Embedded Linux boot time optimization training

On-site training, 3 days

Latest update: February 24, 2023

Title	**Embedded Linux boot time optimization training**
Training objectives	• 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 *user-space* initialization. • Be able to reduce the boot time spent during the *kernel* initialization. • Be able to reduce the boot time spent during the *bootloader* initialization. • Be able to use advanced and alternatives techniques of boot time optimization.
Duration	**Three** days - 24 hours (8 hours per day)
Pedagogics	• Lectures delivered by the trainer: 40% of the duration • Practical labs done by participants: 60% of the duration • 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

- **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 *Embedded Linux* course at bootlin.com/training/embedded-linux/ allows to fulfill this pre-requisite.
- **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

- 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

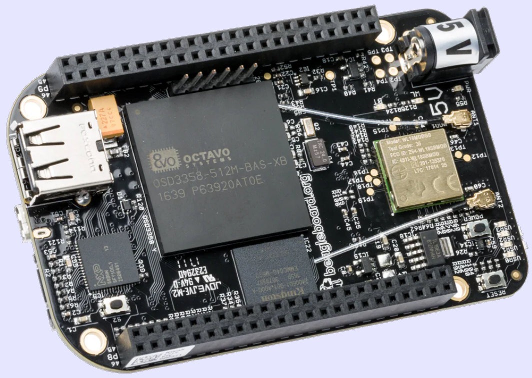
Only the participants who have attended all training sessions, and who have scored over 50% of correct answers at the final evaluation will receive a training certificate from Bootlin.

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

The hardware platform used for the practical labs 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.

Practical labs

The practical labs 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.
- We will also use an Arduino board as a way to measure boot time with accurary, demonstrating a hardware boot time measurement technique.

Day 1 - Morning

Lecture - Principles	**Lab - Preparing the system**
• How to measure boot time • Main ideas	• 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

Day 1 - Afternoon

Lecture - Measuring time	**Lab - Measuring time - Software solution**
• Generic software techniques • Hardware techniques • Specific solutions for each stage	• Modify the system to measure time at various steps • Timing messages on the serial console • Timing the execution of the application

Lab - Measuring time - Hardware solution	**Lecture - Toolchain optimizations**
• Measure total boot time by toggling a GPIO • Setting up an Arduino board • Preparing a test circuit with a 7-segment display • Modifying the DTS to configuring Bone Black pins as GPIOs • Making the application drive the custom GPIOs	• Introduction to toolchains • C libraries • Size information • Measuring executable performance with `time`

Lab - Toolchain optimizations

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

Day 2- Morning

Lecture - Application optimization

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

Lab - 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

Lab - 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

Day 2 - Afternoon

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

Lab - 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

Lab - 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

Day 3 - Morning

Lab - Kernel optimizations

- Continued from Day 2

Day 3 - Afternoon

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

Lab - 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

- Sharing and comparing results achieved by the various groups
- Questions and answers, experience sharing with the trainer