SDK because it is closer to just developing applications in a C-manner and the source-tree is well-structured without making it too hard to find a particular piece of code and also, there are hardly any source-files with irrelevant code.

Then again, sadly, none of the targets for the SDK seemed to produce code that ran on the RIoTboard.