**Zephyr RTOS Scheduling Visualizer**

EECE 4811 - Operating Systems

Himadri Saha, Ashwin Srinivasan, Yaritza Sanchez

**Abstract**

This paper presents the development of a Zephyr RTOS Scheduling Visualizer for a Raspberry Pi Pico that should enable the visibility and examination of real-time operating system (RTOS) scheduler behavior under varying loads on the system. The visualizer makes monitoring of basic scheduling parameters like execution time, context switches, CPU usage by thread, start-time jitter, deadline misses, and USB interrupt-to-handler latency possible. To simulate practical usage, the system executes multiple threads at different priority levels (high, medium, low) with configurable synthetic "noise" workloads. The thread parameters and system stress conditions are dynamically controllable via a USB serial shell interface for real-time configuration and monitoring. Data collected under runtime are exported as CSV for off-line analysis and visualization. This tool is intended to help students and developers better understand and optimize RTOS scheduling phenomena on constrained embedded systems, revealing insight into the trade-offs and performance implications of scheduling policy decisions.

**Motivation + Problem Statement**

Real-time operating systems (RTOS) are widely used in embedded systems to guarantee timing constraints, ensure predictable scheduling, and handle concurrent tasks under strict deadlines. However, the internal behavior of an RTOS scheduler, especially under varying system loads—can be difficult to observe directly. Without visibility into key performance metrics such as jitter, deadline misses, execution time, context switching frequency, and interrupt latency, it is challenging to evaluate whether the system meets real-time requirements or to tune scheduling policies effectively. To address this gap, we propose the development of an RTOS Scheduling Visualizer. The system will run multiple threads of different priorities (high/medium/low) alongside configurable synthetic “noise” loads to stress the scheduler.

Motivation for the project is acquiring knowledge on how to get metrics when stress testing in order to improve performance on this and future projects. Anyone using this device will benefit greatly as they can go in and fine tune the scheduler to their specifications. If somebody could improve the efficiency of said hardware, then the scheduler would be of great value.

**Related Work**

1. “Raspberry Pi Documentation.” *Raspberry Pi*, [www.raspberrypi.com/documentation/](https://www.raspberrypi.com/documentation/). Accessed 6 Oct. 2025.

- Website full of detailed explanations on how to use the primary hardware chosen for the project

1. “Raspberry Pi 4 Model B (Cortex-A72).” *Raspberry Pi 4 Model B (Cortex-A72) - Zephyr Project Documentation*, docs.zephyrproject.org/latest/boards/raspberrypi/rpi\_4b/doc/index.html. Accessed 6 Oct. 2025.

- Zephyr is the primary software that’s going to be used, this website provides some starting code from the developers

1. Natarajan, ThamizhElango. “Complete Guide: Running Zephyr RTOS on Raspberry Pi 4 & Pico.” *Medium*, Medium, 17 Sept. 2025, thamizhelango.medium.com/complete-guide-running-zephyr-rtos-on-raspberry-pi-4-pico-9bcb9244cc29.

- Introductory course on how to get RTOS scheduler up and running as this will be a crucial part of the project

1. “Basic Threads on Zephyr RTOS.” *Linaro*, [www.96boards.org/blog/basic-threads-zephyr/](https://www.96boards.org/blog/basic-threads-zephyr/). Accessed 6 Oct. 2025.

- Reference for how to code low, medium, high priority threads

1. AjayAjay 7511 silver badge1010 bronze badges, et al. “How to Get System Time in Zephyr RTOS?” *Stack Overflow*, 1 Oct. 1964, stackoverflow.com/questions/62798884/how-to-get-system-time-in-zephyr-rtos.

- Timestamp logging: Use Zephyr’s cycle counter (k\_cycle\_get\_32) or k\_uptime\_get() to measure scheduling events. Some users put their code on how to run it here

1. *Technical Documentation*, docs.nordicsemi.com/bundle/ncs-2.5.3/page/zephyr/kernel/services/timing/clocks.html. Accessed 6 Oct. 2025.

- Website all on kernel timings and how to troubleshoot delayed timings

1. Manstein•caronosaurus, Philipp. “Debug the Raspberry Pi Pico Running Zephyr Rtos on Macos.” *Hackster.Io*, 19 Sept. 2023, [www.hackster.io/philippmanstein/debug-the-raspberry-pi-pico-running-zephyr-rtos-on-macos-5b4436](https://www.hackster.io/philippmanstein/debug-the-raspberry-pi-pico-running-zephyr-rtos-on-macos-5b4436).

- One of the members of the group has a MAC (Ashwin) so this is reference code for debugging

1. “Board Index.” *Raspberry Pi Forums - Index Page*, forums.raspberrypi.com/viewtopic.php?t=54676. Accessed 6 Oct. 2025.

- Context switch : Use Zephyr’s tracing hooks or instrument your own counters. Reference code with links to GitHub repo’s.

1. *Technical Documentation*, docs.nordicsemi.com/bundle/ncs-2.1.0/page/zephyr/kernel/timing\_functions/index.html. Accessed 6 Oct. 2025.

- Execution time explanation and code right from Zephyr’s website

1. “Board Index.” *Raspberry Pi Forums - Index Page*, forums.raspberrypi.com/viewtopic.php?t=345162. Accessed 6 Oct. 2025.

- Worst comes to worst, we can stress the Pico by increasing tick rate if the desirable outcomes aren’t achieved.

**Potential Challenges**

* Accurate Timing Issues: The Raspberry Pi Pico has limited hardware timers and Zephyr’s scheduler already consumes system resources. Capturing accurate start-time jitter, execution times, and interrupt latency may be affected by measurement overhead.
* Scheduler Overhead & Intrusiveness of Measurement: Adding instrumentation for logging context switches and CPU usage could itself alter scheduler behavior and distort results.
* Thread Priority Inversion and Deadlock: Running high/medium/low-priority threads with noise may expose priority inversion or even deadlock if resources are not carefully handled.
* Memory strain
* Existing knowledge; C+, Arduino, FPGA-10, Data Structures, Quartus, Microprocessors
* Must learn: Foundational RTOS & Scheduling Concepts, Zephyr RTOS, Raspberry Pi Pico, System load & Stress testing, Zephyr debugging
* Resources: Raspberry Pi Manual, Zephyr code manual, AI Tools, GitHub, YouTube

**Timeline / Expected Milestones**

|  |  |  |
| --- | --- | --- |
| **Week** | **Milestone** | **Description** |
| **Week 1–2**  **(Oct 18)** | **Project Setup** | * Set up Zephyr RTOS v4.2 environment for Raspberry Pi Pico. * Verify USB serial connection and Zephyr shell functionality. * Implement three threads with distinct priorities (High, Medium, Low). * Confirm proper priority-based scheduling operation. |
| **Week 3–4**  **(Nov 1)** | **Basic Thread Scheduling** | * Implement a “noise” load task to simulate background CPU usage. * Test scheduler performance under various load conditions. * Begin collecting real-time metrics:   + Start-time jitter   + Execution time   + Deadline misses * CSV export for metrics working. |
| **Week 5**  **(Nov. 6 –Midterm Presentation)** | **USB Shell Interface + Demo** | * Demonstrate a working RTOS visualizer with:   + High/medium/low-priority threads running concurrently + noise load   + Live measurements of start-time jitter, execution time, and deadline misses * Functional USB shell to adjust priority, timeslice, load. * Initial analysis of scheduler performance under varying conditions. |
| **Week 6-8**  **(Nov 29)** | **Metric Expansion and validation** | * Added metrics:   + Context switch count   + CPU utilization per thread   + USB interrupt-to-handler latency * Logging validated and compared across load scenarios. |
| **Week 9**  **(Dec 4)** | **Final integration and Presentation** | * All features finalized * Full demo with live control and metric visualization * CSV outputs and plotted results * Final presentation slides and analysis completed |

**Midterm Presentation Goals (November 6)**

By November 6, you will demonstrate:

* Zephyr RTOS running on the Raspberry Pi Pico.
* Three functioning threads (high, medium, low) scheduled correctly.
* A concurrent “noise” load task.
* Real-time data for start-time jitter, execution time, and deadline misses.
* A USB shell interface enabling live configuration changes (priorities, timeslice, load).
* Initial analysis of scheduling behavior under different settings.

**Final Presentation Goals (December 4)**

By December 4, we will demonstrate:

* A functional RTOS scheduling visualizer that measures and reports:
* Start-time jitter
* Execution time
* Deadline misses
* Context switches
* CPU utilization per thread
* USB interrupt latency
* Complete USB shell control for live parameter tuning.
* A demonstration of scheduler behavior under variable conditions.
* An analysis of real-time performance using the collected metrics.

**Goal / Final Product**

The final goal is to deliver a RTOS Scheduling Visualizer that:

* Runs on Raspberry Pi Pico under Zephyr 4.2
* Measures all core scheduling metrics
* Offers real-time control via USB shell
* Produces CSV outputs for analysis and visualization
* Demonstrates scheduler performance under configurable load conditions