Zephyr RTOS Overview

Lightweight, secure, and scalable real-time operating system

- **Open Source and Scalable:** Zephyr RTOS is an open source real-time operating system designed for resource-constrained embedded devices, supporting architectures from 8-bit to 64-bit.

- **Modular and Configurable:** Highly modular architecture enables fine-tuned configuration tailored to specific hardware and use cases, promoting efficient memory and power use.

- **Built-in Security Features:** Zephyr incorporates features like stack protection, access control, and memory domain isolation, critical for safety- and security-sensitive applications.

- **Ecosystem and Tooling:** Backed by the Linux Foundation, Zephyr integrates with modern development workflows (e.g., West, CMake) and supports an active and growing community.

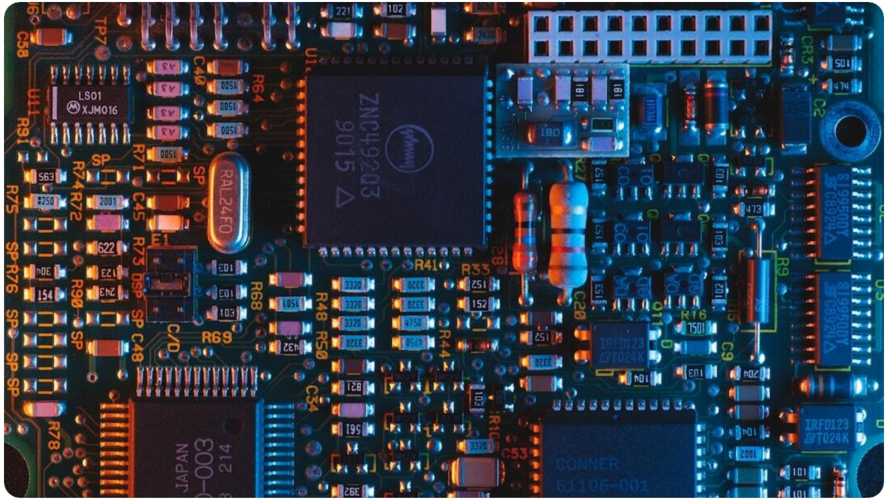

Photo by Umberto on Unsplash

Zephyr’s Unique Process Model

User-mode threads + memory domains instead of Linux-style processes

No Traditional Processes
Zephyr doesn’t implement Linux-style fork/exec processes. Instead, all applications are built as threads managed within a single address space.

Thread-Level Isolation
Each user thread can be assigned to a distinct memory domain, effectively isolating them like separate processes in traditional OSes.

Memory Domain Sharing
Inter-thread communication is achieved by mapping shared memory partitions into multiple domains or using kernel IPC mechanisms.

Lightweight and SMP-Aware
The architecture is optimized for embedded constraints and can leverage multicore scheduling on SMP-supported boards.

Thread Isolation with Memory Domains

Securely sandboxing threads in Zephyr RTOS

Memory Domain Definition
Zephyr allows the creation of k_mem_domain structures that define accessible memory partitions for user-mode threads.

Granular Partition Mapping
Each memory partition can be individually mapped into a domain, allowing fine-tuned control over what data each thread can access.

Domain-Thread Assignment
User threads are explicitly assigned to domains using k_mem_domain_add_thread(), enforcing access boundaries.

Enforcement via MMU/MPU
When hardware supports it, Zephyr uses MMU or MPU to enforce memory isolation at runtime between user threads.

Shared Ring Buffer for Inter-Thread Communication

Using shared memory partition for zero-copy data transfer

- **Zero-Copy Buffer Sharing:** Threads A and B exchange data using a statically allocated ring buffer placed in a shared memory partition, eliminating the need for copying.
- **Shared Partition Mapping:** The buffer's memory partition is mapped into both domains, enabling direct access for both threads to read/write safely.
- **Data Structure and Access:** Zephyr's ring_buf API provides a thread-safe circular buffer interface ideal for producer-consumer models in embedded applications.
- **Use Case Example:** Thread A acts as a producer writing incrementing values, while thread B consumes them, synchronized by a kernel mutex.

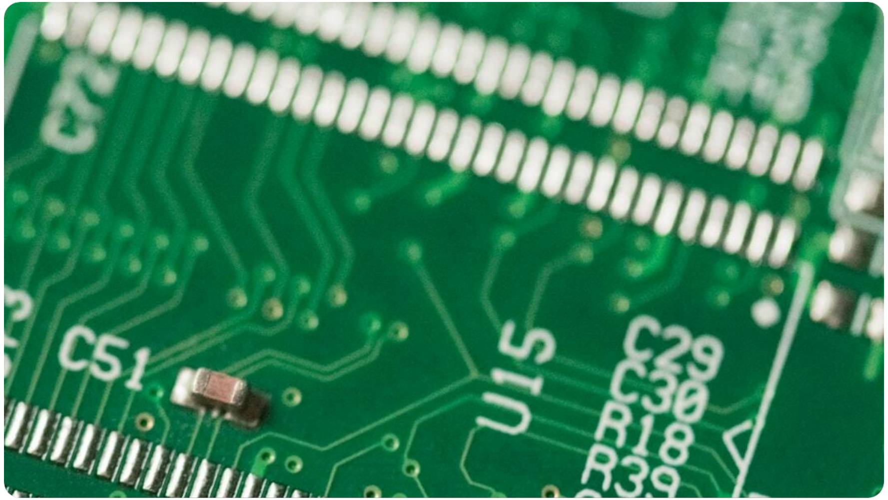

Photo by Tyler Daviaux on Unsplash

Synchronization with Kernel Mutex

Controlling access to shared buffers in user-mode threads

- **Kernel Object Usage:** A k_mutex is used to synchronize access to the shared ring buffer, ensuring atomic operations by user threads.
- **Access Control via Permissions:** Threads must be explicitly granted permission to access kernel objects using k_object_access_grant.
- **Safe Access Across Domains:** Though mutexes reside in kernel space, user threads from different memory domains can safely access them with proper grants.
- **Avoiding Race Conditions:** Mutex ensures mutual exclusion, preventing concurrent modification and ensuring data integrity in the ring buffer.

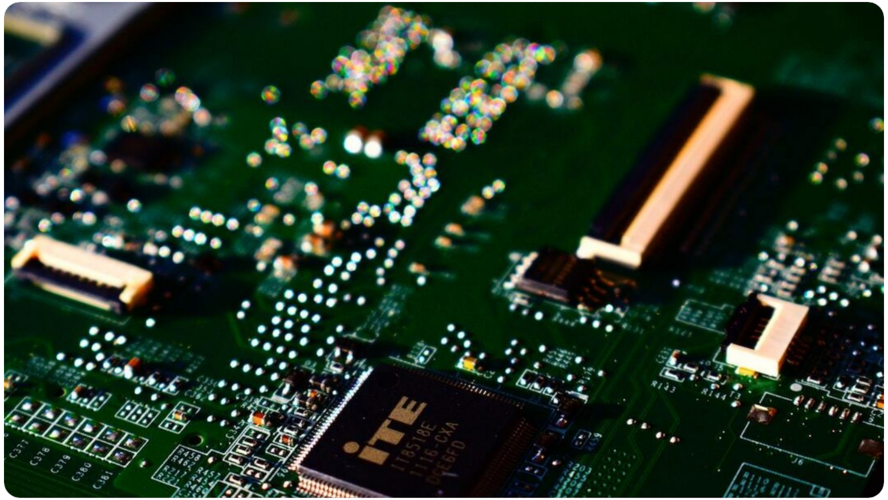

Photo by Lauren Nieuwland on Unsplash