TASK ASSIGNMENT FOR A
MASTER THESIS AT THE DEPARTMENT OF
INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

FALL SEMESTER 2022

Noah Zarro

Rust RTOS on PULP

March 31, 2023

Advisors: Robert Balas, ETZ J78, Tel. +41 44 632 42 56, `balasr@iis.ee.ethz.ch`
Alessandro Ottaviano, ETZ J89, Tel. +41 44 632 57 45, `aottaviano@iis.ee.ethz.ch`
Michael Rogenmoser, ETZ J69.2, Tel. +41 44 632 54 33, `michaero@iis.ee.ethz.ch`

Professor: Prof. Dr. L. Benini

Handout: October 3, 2022
Due: April 3, 2023

1 Introduction

PULP (Parallel Ultra-Low-Power) [1] is an open-source multi-core computing platform. It consists of an advanced microcontroller architecture with a parallel computing cluster composed of 8 RISC-V, fully programmable 32-bit processing elements (PE) featuring DSP extensions targeting energy-efficient digital signal processing [2]. This computing cluster serves as an accelerator.

ControlPULP is a specialized version of PULP that focuses on predictability. It is used in the European Processor Initiative (EPI) [3] playing the part of a Power Controller Subsystem (PCS) dynamically adjusting the operating point of a High Performance Computing (HPC) processor to meet energy, power, and thermal constraints. The PCF executes a multi-task control-law that involves optimal operating point computation and complex multi-input, multi-output interaction with the external world. To schedule the PCF tasks the FreeRTOS real-time OS layer is used.

Programming embedded systems such as ControlPULP traditionally means using the C programming language which uses manual memory management. Bugs related to memory safety issues (e.g. buffer overflows) make up to 70% of all vulnerabilities [4]. Rust is a programming language [5] that provides compile time guaranteed memory safety without using a garbage collector or run-time. The Rust programming language claims that memory unsafe operations are clearly denoted and all other code is proven by the compiler to be safe while at the same time not sacrificing efficiency. In practice, this works out well modulo language soundness related bugs [6][7].

Real-time operating systems (RTOS) are operating systems (OS) that are tailored towards running real-time applications. There are plethora of RTOS' in use nowadays. FreeRTOS [8], written in C, is one such popular open-source RTOS which also supports ControlPULP. In the Rust world, there are upcoming RTOS such as Tock [9], Bern RTOS [10], RTIC [11] and more.

2 Project Description

The goal of this work is to evaluate Rust-based RTOS' and port one to ControlPULP including the minimal required number of drivers (uart, timer). Then, compare the current FreeRTOS-based solution to the ported RTOS in terms of features and benchmark performance (interrupt latency, context switch time, data sharing performance between tasks, ...). Based on these results propose and explore software (e.g. specific Rust language feature such as fearless concurrency, RCU, atomics instead of critical sections, ...) and hardware modifications (e.g. hardware-based mutexes or queues, ...). Another interesting topic to look at this to study and implement offloading schemes to the 8-core accelerator cluster and compare against existing solutions in FreeRTOS.

3 Milestones

The following are the milestones that we expect to achieve throughout the project:

- Analyze and evaluate existing Rust-based RTOS'
- Set up a embedded rust compilation and simulation flow
- Port Rust-based RTOS' to ControlPULP
- Benchmark Rust-based RTOS' comparing it against existing FreeRTOS solution in terms of interrupt latency and jitter, context switching etc. For comparisons using a more real-world work-load the existing power control firmware [12] can be ported too. Evaluate Rust specific language features related to memory safety and concurrency primitives if applicable.
- Propose, explore and potentially implement software or hardware modifications based on obtained measurements.

3.1 Stretch Goals

Should the above milestones be reached earlier than expected and you are motivated to do further work, we propose the following stretch goals to aim for:

- Explore, implement and benchmark offloading schemes to the 8-core accelerator cluster
- Implement drivers for SPI, I2C, AVSbus.

4 Project Realization

4.1 Time Schedule

The time schedule presented in Table 1 is merely a proposition; it is primarily intended as a reference and an estimation of the time required for each required step.

4.2 Meetings

Weekly meetings will be held between the student and the assistants. The exact time and location of these meetings will be determined within the first week of the project in order to fit the student's and the assistants' schedule. These meetings will be used to evaluate the status and progress of the project. Beside these regular meetings, additional meetings can be organized to address urgent issues as well.

Project phase	**Time estimate**
Learn Rust and study ControlPULP (sw stack, sim tools, ...)	4 weeks
Compare existing Rust-based RTOS'	1 week
Port RTOS to ControlPULP	4 weeks
Use existing or ported benchmarks with resulting system; propose and implement software/hardware modifications. This is an iterative process that requires going back and forth	11 weeks
Write report	2 weeks
Prepare presentation	2 week

Table 1: Proposed time schedule and investment

4.3 Weekly Reports

The student is required to a write a weekly report at the end of each week and to send it to his advisors by email. The idea of the weekly report is to briefly summarize the work, progress and any findings made during the week, to plan the actions for the next week, and to discuss open questions and points. The weekly report is also an important means for the student to get a goal-oriented attitude to work.

4.4 Coding Guidelines

HDL Code Style Adapting a consistent code style is one of the most important steps in order to make your code easy to understand. If signals, processes, and modules are always named consistently, any inconsistency can be detected more easily. Moreover, if a design group shares the same naming and formatting conventions, all members immediately *feel at home* with each other's code. At IIS, we use lowRISC's style guide for SystemVerilog HDL: `https://github.com/lowRISC/style-guides/`.

Software Code Style We generally suggest that you use style guides or code formatters provided by the language's developers or community. For example, we recommend LLVM's or Google's code styles with `clang-format` for C/C++, PEP-8 and `pylint` for Python, and the official style guide with `rustfmt` for Rust.

Version Control Even in the context of a student project, keeping a precise history of changes is *essential* to a maintainable codebase. You may also need to collaborate with others, adopt their changes to existing code, or work on different versions of your code concurrently. For all of these purposes, we heavily use *Git* as a version control system at IIS. If you have no previous experience with Git, we *strongly* advise you to familiarize yourself with the basic Git workflow before you start your project.

4.5 Report

Documentation is an important and often overlooked aspect of engineering. A final report has to be completed within this project.

The common language of engineering is de facto English. Therefore, the final report of the work is preferred to be written in English.

Any form of word processing software is allowed for writing the reports, nevertheless the use of LaTeX with Inkscape or any other vector drawing software (for block diagrams) is strongly encouraged by the IIS staff.

If you write the report in LaTeX, we offer an instructive, ready-to-use template, which can be forked from the Git repository at `https://iis-git.ee.ethz.ch/akurth/iisreport`.

Final Report The final report has to be presented at the end of the project and a digital copy needs to be handed in and remain property of the IIS. Note that this task description is part of your report and has to be attached to your final report.

4.6 Presentation

There will be a presentation (15 min presentation and 5 min Q&A) at the end of this project in order to present your results to a wider audience. The exact date will be determined towards the end of the work.

5 Deliverables

In order to complete the project successfully, the following deliverables have to be submitted at the end of the work:

- Final report incl. presentation slides
- Source code and documentation for all developed software and hardware
- Testsuites (software) and testbenches (hardware)
- Synthesis and implementation scripts, results, and reports

References

[1] (2022) Parallel ultra-low-power open-source multi-core computing platform. [Online]. Available: https://github.com/pulp-platform/pulp

[2] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[3] (2022) European processor initiative. [Online]. Available: https://www.european-processor-initiative.eu

[4] M. Miller. Trends, challenges and strategic shifts in the software vulnerability mitigationG landscape. [Online]. Available: https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/2019_01%20-%20BlueHatIL%20-%20Trends%2C%20challenge%2C%20and%20shifts%20in%20software%20vulnerability%20mitigation.pdf

[5] (2022) Rust Programming Language. [Online]. Available: https://www.rust-lang.org/

[6] H. Xu, Z. Chen, M. Sun, and Y. Zhou, "Memory-safety challenge considered solved? an empirical study with all rust cves," *CoRR*, vol. abs/2003.03296, 2020. [Online]. Available: https://arxiv.org/abs/2003.03296

[7] B. Qin, Y. Chen, Z. Yu, L. Song, and Y. Zhang, "Understanding memory and thread safety practices and issues in real-world rust programs," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 763–779. [Online]. Available: https://doi.org/10.1145/3385412.3386036

[8] (2023) Freertos. [Online]. Available: https://www.freertos.org/

[9] (2023) Tock. [Online]. Available: https://www.tockos.org/

[10] (2023) Bern rtos. [Online]. Available: https://bern-rtos.org/

[11] (2023) Rtic. [Online]. Available: https://rtic.rs/1/book/en/

[12] G. Bambini. (2022) An open-source power and temparature control firmware. [Online]. Available: https://iis-git.ee.ethz.ch/giovanni.bambini/epi_pmu_ethz

[13] S. Klabnik and C. Nichols. The rust programming language. [Online]. Available: https://doc.rust-lang.org/book/ch16-00-concurrency.html

Zurich, March 31, 2023 Prof. Dr. L. Benini