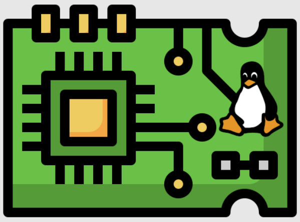


Embedded Linux system development training

Course duration

5 days – 40 hours

Language

Materials	English
Oral Lecture	English
	French
	Portuguese
	Italian

Trainer

One of the following engineers

- Alexandre Belloni
- Alexis Lothoré
- Antonin Godard
- Grégory Clement
- Jérémie Dautheribes
- João Marcos Costa
- Luca Ceresoli
- Maxime Chevallier
- Miquèl Raynal
- Richard Genoud
- Thomas Petazzoni

Contact

training@bootlin.com

+33 484 258 097

bootlin

bootlin.com

Onsite training

Audience

People developing devices using the Linux kernel
People supporting embedded Linux system developers.

Training objectives

- Be able to understand the overall architecture of Embedded Linux systems.
- Be able to choose, build, setup and use a cross-compilation toolchain.
- Be able to understand the booting sequence of an embedded Linux system, and to set up and use the U-Boot bootloader.
- Be able to select a Linux kernel version, to configure, build and install the Linux kernel on an embedded system.
- Be able to create from scratch a Linux root filesystem, including all its elements: directories, applications, configuration files, libraries.
- Be able to choose and setup the main Linux filesystems for block and flash storage devices, and understand their main characteristics.
- Be able to interact with hardware devices, configure the kernel with appropriate drivers and extend the *Device Tree*
- Be able to select, cross-compile and integrate open-source software components (libraries, applications) in an Embedded Linux system, and to handle license compliance.
- Be able to setup and use an embedded Linux build system, to build a complete system for an embedded platform.
- Be able to develop and debug applications on an embedded Linux system.

Prerequisites

- **Knowledge and practice of UNIX or GNU/Linux commands**: participants must be familiar with the Linux command line. Participants lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides.
- **Minimal English language level: B1**, according to the *Common European Framework of References for Languages*, for our sessions in English. See the CEFR grid for self-evaluation.

Pedagogics

- Lectures delivered by the trainer: 50% of the duration
- Practical labs done by participants: 50% of the duration
- Electronic copies of presentations, lab instructions and data files. They are freely available here.

Certificate

Only the participants who have attended all training sessions, and who have scored over 50% of correct answers at the final evaluation will receive a training certificate from Bootlin.

Disabilities

Participants with disabilities who have special needs are invited to contact us at *training@bootlin.com* to discuss adaptations to the training course.

Required equipement

For on-site session delivered at our customer location, our customer must provide:

- Video projector
- One PC computer on each desk (for one or two persons) with at least 16 GB of RAM, and Ubuntu Linux 24.04 installed in a free partition of at least 30 GB
- Distributions other than Ubuntu Linux 24.04 are not supported, and using Linux in a virtual machine is not supported.
- Unfiltered and fast connection to Internet: at least 50 Mbit/s of download bandwidth, and no filtering of web sites or protocols.
- PC computers with valuable data must be backed up before being used in our sessions.

For on-site sessions organized at Bootlin premises, Bootlin provides all the necessary equipment.

Hardware platform for practical labs

STM32MP1 Discovery Kit

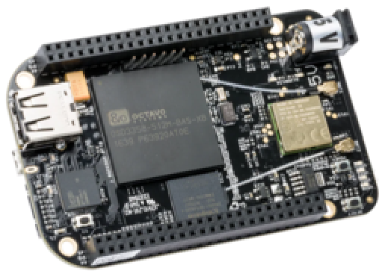
One of these Discovery Kits from STMicroelectronics: **STM32MP157A-DK1**, **STM32MP157D-DK1**, **STM32MP157C-DK2** or **STM32MP157F-DK2**

- STM32MP157, dual Cortex-A7 processor from STMicroelectronics
- USB powered
- 512 MB DDR3L RAM
- Gigabit Ethernet port
- 4 USB 2.0 host ports
- 1 USB-C OTG port
- 1 Micro SD slot
- On-board ST-LINK/V2-1 debugger
- Arduino compatible headers
- Audio codec, buttons, LEDs
- LCD touchscreen (DK2 kits only)

BeagleBone Black

BeagleBone Black or **BeagleBone Black Wireless** board

- An ARM AM335x (single Cortex-A8) processor from Texas Instruments
- USB powered
- 512 MB of RAM
- 2 or 4 GB of on-board eMMC storage
- USB host and device
- HDMI output
- 2 x 46 pins headers, to access UARTs, SPI buses, I2C buses and more.
- Ethernet or WiFi



BeaglePlay

BeaglePlay board

- Texas Instruments AM625x (4xARM Cortex-A53 CPU)
- SoC with 3D acceleration, integrated MCU and many other peripherals.
- 2 GB of RAM
- 16 GB of on-board eMMC storage
- USB host and USB device, microSD, HDMI
- 2.4 and 5 GHz WiFi, Bluetooth and also Ethernet
- 1 MicroBus Header (SPI, I2C, UART, ...), OLDI and CSI connector.

Training Schedule

Day 1 - Morning

Lecture	Introduction to embedded Linux	▪ Advantages of Linux versus traditional embedded operating systems. ▪ Typical hardware platforms used to run embedded Linux systems. ▪ Overall architecture of embedded Linux systems: overview of the major software components. ▪ Development environment for Embedded Linux development.
Lecture	Cross-compiling toolchain and C library	▪ What's inside a cross-compiling toolchain ▪ Choosing the target C library ▪ What's inside the C library ▪ Ready to use cross-compiling toolchains ▪ Building a cross-compiling toolchain with automated tools.
Lab	Cross compiling toolchain	▪ Getting and configuring Crosstool-NG ▪ Executing it to build a custom cross-compilation toolchain ▪ Exploring the contents of the toolchain

Day 1 - Afternoon

Lecture	Boot process, firmware, bootloaders	▪ Booting process of embedded platforms, focus on the *x86* and *ARM* architectures ▪ Boot process and bootloaders on *x86* platforms (legacy and UEFI) ▪ Boot process on ARM platforms: ROM code, bootloaders, *ARM Trusted Firmware* ▪ Focus on U-Boot: configuration, installation, and usage. ▪ U-Boot commands, U-Boot environment, U-Boot scripts, U-Boot generic distro boot mechanism
Lab	Bootloader and U-boot	▪ Set up serial communication with the board. ▪ Configure, compile and install U-Boot for the target hardware. ▪ Only on STM32MP1: configure, compile and install Trusted Firmware-A ▪ Become familiar with U-Boot environment and commands. ▪ Set up TFTP communication with the board. Use TFTP U-Boot commands.

Day 2 - Morning

Lecture	Linux kernel	▪ Role and general architecture of the Linux kernel ▪ Separation between kernel and user-space, and interfaces between user-space and the Linux kernel ▪ Understanding Linux kernel versions: choosing between vendor-provided kernel and upstream kernel, *Long Term Support* versions ▪ Getting the Linux kernel source code
Lab	Fetching Linux kernel sources	▪ Clone the mainline Linux tree ▪ Accessing stable releases
Lecture	Configuring, compiling and booting the Linux kernel	▪ Configuring the Linux kernel: ready-made configuration files, configuration interfaces ▪ Concept of *Device Tree* ▪ Cross-compiling the Linux kernel ▪ Study of the generated files and their role ▪ Installing and booting the Linux kernel ▪ The Linux kernel command line

Lab	Kernel cross-compiling and booting	▪ Configuring the Linux kernel and cross-compiling it for the embedded hardware platform. ▪ Downloading your kernel on the board through U-boot's TFTP client. ▪ Booting your kernel. ▪ Automating the kernel boot process with U-Boot scripts.

Day 2 - Afternoon

Lecture	Root filesystem in Linux	▪ Filesystems in Linux. ▪ Role and organization of the root filesystem. ▪ Location of the root filesystem: on storage, in memory, from the network. ▪ Device files, virtual filesystems. ▪ Contents of a typical root filesystem.
Lecture	BusyBox	▪ Detailed overview. Detailed features. ▪ Configuration, compiling and deploying.
Lab	Tiny root filesystem built from scratch with BusyBox	▪ Setting up a kernel to boot your system on a workstation directory exported by NFS ▪ Passing kernel command line parameters to boot on NFS ▪ Creating the full root filesystem from scratch. Populating it with BusyBox based utilities. ▪ System startup using BusyBox `init` ▪ Using the BusyBox HTTP server. ▪ Controlling the target from a web browser on the PC host. ▪ Setting up shared libraries on the target and compiling a sample executable.

Day 3 - Morning

Lecture	Accessing hardware devices	▪ How to access hardware on popular busses: USB, SPI, I2C, PCI ▪ Usage of kernel drivers and direct user-space access ▪ The *Device Tree* syntax, and how to use it to describe additional devices and pin-muxing ▪ Finding Linux kernel drivers for specific hardware devices ▪ Using kernel modules ▪ Hardware access using `/dev` and `/sys` ▪ User-space interfaces for the most common hardware devices: storage, network, GPIO, LEDs, audio, graphics, video
Lab	Accessing hardware devices	▪ Exploring the contents of `/dev` and `/sys` and the devices available on the embedded hardware platform. ▪ Using GPIOs and LEDs. ▪ Modifying the Device Tree to control pin multiplexing and to declare an I2C-connected joystick. ▪ Adding support for a USB audio card using Linux kernel modules ▪ Adding support for the I2C-connected joystick through an out-of-tree module.

Day 3 - Afternoon

Lecture	Block filesystems	▪ Accessing and partitioning block devices. ▪ Filesystems for block devices. ▪ Usefulness of journaled filesystems. ▪ Read-only block filesystems. ▪ RAM filesystems. ▪ How to create each of these filesystems. ▪ Suggestions for embedded systems.

Lab	Block filesystems	▪ Creating partitions on your SD card ▪ Booting a system with a mix of filesystems: *SquashFS* for the root filesystem, *ext4* for system data, and *tmpfs* for temporary system files.
Lecture	Flash filesystems	▪ The Memory Technology Devices (MTD) filesystem. ▪ Filesystems for MTD storage: JFFS2, Yaffs2, UBIFS. ▪ Kernel configuration options ▪ MTD storage partitions. ▪ Focus on today's best solution, UBI and UBIFS: preparing, flashing and using UBI images. *Note: as the embedded hardware platform used for the labs does not have any flash-based storage, this lecture will not be illustrated with a corresponding practical lab.*

Day 4 - Morning

Lecture	Cross-compiling user-space libraries and applications	▪ Configuring, cross-compiling and installing applications and libraries. ▪ Concept of build system, and overview of a few common build systems used by open-source projects: Makefile, *autotools*, *CMake*, *meson* ▪ Overview of the common issues encountered when cross-compiling.
Lab	Cross-compiling applications and libraries	▪ Manual cross-compilation of several open-source libraries and applications for an embedded platform. ▪ Learning about common pitfalls and issues, and their solutions. ▪ This includes compiling *alsa-utils* package, and using its `speaker-test` program to test that audio works on the target.

Day 4 - Afternoon

Lecture	Embedded system building tools	▪ Approaches for building embedded Linux systems: build systems and binary distributions ▪ Principle of *build systems*, overview of Yocto Project/OpenEmbedded and Buildroot. ▪ Principle of *binary distributions* and useful tools, focus on Debian/Ubuntu ▪ Specialized software frameworks/distributions: Tizen, AGL, Android
Lab	System build with Buildroot	▪ Using Buildroot to rebuild the same basic system plus a sound playing server (*MPD*) and a client to control it (*mpc*). ▪ Driving music playback, directly from the target, and then remotely through an MPD client on the host machine. ▪ Analyzing dependencies between packages.
Lecture	Open source licenses and compliance	▪ Presentation of the most important open-source licenses: GPL, LGPL, MIT, BSD, Apache, etc. ▪ Concept of *copyleft* licenses ▪ Differences between (L)GPL version 2 and 3 ▪ Compliance with open-source licenses: best practices

Day 5 - Morning

Lecture	Overview of major embedded Linux software stacks	▪ `systemd` as an *init* system ▪ Hardware management with *udev* ▪ Inter-process communication with *D-Bus* ▪ The graphics software stack: DRM/KMS, X.org, Wayland, Qt, Gtk, OpenGL ▪ The multimedia software stack: Video4Linux, GStreamer, Pulseaudio, Pipewire

Lab	Integration of additional software stacks	- Integration of *systemd* as an init system - Use *udev* built in *systemd* for automatic module loading

Day 5 - Afternoon

Lecture	Application development and debugging	- Programming languages and libraries available. - Build system for your application, an overview of *CMake* and *meson* - The *gdb* debugger: remote debugging with *gdbserver*, post-mortem debugging with `core` files - Performance analysis, tracing and profiling tools, memory checkers: `strace`, `ltrace`, `perf`, `valgrind`
Lab	Application development and debugging	- Creating an application that uses an I2C-connected joystick to control an audio player. - Setting up an IDE to develop and remotely debug an application. - Using *strace*, *ltrace*, *gdbserver* and *perf* to debug/investigate buggy applications on the embedded board.
Lecture	Useful resources	- Books about embedded Linux and system programming - Useful online resources - International conferences