oRT5 - Zephyr RTOS Programming

From Theory to Practice

Objectives

- Learn how to develop, configure, debug and trace Zephyr applications
- Devicetree and Kconfig usage and development
- Zephyr real time multitasking overview
- Using west and writing west manifest
- Understand the Zephyr kernel Services and ecosystem
- Learn communication and synchronization mechanisms
- Understand Zephyr memory management and data structures
- Understand User mode and kernel mode separation
- Writing a device tree and developing Zephyr drivers
- Using and integrating common Zephyr subsystems

Course Environment

- Theoretical course
  - PDF course material (in English).
  - Course dispensed using the Teams video-conferencing system.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance through the Teams video-conferencing system.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One Online Linux PC per trainee for the practical activities.
  - The trainer has access to trainees' Online PCs for technical and pedagogical assistance.
  - Eclipse environment and GCC compiler.
  - QEMU Emulated board or physical board connected to the online PC (depending on the course).
  - Some Labs may be completed between sessions and are checked by the trainer on the next session.
- Downloadable preconfigured virtual machine for post-course practical activities
- At the start of each session the trainer will interact with the trainees to ensure the course fits their expectations and correct if needed

Prerequisites

- Good C programming skills (see our oL2 - C Language for Embedded MCUscourse)

Duration

- Total: 30 hours
- 5 sessions, 6 hours each (excluding break time)
- From 40% to 50% of training time is devoted to practical activities
- Some Labs may be completed between sessions and are checked by the trainer on the next session

Target Audience

- Any embedded systems engineer or technician with the above prerequisites.

Evaluation modalities

- The prerequisites indicated above are assessed before the training by the technical supervision of the traineein his company, or by the trainee himself in the exceptional case of an individual trainee.
- Trainee progress is assessed in two different ways, depending on the course:
  - For courses lending themselves to practical exercises, the results of the exercises are checked by the trainer while, if necessary, helping trainees to carry them out by providing additional details.
  - Quizzes are offered at the end of sections that do not include practical exercises to verifythat the trainees have assimilated the points presented
- At the end of the training, each trainee receives a certificate attesting that they have successfully completed the course.
  - In the event of a problem, discovered during the course, due to a lack of prerequisites by the trainee a different or additional training is offered to them, generally to reinforce their prerequisites,in agreement with their company manager if applicable.

Plan

Day 1

Introduction to Zephyr

- Zephyr Ecosystem
- Why use Zephyr
  - Drivers' API abstraction
  - Hardware-agnostic configuration
  - Modular Architecture
- Host tools dependencies
- Install and Use Zephyr

Zephyr Build System

- Overview
  - West
  - CMake
  - Toolchains and Zephyr SDK
  - VSCode Configuration
- Application components and structure
  - Application
  - Modules
  - West workspace
- West
  - Why west?
  - Usage
  - Manifest
  - Topologies
  - Commands

Exercise: *Getting started with Zephyr and using west*
Exercise: *Writing a custom west manifest*

Configure Zephyr

- Overview
- Kconfig
  - Default configuration
  - Interactive configuration tools
  - Config fragments
- Devicetree
  - Syntax

  - Standard properties
  - Device Tree node structure
  - Device Tree bindings
  - Initial devicetree source
  - Access devicetree from source code
  - Overlays
  - Best practices
- Snippets

Exercise: *Write a device tree overlay*

Zephyr Fundamentals

- Operations without Threads
- Common subsytems
  - GPIOs
  - DeviceTree specification structures dt_spec
- Utilities
- Preprocessor meta-programming macros
- Data Structures
  - Single-linked List
  - Double-linked List
  - Ring Buffers

Exercise: *Using X-Macros in Zephyr and understanding CONTAINER_OF*

Day 2

Thread Management

- Thread Fundamentals
  - Thread Control Block
  - Creating Threads
  - Threads Priorities
  - Thread States
- Main and Idle Threads
- System Initialization
- Delays and timeout
- Kernel Structures
  - Simple linked-list ready queue
  - Red/black tree ready queue
  - Traditional multi-queue ready queue
- Thread Custom Data

Exercise: *Create and manage threads*
Exercise: *Create periodic threads*

Tracing and logging

- Runtime Statistics
- Scheduling Traces
  - User-Defined Tracing
  - Percepio Tracealyzer

Exercise: *Create config overlay for visual trace diagnostics using Tracealyzer*

Memory Management in Zephyr

- Memory Overview
- Dynamic memory managers
  - K_heap
  - System heap
  - Memory Slabs

    - Memory Blocks
- Heap Listeners
- Thread Resource Pools
- RAM/ROM reports
- Stack information
    - Stack analysis
    - Puncover
    - High watermark
- Stack overflow detection

Exercise: *Understand dynamic memory allocation in Zephyr*
Exercise: *Display threads information and detect stack overflow*

Day 3

User Mode

- Overview
- Memory Domains
    - Partitions
    - Logical apps
- Syscalls
    - Kernel objects
    - Permissions

Resource Management and Synchronization

- Mutual Exclusion
- Mutexes
- Gatekeeper threads
- Critical Sections
- Atomic
- SpinLocks
- Semaphores
- Events
- Polling

Exercise: *The producer-consumer problem, synchronize and avoid concurrent access problems*
Exercise: *Understanding event bit group by synchronizing several threads*

Inter-Thread Communication

- Data passing
    - Message Queues
    - Queues (FIFOs & LIFOs)
    - Mailboxes
    - Pipes
    - Stacks
- Zephyr Bus (Zbus)
    - Zbus overview
    - Elements
    - Usage

Exercise: *Create a print gatekeeper thread using message queue*
Exercise: *Synchronous communication using mailboxes*

Day 4

Interrupt Management

- Threads and Interrupts

- Interrupts in zephyr
- Interrupts on ARM Cortex-M
- Handler thread
- Queue within an ISR
- Workqueue Threads

Exercise: *Understand how to wait on multiple events and interrupt safe APIs*
Exercise: *Understand how to pass data using Queues from an interrupt to a thread*
Exercise: *Create and submit work items from interrupts to custom WorkQueue*

Software Timers

- Timers
  - Defining a Timer
  - Using a Timer Expiry Function
- Timer types
  - One-shot timers
  - Auto-reload timers
- Timer Commands

Exercise: *Understand the use of one-shot and auto-reload timers*

Modules

- Why to use modules?
- Module structure
- Out-of-tree module
- Module's YAML
- Module CMakeLists.txt

Exercise: *Create a basic module*

Writing Kconfig symbols

- Advantages
- Kconfig Options in Zephyr RTOS
- Configuration System
- Writing custom Kconfig Options
- Kconfig extension
- Using Kconfigs

Exercise: *Create and configure a module that uses custom Kconfig options*

Day 5

Device Driver Architecture

- Zephyr Device Driver Model
  - Overview and its role
- Standard Drivers
  - The struct device
  - Subsystems
  - Device definition
- API Extensions
- Devices allocation and initialization
- Using drivers in application
- Initialization Levels
  - Dependencies between device drivers

Exercise: *Create a driver that respects the Zephyr Device Driver Model and define devices*

Device Trees in Zephyr

- Overview of Device Tree (DT) and its role in Zephyr

- Device Tree VS Kconfig
- Device Tree node structure
- Device Tree bindings
- Overlay and yaml files
- APIs to access device tree properties
- Write device drivers using device tree APIs
- Device Tree in Zephyr VS Linux
- Adding In-Tree Code to Zephyr Source Code
- Common properties
  - compatible
  - reg
  - interrupts

Exercise: *Create a driver that uses custom device tree and Kconfig*
Exercise: *Writing in-tree drivers*

Power Management

- Overview
- System Power Management
- Device Power Management
  - System-Managed
  - Runtime
- Power domains

Exercise: *Write a driver compatible with power management subsystem*

Renseignements pratiques

Inquiry : 30 hours