
# Embedded Linux system development training

On-line seminar, 6 sessions of 4 hours

Latest update: May 12, 2022

| | |
|---|---|
| **Title** | **Embedded Linux system development training** |
| **Training objectives** | • Be able to understand the overall architecture of Embedded Linux systems.<br>• Be able to choose, build, setup and use a cross-compilation toolchain.<br>• Be able to understand the booting sequence of an embedded Linux system, and to set up and use the U-Boot bootloader.<br>• Be able to select a Linux kernel version, to configure, build and install the Linux kernel on an embedded system.<br>• Be able to create from scratch a Linux root filesystem, including all its elements: directories, applications, configuration files, libraries.<br>• Be able to choose and setup the main Linux filesystems for block storage devices, and understand their main characteristics.<br>• Be able to select, cross-compile and integrate open-source software components (libraries, applications) in an Embedded Linux system, and to handle license compliance.<br>• Be able to setup and use an embedded Linux build system, to build a complete system for an embedded platform.<br>• Be able to develop and debug applications on an embedded Linux system. |
| **Duration** | **Six** half days - 24 hours (4 hours per half day). |
| **Pedagogics** | • Lectures delivered by the trainer, over video-conference. Participants can ask questions at any time.<br>• Practical demonstrations done by the trainer, based on practical labs, over video-conference. Participants can ask questions at any time. Optionally, participants who have access to the hardware accessories can reproduce the practical labs by themselves.<br>• Instant messaging for questions between sessions (replies under 24h, outside of week-ends and bank holidays).<br>• Electronic copies of presentations, lab instructions and data files. They are freely available at bootlin.com/doc/training/embedded-linux. |
| **Trainer** | One of the engineers listed on:<br>`https://bootlin.com/training/trainers/` |
| **Language** | Oral lectures: English<br>Materials: English. |


| | |
|---|---|
| **Audience** | People developing devices using the Linux kernel<br>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/.<br>• **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. |
| **Required equipment** | • Computer with the operating system of your choice, with the Google Chrome or Chromium browser for videoconferencing<br>• Webcam and microphone (preferably from an audio headset)<br>• High speed access to the Internet<br>• For people interested in our optional practical labs, an installation of VirtualBox and about 30 GB of free disk space. |
| **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. |


## Real hardware in practical demos

The hardware platform used for the practical demos of this training session is the **STMicroelectronics STM32MP157D-DK1 Discovery board** board, based on a dual Cortex-A7 processor from ST, which 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

## Optional labs on emulated hardware

For the interested participants, we propose optional labs, to be done between the training sessions, that use the QEMU emulated ARM Vexpres Cortex-A9 board. As they rely on an emulated platform, no specific hardware is necessary.


Image credits (Wikipedia): https://frama.link/mW71eosa


## Half day 1

### 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.

### Demo - 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


# Half day 2

## Demo - Bootloader and U-boot

*Using the STM32MP1 Discovery Kit 1 board*

- Set up serial communication with the board.
- Configure, compile and install the first-stage bootloader and U-Boot on the 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

## Demo - Kernel sources

- Downloading kernel sources
- Apply kernel patches

## Lecture – Configuring and compiling a Linux kernel

- Kernel configuration.
- Using ready-made configuration files for specific architectures and boards.
- Kernel compilation.
- Generated files.
- Using kernel modules

## Demo - Kernel cross-compiling and booting

*Using the STM32MP1 Discovery Kit 1 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.


## Half day 3

### 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.

### Demo – Tiny root filesystem built from scratch with BusyBox

*Using the STM32MP1 Discovery Kit 1 board*

- Now build a basic root filesystem from scratch for your ARM system
- Setting up a kernel to boot your system on a host 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.

## Half day 4

### 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.

### Demo - Block filesystems

*Using the STM32MP1 Discovery Kit 1 board*

- Booting your system with a mix of filesystems on MMC/SD storage: SquashFS for applications, ext4 for configuration and user data, and tmpfs for temporary system files.


**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.

## Half day 5

**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.

**Demo – Cross-compiling applications and libraries**

*Using the STM32MP1 Discovery Kit 1 board*

- 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.

## Half day 6

**Lecture - Embedded system building tools**

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

**Demo - System build with Buildroot**

*Using the STM32MP1 Discovery Kit 1 board*

- Using Buildroot to rebuild the same system as in the previous lab.
- Seeing how easier it gets.


## 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.

## Demo – Application development and debugging

*Using the STM32MP1 Discovery Kit 1 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.