Frame Counter Demo Using Timer/Counter in Zephyr

Integrating Timing and Counting for Frame-Based Applications

- **Purpose of the Demo:** Demonstrate how Zephyr’s timer and counter subsystems can be used together to implement a frame counter suitable for real-time embedded systems.
- **Key Components:** Utilizes Zephyr’s kernel timer (k_timer) and hardware counter drivers to synchronize frame timing with system ticks.
- **Applications:** Applicable in embedded graphics, motor control, sensor sampling, and signal processing where precise frame timing is critical.

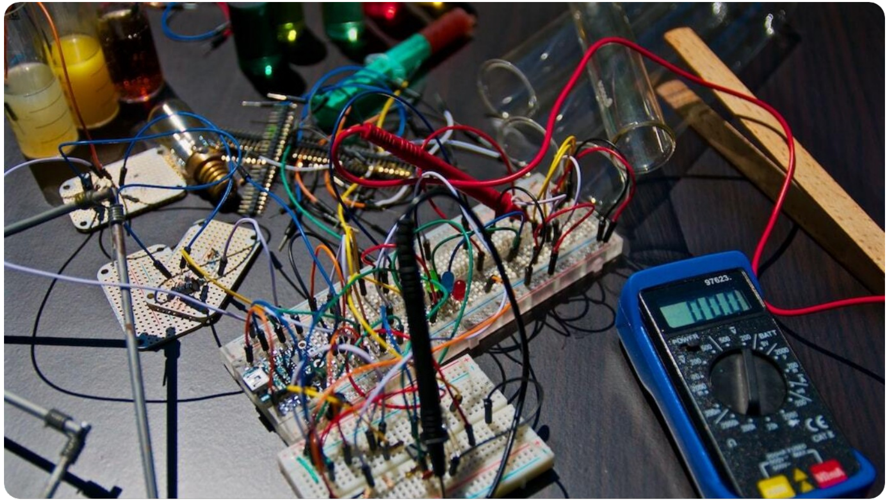

Photo by Nicolas Thomas on Unsplash

Background: Timer vs Counter Concepts

Understanding Timekeeping and Event Measurement in Embedded Systems

- **Timer Function:** A timer measures elapsed time using clock ticks, generating periodic events such as system heartbeats or task scheduling signals.
- **Counter Function:** A counter increments or decrements based on external events (e.g., pulses from sensors or I/O triggers), ideal for counting occurrences.
- **Key Difference:** Timers rely on internal clock sources, while counters depend on external stimuli — together enabling event timing and measurement precision.

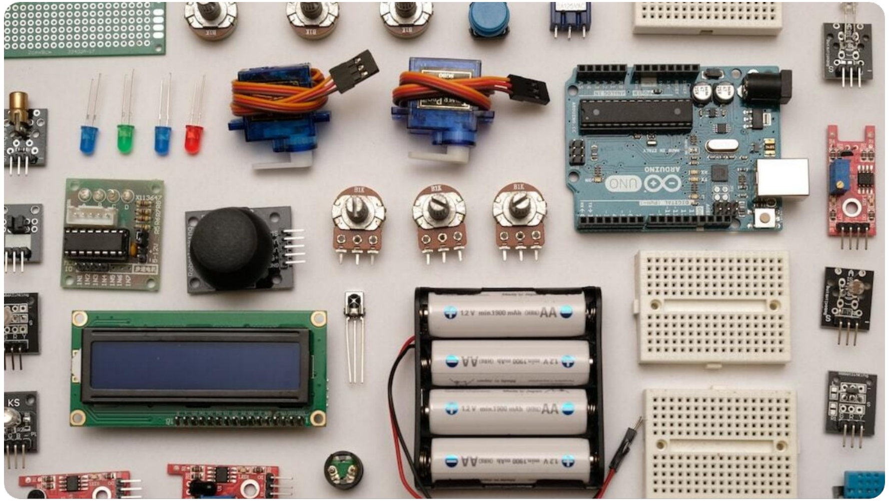

Photo by Robin Glauser on Unsplash

Zephyr RTOS: Timing Architecture Overview

Kernel, Clock, and Counter Interactions

- **System Clock Driver:** Provides a unified interface for tickless timekeeping, supporting multiple hardware timer backends across architectures.
- **Kernel Timing Layer:** Manages system ticks, time slicing, and scheduling with millisecond or microsecond precision, depending on hardware capabilities.
- **Counter Subsystem:** Abstracts hardware counters for events and periodic callbacks; interacts with the timer driver for unified timing control.

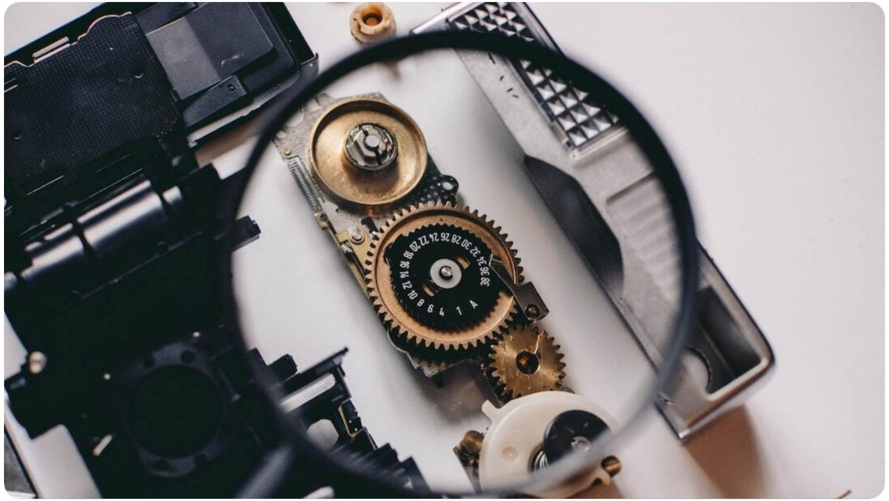

Photo by Shane Aldendorff on Unsplash

Zephyr Timer APIs (k_timer)

High-Level Software Timers for Periodic Operations

k_timer Structure
Defines start, stop, and expiry callbacks, providing an abstraction over kernel timing mechanisms for periodic or one-shot tasks.

Initialization & Control
Timers are initialized using k_timer_init(), started with k_timer_start(), and automatically trigger callbacks on expiration.

Use Cases
Ideal for periodic housekeeping tasks, LED blinking, watchdog refresh, or scheduling frame count updates in sync with system tick.

Zephyr Counter (Hardware) APIs

Low-Level Access to Hardware Counters and Timers

- **Counter Device Interface:** Provides unified functions like counter_start(), counter_stop(), and counter_read() across all supported hardware backends.
- **Alarm and Capture Features:** Supports alarms, compare events, and capture channels, enabling event-driven or interrupt-based time measurement.
- **Synchronization with Timer:** Counter API can synchronize with k_timer or other periodic triggers for hybrid timing and counting operations.

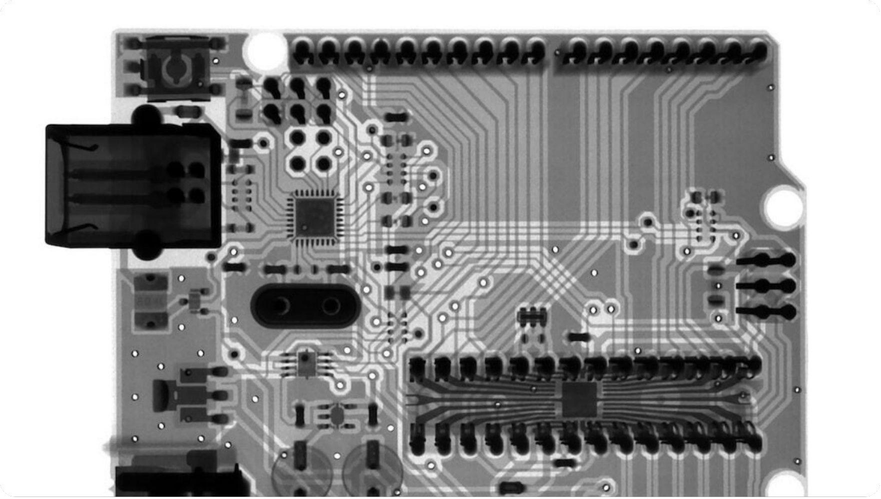

Photo by Mathew Schwartz on Unsplash

Design of Frame-Counter Demo in Zephyr

Architecture and Flow of Timer/Counter Integration

- **System Components:** Combines a periodic k_timer for frame pacing and a counter device for frame counting; synchronized via callback chaining.
- **Workflow:** The timer triggers frame updates at defined intervals, incrementing a counter value to track the number of frames processed.
- **Data Flow:** Counter value feeds into application logic or logging subsystem for real-time display and performance monitoring.

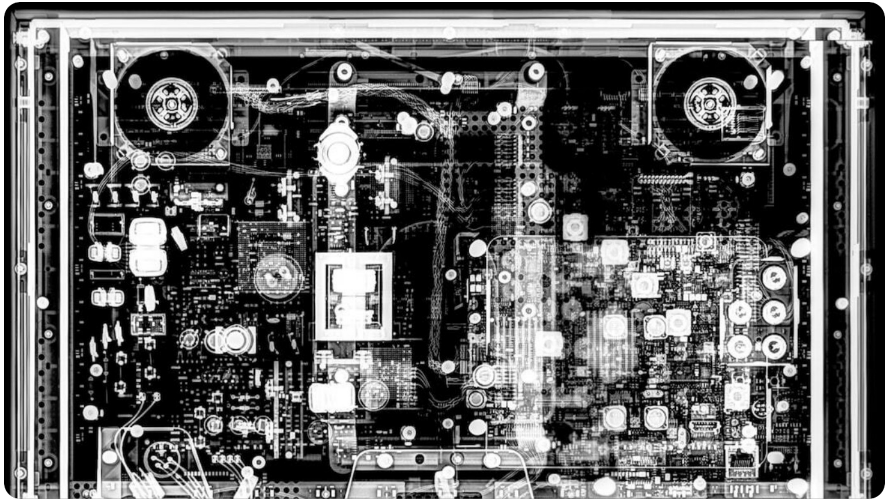

Photo by David Condrey on Unsplash

Implementation Details: Code Structure and Interrupts

Integrating Timers, Counters, and ISR Callbacks

Core Modules
Main.c initializes k_timer, binds to counter device, and defines ISR handlers for event updates and data logging.

Interrupt Handling
The counter’s alarm callback triggers on each event, updating the frame count safely using atomic operations.

Concurrency & Safety
Mutexes or atomic variables ensure thread safety when accessing frame data across timer callbacks and main loop.