Table of contents
Chapter 1

Introduction

The Zephyr OS is based on a small-footprint kernel designed for use on resource-constrained and embedded systems: from simple embedded environmental sensors and LED wearables to sophisticated embedded controllers, smart watches, and IoT wireless applications.

The Zephyr kernel supports multiple architectures, including:

- ARCv2 (EM and HS) and ARCv3 (HS6X)
- ARMv6-M, ARMv7-M, and ARMv8-M (Cortex-M)
- ARMv7-A and ARMv8-A (Cortex-A, 32- and 64-bit)
- ARMv7-R, ARMv8-R (Cortex-R, 32- and 64-bit)
- Intel x86 (32- and 64-bit)
- MIPS (MIPS32 Release 1 specification)
- NIOS II Gen 2
- RISC-V (32- and 64-bit)
- SPARC V8
- Tensilica Xtensa

The full list of supported boards based on these architectures can be found here.

In the context of the Zephyr OS, a subsystem refers to a logically distinct part of the operating system that handles specific functionality or provides certain services. Subsystems can include components such as networking, file systems, device driver classes, power management, and communication protocols, among others. Each subsystem is designed to be modular and can be configured, customized, and extended to meet the requirements of different embedded applications.

1.1 Licensing

Zephyr is permissively licensed using the Apache 2.0 license (as found in the LICENSE file in the project's GitHub repo). There are some imported or reused components of the Zephyr project that use other licensing, as described in Licensing of Zephyr Project components.

1.2 Distinguishing Features

Zephyr offers a large and ever growing number of features including:
Extensive suite of Kernel services
Zephyr offers a number of familiar services for development:

- **Multi-threading Services** for cooperative, priority-based, non-preemptive, and preemptive threads with optional round robin time-slicing. Includes POSIX pthreads compatible API support.
- **Interrupt Services** for compile-time registration of interrupt handlers.
- **Memory Allocation Services** for dynamic allocation and freeing of fixed-size or variable-size memory blocks.
- **Inter-thread Synchronization Services** for binary semaphores, counting semaphores, and mutex semaphores.
- **Inter-thread Data Passing Services** for basic message queues, enhanced message queues, and byte streams.
- **Power Management Services** such as overarching, application or policy-defined, System Power Management and fine-grained, driver-defined, Device Power Management.

Multiple Scheduling Algorithms
Zephyr provides a comprehensive set of thread scheduling choices:

- Cooperative and Preemptive Scheduling
- Earliest Deadline First (EDF)
- Meta IRQ scheduling implementing “interrupt bottom half” or “tasklet” behavior
- Timeslicing: Enables time slicing between preemptible threads of equal priority
- Multiple queuing strategies:
  - Simple linked-list ready queue
  - Red/black tree ready queue
  - Traditional multi-queue ready queue

Highly configurable / Modular for flexibility
Allows an application to incorporate only the capabilities it needs as it needs them, and to specify their quantity and size.

Cross Architecture
Supports a wide variety of supported boards with different CPU architectures and developer tools. Contributions have added support for an increasing number of SoCs, platforms, and drivers.

Memory Protection
Implements configurable architecture-specific stack-overflow protection, kernel object and device driver permission tracking, and thread isolation with thread-level memory protection on x86, ARM, and ARM architectures, userspace, and memory domains.

For platforms without MMU/MPU and memory constrained devices, supports combining application-specific code with a custom kernel to create a monolithic image that gets loaded and executed on a system's hardware. Both the application code and kernel code execute in a single shared address space.

Compile-time resource definition
Allows system resources to be defined at compile-time, which reduces code size and increases performance for resource-limited systems.

Optimized Device Driver Model
Provides a consistent device model for configuring the drivers that are part of the platform/system and a consistent model for initializing all the drivers configured into the system and allows the reuse of drivers across platforms that have common devices/IP blocks.
Devicetree Support
Use of devicetree to describe hardware. Information from devicetree is used to create the application image.

Native Networking Stack supporting multiple protocols
Networking support is fully featured and optimized, including LwM2M and BSD sockets compatible support. OpenThread support (on Nordic chipsets) is also provided - a mesh network designed to securely and reliably connect hundreds of products around the home.

Bluetooth Low Energy 5.0 support
Bluetooth 5.0 compliant (ESR10) and Bluetooth Low Energy Controller support (LE Link Layer). Includes Bluetooth Mesh and a Bluetooth qualification-ready Bluetooth controller.

- Generic Access Profile (GAP) with all possible LE roles
- Generic Attribute Profile (GATT)
- Pairing support, including the Secure Connections feature from Bluetooth 4.2
- Clean HCI driver abstraction
- Raw HCI interface to run Zephyr as a Controller instead of a full Host stack
- Verified with multiple popular controllers
- Highly configurable

Mesh Support:
- Relay, Friend Node, Low-Power Node (LPN) and GATT Proxy features
- Both Provisioning bearers supported (PB-ADV & PB-GATT)
- Highly configurable, fitting in devices with at least 16k RAM

Native Linux, macOS, and Windows Development
A command-line CMake build environment runs on popular developer OS systems. A native port (native_sim) lets you build and run Zephyr as a native application on Linux, aiding development and testing.

Virtual File System Interface with ext2, FatFs, and LittleFS Support
ext2, LittleFS and FatFS support; FCB (Flash Circular Buffer) for memory constrained applications.

Powerful multi-backend logging Framework
Support for log filtering, object dumping, panic mode, multiple backends (memory, networking, filesystem, console, ...) and integration with the shell subsystem.

User friendly and full-featured Shell interface
A multi-instance shell subsystem with user-friendly features such as autocompletion, wildcards, coloring, metakeys (arrows, backspace, ctrl+u, etc.) and history. Support for static commands and dynamic sub-commands.

Settings on non-volatile storage
The settings subsystem gives modules a way to store persistent per-device configuration and runtime state. Settings items are stored as key-value pair strings.

Non-volatile storage (NVS)
NVS allows storage of binary blobs, strings, integers, longs, and any combination of these.

Native port
Native sim allows running Zephyr as a Linux application with support for various subsystems and networking.
1.3 Community Support

Community support is provided via mailing lists and Discord; see the Resources below for details.

1.4 Resources

Here's a quick summary of resources to help you find your way around:

1.4.1 Getting Started

- Zephyr Documentation
- Getting Started Guide
- Tips when asking for help
- Code samples

1.4.2 Code and Development

- Source Code Repository
- Releases
- Contribution Guide

1.4.3 Community and Support

- Discord Server for real-time community discussions
- User mailing list (users@lists.zephyrproject.org)
- Developer mailing list (devel@lists.zephyrproject.org)
- Other project mailing lists
- Project Wiki

1.4.4 Issue Tracking and Security

- GitHub Issues
- Security documentation
- Security Advisories Repository
- Report security vulnerabilities at vulnerabilities@zephyrproject.org

1.4.5 Additional Resources

- Zephyr Project Website
- Zephyr Tech Talks

1.5 Fundamental Terms and Concepts

See Glossary of Terms