⇓

1 Motivation

⇓The Board ↓
comes with instructions for android and linux. android preinstalled ↓
1 GHz, single-core
↓
SOC for mm-applications
↓
chips for decoding x264, 2d- and 3d-graphics
↓
USB, eth, sound, HDMI, Display-Port, dev-stuff like JTAG, UART Can boot from internal memory, or SD-card

⇓Why Do We Want This? RIOT-OS is not known for it's need to run on phones. With i.MX6-support RIoT-OS could be used for ↓
automotive,
↓
industrial,
↓
handheld consoles, ↓
easy developement-¿This often is underestimated. Most microprocessors are painful to target for developing. The RIoTboard is aimed at developers: You can have different programs on the board itself or on several SD-cards. Flashing can be done to either targets via USB or directly to an SD-Card without having to destroy the data on the partitions (after some preparation). The fully-developed program can probably be used for any other i.MX6-board without too many modifications, if any (maybe IOMux brrrrrrrrrr).

2 Our original idea of how to do shit

⇓Initial Work-Model This was easy. Many documents. ↓
We have to documents from the manufacturer: One is a schematic and the other is specification for the hardware with instructions on flashing either the SD -card or the internal eMMC (micro-SD they forgot somehow) with ubuntu or android. Glad there were instructions for this. ↓
After having realised that we wouldn't be able to flash the ROM, we tried going through u-boot and loaded just a main() onto the board. ↓
We wanted to use u-boot but the IOMux-configuration would then have been fixed and it is usually a bad idea to change that (what RIoT would have ultimately done). Plus, the teaching staff convinced us that the i.MX6-SDK is much nicer. Somehow, we have got a working framework. Even though we could, in theory, start getting specific hardware to work with RIoT-OS, there is a significant problem.

3 Achievements

⇓The Original Plan: from the orinal slides: ↓
uart. erm. problems. everything set up, but no cigar
↓
timers are running by default, but not yet usable by RIoT (or in fact: the UART)
↓
interrupts are enable, but no implementation for interface to RIoT
↓
set-up stack, done (works nicely due to the SDK)
↓
This worked after about a month

⇓Unfinished Work The slide should rather be called "The One Big Problem" because there is this one thing that is really bugging us: ↓
We tried the default configuration from the SDK, diy-UART-init, inserted adapted u-boot-code, tried configs for other boards and made some ourselves with the IOMux-config-tool from the SDK which generates headers with register -definitions and macro-implementations.
↓
Then there is timers ↓
and interrupts
but with the only working debugging facilities being to LEDs, this is a rather terrifying task

⇓Interference This was initially a drawback because it took us significantly longer to integrate the SDK than it would have with u-boot(mainly because the board- specific code from u-boot is squeezed into just 4 source-files. But after the make-process of the SDK was integrated into RIoT-OS it was really easy to adjust the start-process. So by then we were early. ↓
Nuff said on that.

4 Future Tense

⇓Plans Other IOMux-confs might offer insight on what is going wrong ↓
Avoid the UART-related problems to allow getting to work on other components