



Embedded Linux boot time optimization training

On-line seminar, 4 sessions of 4 hours

Latest update: March 18, 2024

Title	Embedded Linux boot time optimization training
Training objectives	<ul style="list-style-type: none">• Be able to use various tools and techniques to measure the boot time of an embedded Linux system.• Be able to reduce the boot time spent during the <i>user-space</i> initialization.• Be able to reduce the boot time spent during the <i>kernel</i> initialization.• Be able to reduce the boot time spent during the <i>bootloader</i> initialization.• Be able to use advanced and alternatives techniques of boot time optimization.
Duration	Four half days - 16 hours (4 hours per half day)
Pedagogics	<ul style="list-style-type: none">• Lectures delivered by the trainer, over video-conference. Participants can ask questions at any time.• 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.• Instant messaging for questions between sessions (replies under 24h, outside of week-ends and bank holidays).• Electronic copies of presentations, lab instructions and data files. They are freely available at https://bootlin.com/doc/training/boot-time.
Trainer	One of the engineers listed on: https://bootlin.com/training/trainers/
Language	Oral lectures: English, French. Materials: English.
Audience	People developing embedded Linux systems. People supporting embedded Linux system developers.



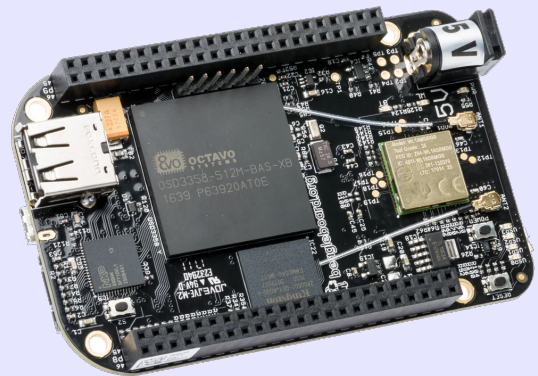
Prerequisites	<ul style="list-style-type: none">• Knowledge and practice of UNIX or GNU/Linux commands: participants must be familiar with the Linux command line. Participants lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides at bootlin.com/blog/command-line/.• Minimal experience in embedded Linux development: participants should have a minimal understanding of the architecture of embedded Linux systems: role of the Linux kernel vs. user-space, development of Linux user-space applications in C. Following Bootlin's <i>Embedded Linux</i> course at bootlin.com/training/embedded-linux/ allows to fulfill this pre-requisite.• Minimal English language level: B1, according to the <i>Common European Framework of References for Languages</i>, for our sessions in English. See bootlin.com/pub/training/cefr-grid.pdf for self-evaluation.
Required equipment	<ul style="list-style-type: none">• Computer with the operating system of your choice, with the Google Chrome or Chromium browser for videoconferencing.• Webcam and microphone (preferably from an audio headset)• High speed access to the Internet
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

The hardware platform used for the practical demos of this training session is the **BeagleBone Black** board, which features:

- An ARM AM335x processor from Texas Instruments (Cortex-A8 based), 3D acceleration, etc.
- 512 MB of RAM
- 2 GB of on-board eMMC storage (4 GB in Rev C)
- USB host and device
- HDMI output
- 2 x 46 pins headers, to access UARTs, SPI buses, I2C buses and more.



Demos

The practical demos of this training session use the following hardware peripherals:

- A USB webcam
- An LCD and touchscreen cape connected to the BeagleBone Black board, to display the video captured by the webcam.



Half day 1

Lecture - Principles

- How to measure boot time
- Main ideas

Demo - Preparing the system

- Downloading bootloader, kernel and Buildroot source code
- Board setup, setting up serial communication
- Configure Buildroot and build the system
- Configure and build the U-Boot bootloader. Prepare an SD card and boot the bootloader from it.
- Configure and build the kernel. Boot the system

Lecture - Measuring time

- Generic software techniques
- Hardware techniques
- Specific solutions for each stage

Demo - Measuring time - Software solution

- Modify the system to measure time at various steps
- Timing messages on the serial console
- Timing the launching of the application

Half day 2

Lecture - Toolchain optimizations

- Introduction to toolchains
- C libraries
- Size information
- Measuring executable performance with `time`



Demo - Toolchain optimizations

- Measuring application execution time
- Switching to a Thumb2 toolchain
- Generate a Buildroot SDK to rebuild faster

Lecture - Application optimization

- Using `strace` and `ltrace`
- Other profiling techniques

Demo - Application optimization

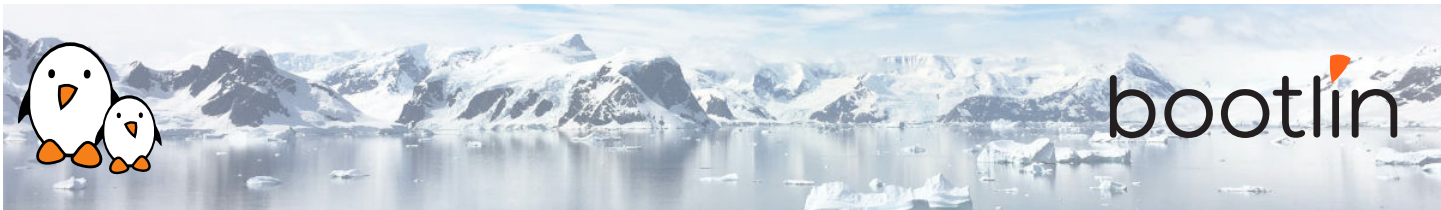
- Finding unnecessary configuration options in applications
- Modifying configuration options through Buildroot
- Experiments with `strace` to trace program execution

Lecture - Optimizing system initialization

- Using BusyBox `bootchartd`
- Optimizing init scripts
- Possibility to start your application directly

Demo - Optimizing system initialization

- Using Buildroot to remove unnecessary scripts and commands
- Access-time based technique to identify unused files
- Simplifying BusyBox
- Starting the application as the init program



Half day 3

Lecture - Filesystem optimizations

- Available filesystems, performance and boot time aspects
- Making UBIFS faster
- Tweaks for reducing boot time
- Booting on an initramfs
- Using static executables: licensing constraints

Demo - Filesystem optimizations

- Trying and measuring two block filesystems: ext4 and SquashFS.
- Trying and measuring the initramfs solution. Constraints due to this solution.

Lecture - Kernel optimizations

- Using *Initcall debug* to generate a boot graph
- Compression and size features
- Reducing or suppressing console output
- Multiple tweaks to reduce boot time

Demo - Kernel optimizations

- Generating and analyzing a boot graph for the kernel
- Find and eliminate unnecessary kernel features
- Find the best kernel compression solution for our system

Half day 4

Demo - Kernel optimizations

Continued from the previous session



Lecture - Bootloader optimizations

- Generic tips for reducing U-Boot's size and boot time
- Optimizing U-Boot scripts and kernel loading
- Skipping the bootloader - How to modify U-Boot to enable its *Falcon mode*

Lecture - U-Boot Falcon mode

- Principles and goals
- The Device Tree preparation work that U-Boot does to prepare Linux kernel booting
- Using the `spl export` command to do this work in advance
- Modifying U-Boot's source code and configuring it for directly booting Linux and skipping the U-Boot second stage.
- Example instructions and setups for booting from MMC and NAND flash
- How to debug Falcon mode
- How to fall back to U-Boot
- Limitations

Demo - Bootloader optimizations

- Using the above techniques to make the bootloader as quick as possible.
- Switching to faster storage
- Configuring U-Boot for *Falcon mode* booting, skipping U-Boot's second stage.

Wrap-up - Achieved results

- Summary of results
- Questions and answers, experience sharing with the trainer