Boot Time Optimization Training

3-day session

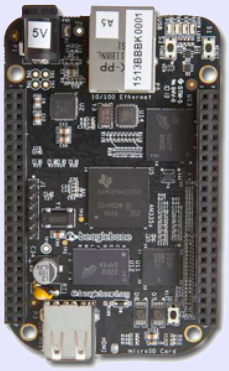
Title	**Boot Time Optimization Training**
Overview	Measuring boot time Reducing user space boot time Reducing kernel boot time Bootloader optimizations Advanced techniques and alternatives Practical labs with the ARM-based BeagleBone Black board (or with its Wireless variant).
Materials	Check that the course contents correspond to your needs: `https://bootlin.com/doc/training/boot-time`.
Duration	**Three** days - 24 hours. 25% of lectures, 75% of practical labs.
Trainer	One of the engineers listed on `https://bootlin.com/training/trainers/`
Language	Oral lectures: English or 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** People lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides: `https://bootlin.com/blog/command-line/` **Knowledge and practice of embedded Linux system development**

Required equipment	**For on-site sessions only.** Everything is supplied by Bootlin in public sessions. • Video projector • PC computers with at least 8 GB of RAM, and Ubuntu Linux installed in a **free partition of at least 30 GB. Using Linux in a virtual machine is not supported**, because of issues connecting to real hardware. • We need Ubuntu Desktop 20.04 (Xubuntu and other variants are fine). We don't support other distributions, because we can't test all possible package versions. • **Connection to the Internet** (direct or through the company proxy). • **PC computers with valuable data must be backed up** before being used in our sessions. Some people have already made mistakes during our sessions and damaged work data.
Materials	Electronic copies of presentations and labs. Electronic copy of lab files.

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

- How to measure boot time
- Main ideas

Lab - 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

Day 1 - Afternoon

Lecture - Measuring time

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

Lab - Measuring time - Software solution

- 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

- 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

Lecture - Toolchain optimizations

- 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`
- 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 Bootchart
- 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

- Compiling U-Boot with less features
- U-Boot configuration settings that impact boot time
- Optimizing kernel loading
- Skipping the bootloader - How to modify U-Boot to enable its *Falcon mode*

Lab - Bootloader optimizations

- Using the above techniques to make the bootloader as quick as possible.
- Switching to faster storage
- Skip the bootloader with U-Boot's *Falcon mode*

Wrap-up - Achieved results

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