NR3 - NXP + FreeRTOS + West

FreeRTOS with NXP MCUXpresso

Objectives

- Understand MCUXpresso SDK (MCUXSDK) structure
- Manage multi-repository projects using Zephyr West
- Use Kconfig and prj.conf for configuration
- Create and integrate custom boards
- Extend projects with FreeRTOS
- Get an overview of Cortex-M architecture
  - Discover the concepts of real time multitasking
  - Understand Real Time constraints
  - Understand the FreeRTOS architecture
  - Discover the various FreeRTOS services and APIs
  - Learn how to develop, debug and trace FreeRTOS applications
- Best practices for large MCUXpresso/FreeRTOS projects

Prerequisite

- C Language knowledge (see for example our L2 training course)
- Familiarity with Git and command-line tools

Course environment

- Theoretical course
  - PDF course material (in English)
  - The trainer to answer trainees' questions during the training and provide technical and pedagogical assistance
- Practical activities
  - Practical activities represent from 40% to 50% of course duration
  - Example code, labs and solutions
  - NXP MCUX or simulated IMXRT board using Zazu Simulator

Environnement du cours

- Cours théorique
  - Support de cours au format PDF (en anglais) et une version imprimée lors des sessions en présentiel
  - Cours dispensé via le système de visioconférence Teams (si à distance)
  - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique
- Activités pratiques
  - Les activités pratiques représentent de 40% à 50% de la durée du cours
  - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
  - Exemples de code, exercices et solutions
  - Pour les formations à distance:
    - Un PC Linux en ligne par stagiaire pour les activités pratiques, avec tous les logiciels nécessaires préinstallés.
    - Le formateur a accès aux PC en ligne des stagiaires pour l'assistance technique et pédagogique
    - Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante.
  - Pour les formations en présentiel::
    - Un PC (Linux ou Windows) pour les activités pratiques avec, si approprié, une carte cible embarquée.
    - Un PC par binôme de stagiaires s'il y a plus de 6 stagiaires.
  - Pour les formations sur site:

    - Un manuel d'installation est fourni pour permettre de préinstaller les logiciels nécessaires.
    - Le formateur vient avec les cartes cible nécessaires (et les remporte à la fin de la formation).
- Une machine virtuelle préconfigurée téléchargeable pour refaire les activités pratiques après le cours
- Au début de chaque session (demi-journée en présentiel) une période est réservée à une interaction avec les stagiaires pour s'assurer que le cours répond à leurs attentes et l'adapter si nécessaire

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus.

Course Outline

First day

Introduction to MCUXpresso SDK

- SDK structure and components
- Toolchains, CMake and Ninja integration
- Application structure and examples

West Tool

- Overview
- Application components and structure
    - Application
    - Modules
    - West workspace
- Why West? Problems solved
- West as a meta-tool: repository + commands
- Alternatives (git submodules, repo) and limitations
- West
    - West structure
    - Using west
    - West manifest
    - West commands
- West topologies
- Anatomy of west.yml
- Specific commands and common extensions
    - Init, update, list, config
    - Build, debug, attach, flash
    - Other common commands
- Extending West with custom commands

Exercise : Getting started with West and MCUXpresso SDK
Exercise : Create a custom workspace manifest while importing only required projects

Development Environment

- Setting up host tools (Git, Python, CMake, Ninja)
- Integrating LinkServer, Jlink and other debuggers
- Debugging workflow with GDB
- VSCode integration (tasks, debug sessions)
- MCUXpresso for VSCode

Exercise : Build, flash and debug using command line and customize IDE

MCUXpresso Config Tools

- Overview of the configuration tool suite (Pins, Clocks, Peripherals, Device settings)
- How Config Tools integrate with MCUXpresso SDK and West builds
- Generating initialization code (pin_mux.c/h, clock_config.c/h, peripheral setup)
- Using the graphical interface to configure GPIO, UART, and system clocks
- Exporting configuration files and re-integrating them into applications
- Limitations and best practices when combining with Kconfig/prj.conf

Exercise : Customize existing boards

Customization and Extensions

- Custom manifests for minimal projects
- Writing custom West commands
- Modifying in-tree applications (LED blinky)
- Freestanding applications outside the SDK

Exercise : Extend west commands
Exercise : Create a custom freestanding application

Second day

Integration and Analysis

- Adding FreeRTOS using West
- Multicore projects with Sysbuild
- SPDX analysis and compliance check
- Memory footprint and Puncover analysis

Exercise : Extend the workflow with FreeRTOS and advanced tools
Exercise : Using west memory analysis features

Kconfig and Project Configuration

- Configuration phase in West/CMake
- Kconfig framework:
  - Enabling/disabling global features
  - Tuning and conditional compilation
  - Default values and symbol dependencies
- Role of prj.conf and fragments
- Interactive configuration (menuconfig, guiconfig)
- Generated config files: .config, mcux_config.h
- Writing new Kconfig entries (symbols, menus, defaults)
- Limitations and best practices
- MCUXpresso SDK specifics (custom prefixes, no CONFIG_ macros)

Exercise : Customize prj.conf
Exercise : Create and use custom kconfig options

Developing Custom Boards

- Board Architecture Overview
- Structure and components of a board port
- Creating a New Board Definition
- Configuring custom boards
- Board debuggers
- Linker Script
- Integrating the custom Board into the SDK

Exercise : Write a custom board

External MCUX Modules

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

Exercise : Create a custom library module

Cortex-M resources used by FreeRTOS

- Cortex-M Architecture Overview
  - Two stacks pointers
  - Different Running-modes and Privileged Levels
  - MPU Overview
  - Systick Timer Description
  - ARMv8-M evolutions
- Exception / Interrupt Mechanism Overview
  - Interrupt entry and return Overview
  - SVC / PendSV / Systick Interrupt Presentation
- Developing with the IDE

Exercise : Interrupt Management on Cortex-M

Third day

Introduction to Real Time

- Base real time concepts
- The Real Time constraints
- Multi-task and real time

Element of a real time system

- Tasks and Task Descriptors
- Context Switch
- Task Scheduling and Preemption
  - Tick based or tickless scheduling
- Scheduling systems and schedulability proof
- Scheduling through FreeRTOS

Exercise : Implement a Context Switch routine

FreeRTOS Task Management

- The Task life-cycle
- Creating Tasks
- Task Priorities
- Task States
- The idle task
- Delays
- Changing Task Priority
- Deleting Tasks
- Suspending Tasks
- Kernel Structures
- Thread Local Storage
- Kernel Interrupts on Cortex-M
- Scheduling Traces

- Visual trace diagnostics using Tracealyzer

Exercise : Managing tasks

Fourth day

Memory Management in FreeRTOS

- Memory management algorithms
- FreeRTOS-provided memory allocation schemes
  - Allocate-only scheme
  - Best-fit without coalescing
  - Thread-safe default malloc
- Checking remaining free memory
- Adding an application-specific memory allocator
- Memory management errors
- Stack monitoring

Exercise : Enhance the memory manager for memory error detection

Exercise : Detect stack overflow

Resource Management with FreeRTOS

- Critical sections
  - Critical sections
  - Suspending (locking) the scheduler
- Mutual Exclusion
  - Spinlocks and interrupt masking
  - Mutex or Semaphore
  - Recursive or not recursive mutexes
  - Priority inversion problem
  - Priority inheritance (the automatic answer)
  - Priority ceiling (the design centric answer)
- Gatekeeper tasks

Exercise : Implement mutual exclusion between two tasks

Synchronization Primitives

- Introduction
  - Waiting and waking up tasks
  - Semaphores
  - Events
  - Mailboxes
- FreeRTOS Binary Semaphores
- FreeRTOS Queue Management
  - Creation
  - Sending on a queue
  - Receiving from a queue
  - Data management
  - Sending compound types
  - Transferring large data
- Queue sets
- Event Groups
- Task Notifications

Exercise : Synchronizing a task with another one through queues

Exercise : Synchronizing a task with another one through binary semaphores

Fifth day

Parallelism Problems Solutions

- Parallel programming problems
  - Uncontrolled parallel access
  - Deadlocks
  - Livelocks
  - Starvation

Exercise : The producer-consumer problem, illustrating (and avoiding) concurrent access problems
Exercise : The philosophers' dinner problem, illustrating (and avoiding) deadlock, livelock and starvation
Exercise : The readers-writer problem, illustrating complex concurrent access solving

Interrupt Management

- Need for interrupts in a real time system
  - Software Interrupt
  - Time Interrupts
  - Device Interrupts
- Level or Edge interrupts
- Hardware and Software acknowledge
- Interrupt vectoring
- Interrupts and scheduling
- Deferred interrupt processing through FreeRTOS
  - Tasks with interrupt synchronization
  - Using semaphores within an ISR
  - Counting semaphores
  - Using queues within an ISR
- FreeRTOS interrupt processing
  - Writing ISRs in C
  - Interrupt safe functions
  - Interrupt nesting

Exercise : Synchronize Interrupts with tasks

Software Timer

- The Timer Daemon Task
- Timer Configuration
- One-shot / Auto-reload Timer
- Software Timer API

Exercise : Use Software Timers