

## Building and installing an embedded and real-time Linux platform

### Objectives

- Understanding the architecture of the Linux system
- Create and use a cross-development toolchain
- Learn how to install Linux on your hardware and create a BSP
- Explore the Linux system architecture
  - Booting Linux
  - Initializing the system
- Install existing packages on the target
- Learn how to install Linux on flash chips

*Some labs are conducted using the [System Workbench for LinuxIDE](#).*

### Course Environment

- Theoretical course
  - PDF course material (in English) supplemented by a printed version.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One PC (Linux ou Windows) for the practical activities with, if appropriate, a target board.
    - ▶ One PC for two trainees when there are more than 6 trainees.
  - For onsite trainings:
    - ▶ An installation and test manual is provided to allow preinstallation of the needed software.
    - ▶ The trainer come with target boards if needed during the practical activities (and bring them back at the end of the course).
- 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

### Prerequisite

- Good C programming skills
- Knowledge of Linux user programming (see our [D0 - Linux user mode programming](#) course)
- Preferably knowledge of Linux kernel and driver programming (see our [D3 - Linux Drivers](#) course)

### 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 trainee in 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 verify that 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

#### FIRST DAY

##### **Linux overview**

- Linux
  - History
  - Version management
- Linux architecture and modularity
- Linux system components
- The various licenses used by Linux (GPL, LGPL, etc)

##### **Cross compiling toolchains**

- Pre-compiled toolchains
- Toolchain generation tools
  - Crosstool-ng
  - Buildroot
- Manual toolchain compilation

*Exercise: Creating a toolchain with Crosstool-ng*

##### **Linux tools for embedded systems**

- Bootloaders (UBoot, Redboot, barebox)
- C libraries (glibc, eglibc, uClibc)
- Embedded GUIs
- Busybox
- Embedded distributions

##### **The U-Boot boot loader**

- Introduction to U-Boot
- Booting the board through U-Boot
  - Booting from NOR
  - Booting from NAND
  - Booting from eMMC
- U-Boot environment variables
  - User-defined variables
  - Predefined variables
  - Variables substitution
- The U-Boot minimal shell
  - Writing scripts in variables
  - Executing scripts
  - Using variables in scripts: the set-script pattern
- U-Boot main commands
  - Booting an OS
  - Accessing flash chips
  - Accessing file systems (NFS, FAT, EXT<sub>x</sub>, JFFS2&)
- The full U-Boot shell
  - Script structure

- Control flow instructions (if, for&)

*Exercise: Booting the board on NFS, using pre-existing images*

### **SECOND DAY**

#### **Creating the embedded Linux kernel**

- Downloading stable source code
  - Getting a tarball
  - Using GIT
- Configuring the kernel
- Compiling the kernel and its modules
  - Modules delivered in-tree
  - Out-of-tree modules
- Installing the kernel and the modules

*Exercise: Configuring and compiling a target kernel for the target board*

#### **The Linux BSP**

- Linux BSP architecture
  - Overall structure
  - The ARM BSP
  - The Linux build system
- Linux device drivers
- Defining and initializing the board
  - Plug and Play buses
  - Using the Flattened Device Tree

*Exercise: Create a minimal BSP for the target board, editing the device tree.*

#### **Creating a root file system**

- Packages
  - Various package build systems (autotools, CMake, &)
  - Cross-compiling a package
- The all-in-one applications
  - Busybox, the basic utilities
  - Dropbear: encrypted communications (ssh)
- Manually building your root file system
  - Device nodes, programs and libraries
  - Configuration files (network, udev, &)
  - Installing modules
  - Looking for and installing the needed libraries
  - Testing file system consistency and completeness

*Exercise: Configuring and compiling Busybox and Dropbear*

*Exercise: Creating a minimal root file system using busybox and dropbear*

#### **The Linux Boot**

- Linux kernel parameters
- The Linux startup sequence
- Various initialization systems
  - busybox init
  - system V init
  - systemd
- Automatically starting an embedded system

*Exercise: Boot Linux automatically starting a user application*

## **THIRD DAY**

### **Embedded file systems**

- Storage interfaces
  - Block devices
  - MTD
- Flash memories and Linux MTDs
  - NOR flashes
  - NAND flashes
  - ONENAND flashes
- The various flash file system formats
  - JFFS2, YAFFS2, UBIFS
- Read-only file system
  - CRAMFS, SQUASHFS
- Standard Linux file systems
  - Ext2/3/4, FAT, NFS
- Ramdisks and initrd
  - Creating an initramfs
  - Booting through an initramfs
- Choosing the right file system formats
- Flashing the file system

*Exercise: Building an initrd root file system*

### **Buildroot**

- Operation
  - Toolchain configuration
  - Package selection
  - System configuration (serial port, filling /dev, &)
  - Kernel and bootloader configuration
  - Building File system image
- Customization
  - Using a pre-built toolchain
  - Adding a patch to an existing package
  - Adding a new package
  - Using a custom rootfs skeleton

*Exercise: Building a root file system using Buildroot*

*Exercise: Add a package to the root file system using Buildroot*

## **FOURTH DAY**

### **Introduction to Yocto**

- Overview of Yocto
  - History
  - Yocto, Open Embedded and Poky
  - Purpose of the Yocto project
  - The main projects
- Yocto architecture
  - Overview
  - Recipes and classes
  - Tasks

*Exercise: Building a root file system using Yocto*

## The Yocto build system

- Build system objectives
  - Building deployable images

*Exercise: Building a root file system using Yocto*

- Layers and layer priorities
- Directory layout
- Configuration files (local, machine and distribution)
- The bitbake tool
  - Common options
- Using Yocto
  - Building a package
  - Building an image (root file system + u-boot + kernel)

*Exercise: Use bitbake commands to build package & images*

## Yocto package recipes structure

- Recipe architecture
  - Tasks
  - Task dependencies
  - Recipe dependencies
- The bitbake language
  - Standard variables and functions
  - Classes and recipes
  - The base Yocto classes
  - Main bitbake commands
- Adding a new layer
  - Layer structure
  - Various kinds of layers

*Exercise: Adding a new layer*

*Exercise: Adding a new recipe*

## ANNEXES

## Real-time Linux

- real-time solutions for Linux
  - xenomai
  - real-time patch
- xenomai architecture
  - co-kernel
  - skins
  - RTDM drivers
- install
  - installing xenomai
  - cross-compiling a xenomai application

*Exercise: installing Xenomai 3.x*

*Exercise: measuring latencies*

*Exercise: cross-compiling a xenomai application*

## The U-Boot BSP

- Building and installing U-Boot
- Porting U-Boot
  - U-Boot source structure
  - Adding a new board to U-Boot sources
  - U-Boot drivers (RS232, network, flash, SD/MMC)
  - U-Boot startup and board initialization

*Exercise: Creating a board support package in U-Boot and setup of initialization code*

*Exercise: Looking at a first-load program*

## Renseignements pratiques

**Inquiry : 4 days**