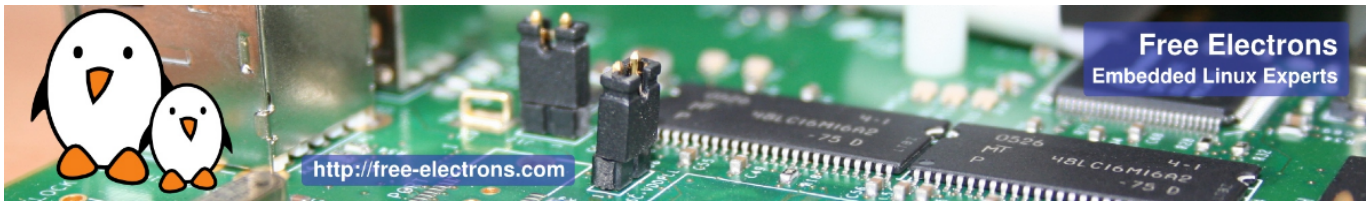
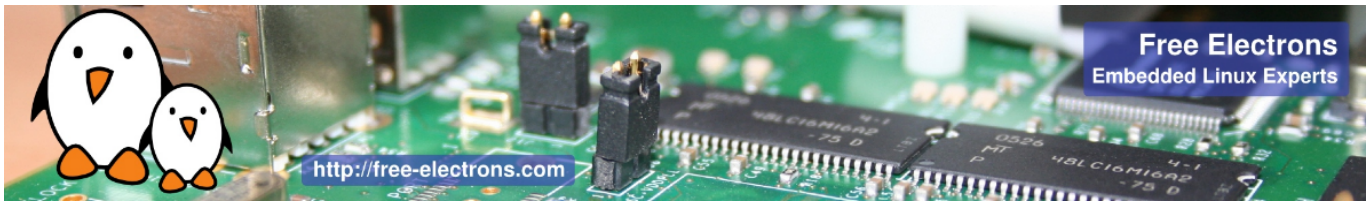


Embedded Linux system development training

5-day session

Title	**Embedded Linux system development training**
Overview	Bootloaders Kernel (cross) compiling and booting Block and flash filesystems C library and cross-compiling toolchains Lightweight building blocks for embedded systems Embedded system development tools Embedded application development and debugging Implementing real-time requirements in embedded Linux systems Practical labs with the ARM based SAMA5D3 Xplained board from Atmel
Materials	Check that the course contents correspond to your needs: `http://free-electrons.com/doc/training/embedded-linux`.
Duration	**Five** days - 40 hours (8 hours per day). 50% of lectures, 50% of practical labs.
Trainer	One of the engineers listed on: `http://free-electrons.com/training/trainers/`
Language	Oral lectures: English, French or Polish. 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** People lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides: `http://free-electrons.com/blog/command-line/`.
Alternative version	Reduced version of the Embedded Linux system development training (**4 days long**) with the following topics removed: • Flash file system • Real time `http://free-electrons.com/doc/training/embedded-linux/embedded-linux-4-days-agenda.pdf`.

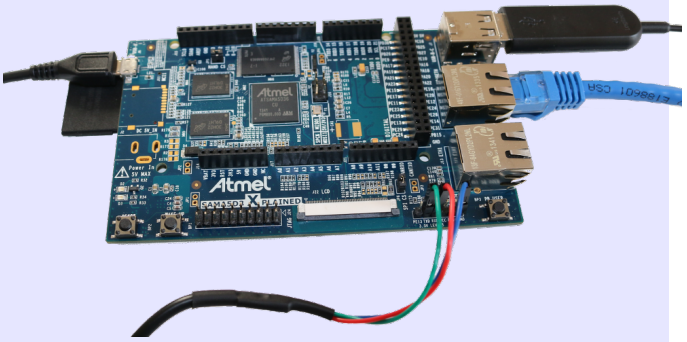


Required equipment	**For on-site sessions only** Everything is supplied by Free Electrons in public sessions. • Video projector • PC computers with at least 4 GB of RAM, and Ubuntu Linux installed in a **free partition of at least 30 GB. Using Linux in a virtual machine is not supported**, because of issues connecting to real hardware. • We need Ubuntu Desktop 16.04 (32 or 64 bit, Xubuntu and Kubuntu 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	Print and electronic copies of presentations and labs. Electronic copy of lab files.

Hardware

Using the Atmel SAMA5D3 Xplained board in all practical labs SAMA5D36 (Cortex A5) CPU from Atmel, which features:

- USB powered
- 256 MB DDR2 RAM
- 256 MB NAND flash
- 2 Ethernet ports (Gigabit + 100 Mbit)
- 2 USB 2.0 host ports
- 1 USB device port
- 1 MMC/SD slot
- 3.3 V serial port (like Beaglebone Black)
- Arduino R3-compatible header
- Misc: JTAG, buttons, LEDs

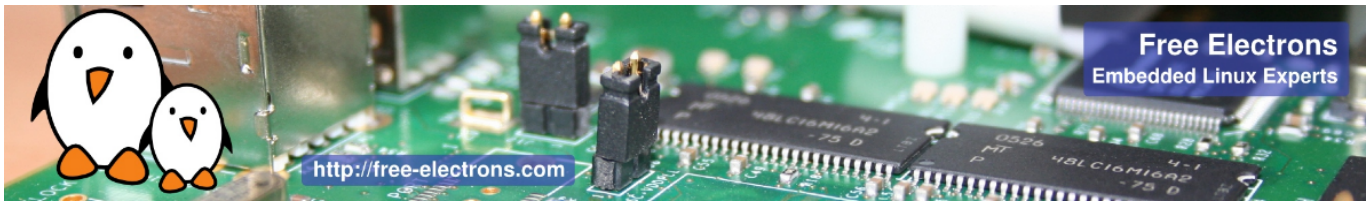


Day 1 - Morning

Lecture - Introduction to embedded Linux

- Advantages of Linux versus traditional embedded operating systems. Reasons for choosing Linux.
- Global picture: understanding the general architecture of an embedded Linux system.Overview of the major components in a typical system.

The rest of the course will study each of these components in detail.



Lecture - Embedded Linux development environment

- Operating system and tools to use on the development workstation for embedded Linux development.
- Desktop Linux usage tips.

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

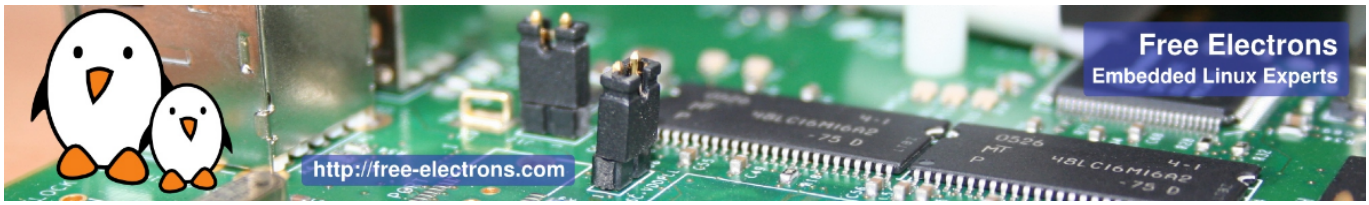
Lab - Bootloader and U-boot

Using the Atmel SAMA5D3 Xplained board

- Set up serial communication with the board.
- Configure, compile and install the first-stage bootloader and U-Boot on the Xplained 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
- Understanding Linux kernel versions.
- Using the patch command



Day 2 - Morning

Lab - Kernel sources

- Downloading kernel sources
- Apply kernel patches

Lecture – Configuring and compiling a Linux kernel

- Kernel configuration.
- Useful settings for embedded systems.
- Native compiling.
- Generated files.
- Using kernel modules

Lecture - Kernel cross-compiling

- Kernel cross-compiling setup.
- Using ready-made configuration files for specific architectures and boards.
- Cross-compiling Linux

Lab - Kernel cross-compiling and booting

Using the Atmel Xplained 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.

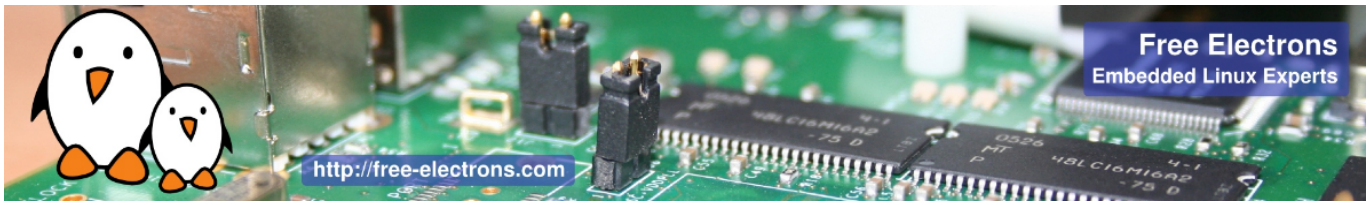
Day 2 - Afternoon

Lecture – Root filesystem in Linux

- 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.



Lab – Tiny root filesystem built from scratch with BusyBox

Using the Atmel Xplained 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

Lab – Tiny root filesystem built from scratch with BusyBox

Continued from the previous afternoon.

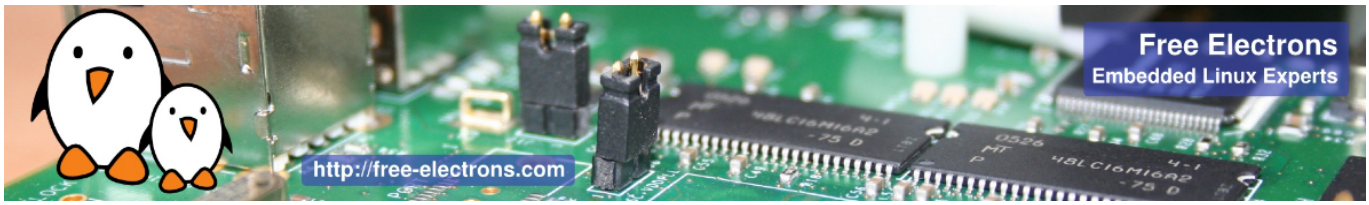
Lecture - Block filesystems

- 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 Xplained ARM board

- Creating partitions on your block storage
- 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 - Flash filesystems

- The Memory Technology Devices (MTD) filesystem.
- Filesystems for MTD storage: JFFS2, Yaffs2, UBIFS.
- Kernel configuration options
- MTD storage partitions.
- Focus on today's best solution, UBI and UBIFS: preparing, flashing and using UBI images.

Lab – Flash filesystems

Using the SAMAD3 Xplained ARM board

- Defining partitions in U-Boot for your internal flash storage instead of using raw offsets.
- Sharing these definitions with Linux.
- Creating a UBI image on your workstation, flashing it from U-Boot and booting your system on one of the UBI volumes with UBIFS.

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.

Day 4 - Morning

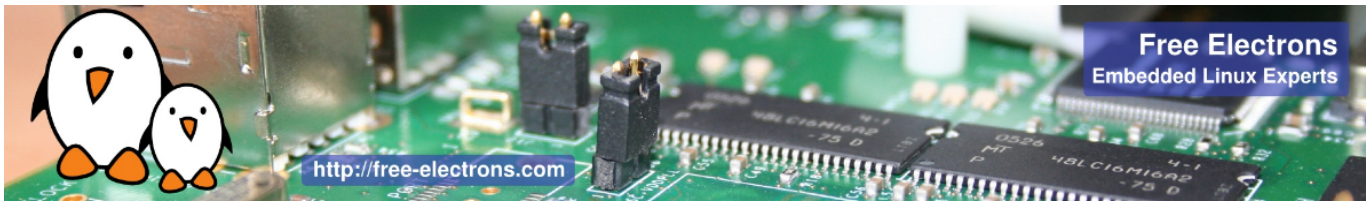
Lecture – Cross-compiling applications and libraries

- 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 - Afternoon

Lecture - Embedded system building tools

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

Lab - System build with Buildroot

Using the Atmel Xplained 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 5 - Morning

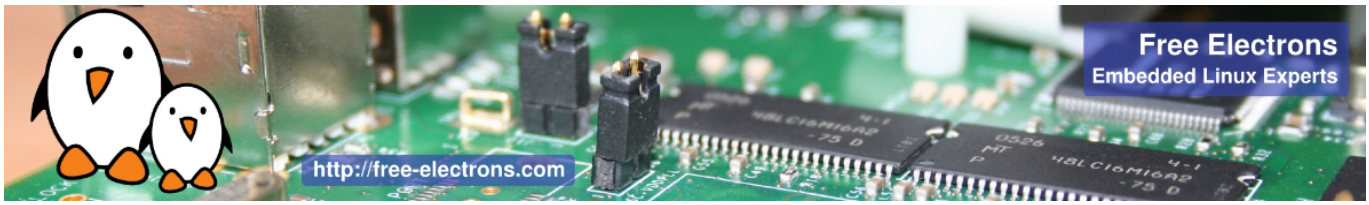
Lecture - Application development and debugging

- 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.
- Source browsers and Integrated Development Environments (IDEs).
- Debuggers. Debugging remote applications with gdb and gdbserver. Post-mortem debugging with core files.
- Code checkers, memory checkers, profilers.

Lab – Application development and debugging

On the Atmel Xplained 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.
- Do post-mortem analysis of a crashed application.



Day 5 - Afternoon

Lecture - Linux and real-time

Very useful for many kinds of devices, industrial or multimedia systems.

- Understanding the sources of latency in standard Linux.
- Soft real-time solutions for Linux: improvements included in the mainline Linux version.
- Understanding and using the latest RT preempt patches for mainline Linux.
- Real-time kernel debugging. Measuring and analyzing latency.
- Xenomai, a hard real-time solution for Linux: features, concepts, implementation and examples.

Lab - Linux latency tests

- Tests performed on the Xplained ARM board.
- Latency tests on standard Linux, with preemption options.
- Latency tests using the `PREEMPT_RT` kernel patchset.
- Setting up Xenomai.
- Latency tests with Xenomai.