UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

Computer Engineering
CEN3907C

Concurrency Frameworks for Embedded Systems

Overview

Concurrent programs are incredibly useful and powerful but are often difficult to develop and debug. The challenge is exacerbated when we have no operating system to help us safely share resources on our hardware, which is often the case when developing for small microcontrollers. In this exercise, you will explore two frameworks for concurrency on embedded systems, one in the Rust programming language, and one built on top of the CircuitPython interpreter.

The **RTIC framework** – for "Real-time Interrupt-driven Concurrency" – is an excellent approach to concurrent development on embedded systems in the Rust programming language. This framework enables *preemptive multitasking* by utilizing the hardware interrupts to schedule *tasks*, which might use or manipulate shared resources. Because of its interrupt-driven nature, RTIC programs can have guarantees on real-time performance and perform hardware triggered tasks without the need to poll the peripherals.

The **asyncio** package, available for use with Circuitpython, enables *cooperative multitasking*, where coroutines cooperate to share resources on the system. The benefits of using CircuitPython – including convenience-of-development, familiarity of a widely-used general purpose scripting language, and an active community with great hardware support – are balanced with fewer guarantees on performance. If a task is unfair or has blocking calls with it, it can prevent other tasks from executing in a timely manner.

Learning Objectives

- Gain a working knowledge of the **RTIC** framework for Rust, and the **asyncio** package for CircuitPython.
    - Understand the differences between preemptive and cooperative multitasking.
    - Appreciate the use-cases for each framework, and the trade-offs between performance and development workflow.
- Build a concurrent application and measure performance properties using each of the frameworks.
    - Utilize test equipment to perform relevant and meaningful experiments.
    - Support your conclusions with quantitative results.

Resources

The following resources should provide you with enough information to complete this assignment. If you have not done so already, follow the instructions in the HW Intro document to get a "hello world" example running in Rust and CircuitPython. If you did not complete the previous hardware labs (Drivers for Embedded Systems, and Introduction to Embedded Rust), read through those assignments to understand the requirements for using your hardware kit. Some of the resources provided in those documents are relevant here as well.

Template code is provided for the Rust application as a *zip* archive on Canvas. This code outlines the structure of the application and has two tasks done for you – one that runs when the USB interrupt is fired (allows the host to recognize the USB serial device), and another that prints a static string to the serial port. The rest of the implementation is up to you.

Please read through this entire document before starting development.

- The RTIC Book
  - Provides an overview of the framework and the required attributes and data structures to implement an RTIC application.
- rp-hal GitHub
  - Provides numerous examples of using Rust on the RP2040 chip. There are examples (see *rp2040-hal-examples*) of using RTIC (note that these examples may need to be adapted slightly for use on your specific development board).
- Docs.rs
  - Provides API documentation for Rust crates. Search for the crate you are interested in, i.e. *rp2040_hal*, *adafruit_feather_rp2040*, or *ws2812_pio*.
- Overview of CircuitPython
  - Provides an overview of the CircuitPython ecosystem. Please use this guide as a reference, but do not necessarily follow all the steps exactly. For instance, while you can use the Mu Editor, you can also effectively use VSCode to edit your *code.py* file, and a serial port program such as *minicom* (linux), *putty*, or *nRF Terminal* (Windows and VSCode extension, respectively).
- Cooperative Multitasking in CircuitPython
  - An excellent introduction to using **asyncio** with CircuitPython.
- API Documentation for asyncio
  - Provides documentation and examples for available asyncio methods.

Specification

For this lab, you shall implement two applications: one using the RTIC framework in Rust, and the other using the asyncio package in CircuitPython. Both applications should have the following behavior:

- Blinks the build-in LED at 10Hz.
- Invokes a "heartbeat" task at 0.5Hz that prints a message to the USB serial device.
- Displays a random pixel with a random color at 5Hz on the NeoMatrix.

Furthermore, you should measure the timing properties of the LED blink rate. Using an oscilloscope, measure the period on the toggling pin 10 times and tabulate the results for each application. Below the table, state the average period and the jitter.

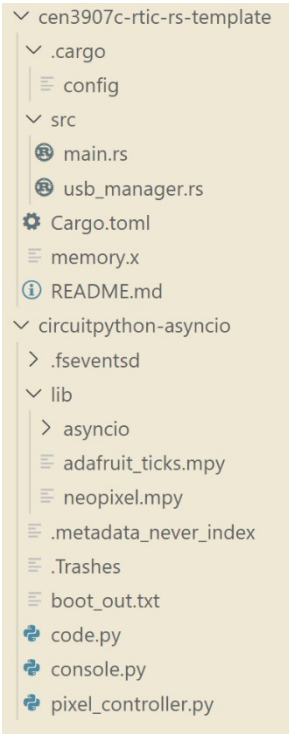
Submission

Your submission will be composed of the following elements:

- Your complete source code as a **ZIP archive on Canvas**. This will be extracted into a folder, then compiled and loaded using *cargo run* (for the Rust app), or run by copying the *lib* files and *.py* files onto the CircuitPython device.
- **Report in PDF format on Canvas**.

ZIP Archive

Your ZIP archive should have a directory/file structure like that shown.



The two projects (Rust code and CircuitPython code) should be siblings under your submission folder.

- To run the Rust code, I will enter the Rust subfolder and run *cargo run* with a mounted RP2040 connected.
- To run the CircuitPython code, I will copy all files under the *circuitpython-asyncio/lib* subfolder onto my mounted *CIRCUITPY/lib* folder, as well as all the files ending in *.py* into the mounted *CIRCUITPY* drive.
- *Note:* many of the files preceded by dots (hidden files) are not required to be in your submission (but it's okay if they are). Use your intuition to decide what files are *needed* and which are only relevant to your local system.

Report

When filling in the report template, in addition to your timing measurements, please include responses to the following questions:

- How was your development experience using CircuitPython vs RTIC?
- How did the asyncio implementation compare to the RTIC one? What are some tradeoffs to consider with designing a concurrent embedded application?
- Do you have any suggestions for improvement to the structure of this project?