The Zephyr Training

Week 1

Foundations, Environment, and First Hardware Interaction

Welcome to the Zephyr Project. This week introduces what Zephyr is and how to develop projects with it in a practical way. You will set up the development environment using Docker, get to know the training board, and understand Zephyr's basic project structure and configuration system.

At the same time, you will begin interacting with real hardware using GPIOs. Through this, you will be introduced to Kconfig, DeviceTree, and overlay files, learning how hardware description connects directly to software configuration.

Session 1 Introduction and Project Setup

- What is the Zephyr Project and why it is more than just a kernel (hardware-agnostic architecture and ecosystem overview)
- Introduction to the development board used during the training
- Introduction to the development tools (Docker, Segger SystemView, Segger Ozone, DevContainers)
- File structure of the base project and overview of Zephyr directories

Session 2 GPIOs and dt_spec

- Configuring a GPIO pin using DeviceTree and referencing a DTC node
- Understanding binding properties associated with GPIO nodes
- Using macros such as DEVICE_DT_GET, DT_ALIAS, and DT_NODELABEL
- Understanding the compatible property and binding files
- Writing a program using gpio_*_dt() and GPIO_DT_SPEC_GET
- Defining a node and child nodes using the gpio-leds and gpio-keys bindings

Week 2

Serial Communication and Interrupts

We continue with our second driver — and one of the most widely used peripherals in embedded systems — the serial port (UART).

This week, we will configure the UART driver and learn how to transmit and receive data, both with and without interrupts. In addition, you will learn how to configure interrupts manually and understand how the Zephyr Project uses them internally within its drivers.

Session 3 UART

- Serial port configuration and review of its DeviceTree nodes and bindings
- Pin configuration with the pin control driver (pinctrl)
- Examples of UART functions using polling
- Examples of UART functions using interrupts and DMA

Session 4 Interrupts

- Defining interrupts using the IRQ_CONNECT macro and irq_enable
- Understanding the _isr_table_entry structure and the vector table in isr_tables.c
- Direct ISRs and zero-latency interrupts
- Dynamic interrupts using irq_connect_dynamic and irq_disconnect_dynamic
- Examining GPIO and UART interrupts

Week 3

Console, Logging, and Build Fundamentals

You will explore Zephyr's console and logging system, including chosen nodes in the board's DeviceTree, log configuration, and routing output through tools like Segger RTT and Ozone for effective debugging.

We then examine the build and configuration system: project manifests, workspace structure, default board settings, and the relationship between DeviceTree and Kconfig. You will also learn how CMake manages build phases and how to troubleshoot common configuration issues.

Session 5 Console Logs

- Zephyr chosen nodes and where they are defined in the board's DeviceTree
- Enabling and configuring the logging system, with practical examples
- Configuring printk and log output using Segger RTT and Ozone

Session 6 Build and Configuration

- Project manifest and introduction to external modules
- Understanding workspace topologies
- Default board configuration in the Zephyr Project (DeviceTree and Kconfig)
- Build system (CMake) build and configuration phases
- Debugging DeviceTree and Kconfig (common errors, tips, and tricks)

Week 4

Kernel Fundamentals - Threads and Scheduling

We begin working directly with the operating system kernel. In this week, you will learn how to define threads and configure their parameters, such as priority, stack size, and flags.

We will also explore thread control functions and learn how to tune the Zephyr kernel to manage time slicing, preemption, and cooperative scheduling.

Bonus: Configuring SystemView to trace threads and events.

Session 7 Threads

- Defining and configuring threads at build time using K_THREAD_DEFINE
- Understanding the different thread states
- Thread control and management (k_sleep, priorities, suspension, k_yield)
- System threads (main and idle) and their configuration through Kconfig
- Spawning new threads at runtime using k_thread_create

Session 8 Kernel

- Scheduler behavior: preemptive vs. cooperative threads
- Time slicing configuration and kernel tuning options
- Kernel configuration through Kconfig (stack sizes, priorities, scheduling options)

Week 5

Inter-Thread Communication and Synchronization

A complete program is composed of multiple tasks, and they must communicate with each other effectively. In this week, you will learn about the different inter-thread communication mechanisms provided by the Zephyr Project, including queues, FIFOs, and message queues.

In addition, we will cover synchronization techniques to ensure safe access to shared resources. You will work with mutexes and semaphores to prevent race conditions and guarantee predictable system behavior.

Session 9 Data Passing

- Zephyr queue-based data passing mechanisms (FIFOs and LIFOs)
- Message queues (data passing by copy)
- Passing data to and from interrupts
- Other data passing mechanisms (mailboxes, pipes, and stacks)

Session 10 Thread Synchronization

- Mutexes: protecting shared resources and priority inheritance
- Semaphores: counting and binary synchronization
- Synchronization between producing and consuming threads or ISRs
- Avoiding race conditions, deadlocks, and priority inversion

Week 6

Deferred Execution and Advanced Communication

In this week, we focus on mechanisms that allow you to defer work and manage time-based events efficiently inside the Zephyr Project. You will learn how to offload processing from interrupt context using work queues.

Zbus is an efficient communication mechanism that supports one-to-one and many-to-many patterns in a simple way. You will learn its core concepts, including channels and observers, as well as the four listening modes and when to use them.

Session 11 Work Queues

- Concept of deferred work and why it is needed
- Defining and initializing k_work and k_work_delayable
- Submitting work from threads and interrupts
- Creating and managing custom work queue threads

Session 12 Zbus

- Overview of Zbus architecture within the Zephyr Project
- Defining and declaring Zbus channels and message types
- Observers and the basic publish/subscribe workflow
- Listener types: listener, async listener, subscriber, and message subscriber
- Execution context and priority handling of observers

Week 7

ADC and Sensor Interface

In this week, we focus on analog data acquisition and high-level sensor integration within the Zephyr Project. You will learn how to configure and use the ADC driver to read analog signals from hardware, and then move one layer higher by working with Zephyr’s generic sensor interface. This will help you understand the difference between low-level peripheral access and standardized driver abstractions.

Session 13 ADC

- ADC configuration using DeviceTree and Kconfig
- Setting up ADC channels and reading raw samples
- Single-shot vs. multi-sampling conversions
- Working with ADC sequences and buffers
- Handling ADC interrupts and asynchronous reads

Session 14 Sensors

- Introduction to Zephyr's generic sensor API
- Differences between direct ADC usage and the sensor abstraction layer
- Fetching samples with sensor_sample_fetch()
- Reading processed values using sensor_channel_get()
- DeviceTree configuration of sensor nodes and bindings

Week 8

I2C and RTIO

This week focuses on communication with external peripherals and efficient data handling mechanisms in the Zephyr Project. You will learn how to configure and use the I2C bus to communicate with external devices such as sensors, and then explore RTIO (Real-Time I/O), a modern framework for asynchronous and high-performance I/O operations.

Session 15 I2C

- Overview of the I2C protocol and typical use cases
- DeviceTree configuration of I2C controllers and child devices
- Understanding I2C bindings and the compatible property
- Using i2c_read(), i2c_write(), and i2c_write_read()
- Interrupt-driven vs. polling-based I2C transfers

Session 16 RTIO

- Introduction to RTIO and its purpose
- Core RTIO data structures and submission/completion queues
- Asynchronous I/O concepts in Zephyr
- Integrating RTIO with drivers (e.g., ADC, I2C, SPI)
- Performance considerations and concurrency model
- Comparison between traditional blocking APIs and RTIO

Week 9

SPI and the RTIO-Based Sensor Interface

In this week, we continue exploring peripheral communication and modern driver abstractions in the Zephyr Project. First, we will study the SPI protocol and how to configure and use it in Zephyr. Then, we will examine the newer sensor interface built on top of RTIO, which enables asynchronous and more efficient data acquisition from sensors

Session 17 SPI

- DeviceTree configuration of SPI controllers and connected devices
- Understanding SPI bindings and the compatible property
- Configuring `spi_config` structures and using the APIs `spi_read_dt()`, `spi_write_dt()`, and `spi_transceive_dt()`
- Handling chip select (hardware vs. GPIO-controlled CS)
- Interrupt-driven vs. blocking SPI transfers

Session 18 Sensors-Part 2

- Differences between the traditional sensor API and the RTIO-based approach
- Asynchronous sensor data acquisition using RTIO
- Submission and completion queue workflow
- Integration with work queues and threads
- Performance and scalability considerations

Week 10

Device Drivers from Scratch

This is the moment to build your first device driver completely from scratch using the device driver model of the Zephyr Project.

We will start with something simple, such as an LED driver, to understand the fundamentals: DeviceTree bindings, configuration structures, initialization macros, and API definitions. Once the foundation is clear, we will move on to more advanced examples, including an external sensor driver.

By the end of this week, you will clearly understand how to tie together DeviceTree, Kconfig, CMake, and the driver API to create fully integrated Zephyr drivers.

Session 19 Your First Driver

- Review of the Zephyr device driver model
- Defining `DT_DRV_COMPAT` and DeviceTree bindings
- Creating configuration and runtime data structures and implementing initialization functions
- Using `DEVICE_DT_INST_DEFINE()`
- Defining a driver API structure with function pointers
- Adding Kconfig and CMake integration

Session 20 Device Drivers (Sensor)

- Review of the sensor driver structure in Zephyr
- DeviceTree bindings and configuration for external sensors
- Implementing the sensor API (`sample_fetch`, `channel_get`, triggers)
- Managing runtime data and private driver context
- Interrupt-driven vs. polling-based sensor designs

  Integrating deferred work (work queues) when needed

Week 11

Defining Your Own Board

This week is entirely dedicated to creating and configuring your own custom board in the Zephyr Project.

You will learn how boards are structured inside Zephyr, how hardware description is organized, and how to properly define a new board so it can be built, configured, and maintained like any official Zephyr target. By the end of this week, you will be able to create a fully functional custom board definition from scratch.

Session 21 Board definition I

- Overview of the Zephyr board directory structure
- Creating a new board folder and required files
- Writing the board `.dts` file and including the SoC
- Defining board-specific peripherals and aliases
- Creating the board `Kconfig` and `Kconfig.defconfig`

Session 22 Board Definition II

- Adding the board `CMakeLists.txt`
- Creating board overlays for different hardware revisions
- Flash and memory partition configuration
- Board documentation and YAML metadata file
- Best practices for upstream-ready board definitions

Week 12

Startup Sequence and Contributing to Zephyr

In this final week, we step back and analyze what really happens behind the scenes. Where are drivers initialized? What occurs before the first thread runs, and what happens afterward?

We have reached the final part of the training. Before we wrap up and recap everything we have learned, it is time to understand how to contribute to the Zephyr Project — and why our contribution truly matters.

Session 23 Startup Sequence

- What happens immediately after the reset vector
- Early hardware initialization, low-level setup, and kernel initialization sequence
- Device initialization levels and driver ordering
- Creation of system threads (main, idle, system work queue)
- System state after the scheduler starts

Session 24 Contributing to Zephyr

- Overview of the Zephyr open-source governance model
- Reviewing contribution guidelines and coding standards
- Setting up the development environment for contribution
- Writing and running tests before submitting patches
- Creating and submitting a demonstration pull request