Embedded Linux system development training

On-site training, 4 days

Latest update: September 06, 2022

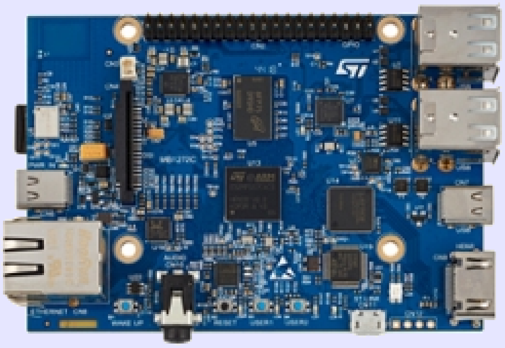
Title	**Embedded Linux system development training**
Training objectives	• Understand the overall architecture of Embedded Linux systems. • Understand the role and internals of a cross-compilation toolchain and setup your own. • Understand the booting process of embedded systems, the main bootloaders, and setup your own bootloader. • Understand the role and overall architecture of Linux kernel, how to configure, build and install it on your embedded system. • Understand the principle and contents of a Linux root filesystem, and create your own Linux root filesystem from scratch. • Discover the different filesystem for block storage devices, and use them on your embedded system. • Discover major open-source software components for embedded systems, understanding licensing constraints, how to integrate and cross-compile third-party software components, and experiment cross-compilation of open-source libraries. • Discover the main embedded Linux build systems, and experiment one of them. • Understand the principles and tools for application development and debugging on embedded Linux systems.
Duration	**Four** days - 32 hours (8 hours per day).
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 at bootlin.com/doc/training/embedded-linux-4d.
Trainer	One of the engineers listed on: `https://bootlin.com/training/trainers/`
Language	Oral lectures: English or French. Materials: English.
Audience	People developing devices using the Linux kernel People supporting embedded Linux system developers.

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 at bootlin.com/blog/command-line/. • **Minimal English language level: B1**, according to the *Common European Framework of References for Languages*, for our sessions in English. See bootlin.com/pub/training/cefr-grid.pdf for self-evaluation.
Alternative version	Full version of the Embedded Linux system development course, (**5 days long**) with 2 additional half days with practical labs: • Flash filesystems • Real time Practical labs using a Microchip SAMA5D3 Xplained board `https://bootlin.com/doc/training/embedded-linux/embedded-linux-agenda.pdf`.
Required equipment	**For on-site sessions at our customer location, the customer must provide**: • Video projector • One PC computer on each desk (for one or two persons) with at least 8 GB of RAM, and Ubuntu Linux 20.04 installed in a **free partition of at least 30 GB** • Distributions others than Ubuntu Linux 20.04 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.
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.

Hardware

Using the *STMicroelectronics STM32MP157D-DK1 Discovery board* in all practical labs. This board features:

- STM32MP157D dual ARM Cortex-A7 processor
- USB-C 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 Uno v3-compatible header
- Audio codec
- Misc: buttons, LEDs



Day 1 - Morning

Lecture - Introduction to embedded Linux

- Introduction to Free Software
- Reasons for choosing Free Software in embedded operating systems
- Example embedded systems running Linux
- CPU, RAM and storage requirements
- Choosing a hardware platform
- System architecture: main components
- Embedded system development tasks

Lecture - Embedded Linux development environment

- Operating system and tools to use on the development workstation 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.

Day 1 - Afternoon

Lab - Cross compiling toolchain

- Configuring Crosstool-NG
- Executing it to build a custom uClibc toolchain.

Lecture - Bootloaders

- Available bootloaders
- Bootloader features
- Installing a bootloader
- Detailed study of U-Boot

Lab - Bootloader and U-boot

Using the STM32MP157D-DK1 Discovery board

- Set up serial communication with the board.
- Configure, compile and install the first-stage bootloader and U-Boot on the Discovery board.
- Become familiar with U-Boot environment and commands.
- Set up TFTP communication with the board. Use TFTP U-Boot commands.

Lecture - Linux kernel

- Role and general architecture of the Linux kernel
- Features available in the Linux kernel, with a focus on features useful for embedded systems
- Kernel user interface
- Getting the sources
- Linux kernel release process. Long Term Support versions.
- Using the patch command

Day 2 - Morning

Lab - Kernel sources	Lecture – Configuring and compiling a Linux kernel
• Downloading kernel sources • Apply kernel patches	• Kernel configuration. • Using ready-made configuration files for specific architectures and boards. • Kernel compilation. • Generated files. • Using kernel modules

Lab - Kernel cross-compiling and booting

Using the STM32MP157D-DK1 Discovery board

- Configuring the Linux kernel and cross-compiling it for the ARM board.
- Downloading your kernel on the board through U-boot's tftp client.
- Booting your kernel from RAM.
- Copying the kernel to flash and booting it from this location.
- Storing boot parameters in flash and automating kernel booting from flash.

Day 2 - Afternoon

Lecture – Root filesystem in Linux	Lecture - BusyBox
• 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.	• Detailed overview. Detailed features. • Configuration, compiling and deploying.

Lab – Tiny root filesystem built from scratch with BusyBox

Using the STM32MP157D-DK1 Discovery board

- Now build a basic root filesystem from scratch for your ARM system
- 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.
- Creating device files and booting the virtual system.
- System startup using BusyBox /sbin/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 - Block filesystems

- 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

Using the STM32MP157D-DK1 Discovery board

- Booting a system with a mix of filesystems: SquashFS for applications, ext3 for configuration and user data, and tmpfs for temporary system files.

Day 3 - Afternoon

Lecture – Leveraging existing open-source components in your system

- Reasons for leveraging existing components.
- Find existing free and open source software components.
- Choosing the components.
- The different free software licenses and their requirements.
- Overview of well-known typical components used in embedded systems: graphical libraries and systems (framebuffer, Gtk, Qt, etc.), system utilities, network libraries and utilities, multimedia libraries, etc.
- System building: integration of the components.

Lecture – Cross-compiling applications and libraries

- Configuring, cross-compiling and installing applications and libraries.
- Details about the build system used in most open-source components.
- Overview of the common issues found when using these components.

Lab – Cross-compiling applications and libraries

If enough time left

- Building a system with audio libraries and a sound player application.
- Manual compilation and installation of several free software packages.
- Learning about common techniques and issues.

Day 4 - Morning

Lecture - Embedded system building tools

- Review of existing system building tools.
- Buildroot example.

Lab - System build with Buildroot

Using the STM32MP157D-DK1 Discovery board

- Using Buildroot to rebuild the same system as in the previous lab.
- Seeing how easier it gets.
- Optional: add a package to Buildroot.

Day 4 - Afternoon

Lecture - Application development and debugging

- Programming languages and libraries available.
- Overview of the C library features for application development.
- Build system for your application, how to use existing libraries in your application.
- Debuggers. Debugging remote applications with gdb and gdbserver. Post-mortem debugging with core files.
- Tracing and profiling solutions.

Lab – Application development and debugging

On the STM32MP157D-DK1 Discovery board

- Develop and compile an application relying on the ncurses library
- Using strace, ltrace and gdbserver to debug a crappy application on the remote system.
- Post mortem analysis: exploit a *core dump* to find out where an application crashed.