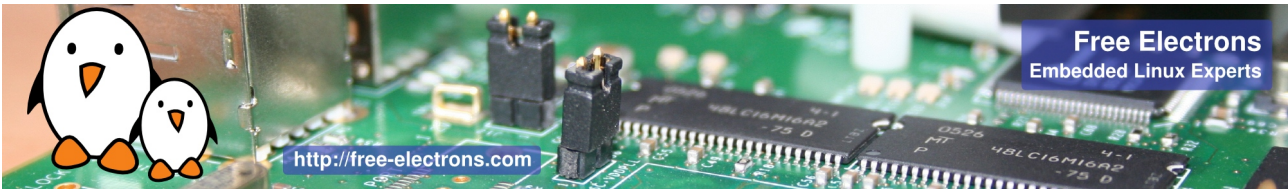


Embedded Linux system development training

5 day session

Overview

Title	Embedded Linux system development training
Overview	<p>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.</p> <p>Embedded application development and debugging. Implementing real-time requirements in embedded Linux systems. Techniques to optimize system size, RAM, power, performance and cost. Practical labs with ARM boards.</p>
Duration	5 days. 40% of presentations and 60% of practical labs.
Trainer	One of the engineers listed on http://free-electrons.com/company/staff/
Language	<p>Oral lectures: English or French Materials: English</p>
Audience	<p>People developing devices using the Linux kernel People supporting embedded Linux system developers.</p>
Prerequisites	<p>Knowledge and practice of Unix or GNU/Linux commands People lacking experience on this topic should get trained by themselves with our freely available on-line slides (http://free-electrons.com/docs/command-line/) Possibility to order an extra training day on this topic.</p>
Required equipment	<p>Video projector</p> <p>PC computers with at least 1 GB of RAM, and Ubuntu Linux installed in a free partition of at least 10 GB. Using Linux in a virtual machine is not supported, because of issues connecting to real hardware.</p> <p>We need a 32 bit (i386) version of Ubuntu Desktop 11.04. We don't support other distributions, because we can't test all possible package versions</p> <p>Connection to the Internet (direct or through the company proxy).</p> <p>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.</p>
Materials	<p>Print and electronic copy of presentations and labs. Electronic copy of lab files.</p>



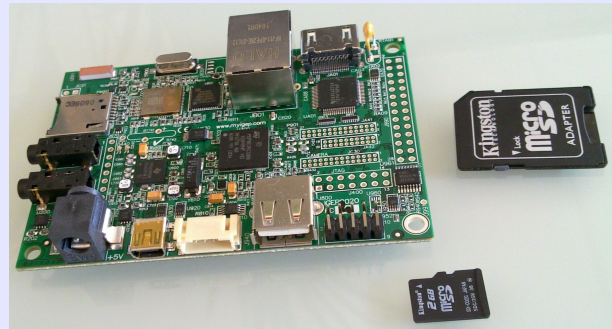
Training - Embedded Linux system development

See our training materials on <http://free-electrons.com/doc/training/embedded-linux>
 This way, you can check by yourself that they correspond to your needs.

Hardware

Using IGEPv2 boards from ISEE in most practical labs

DM3730 (OMAP3) CPU from Texas Instruments
 512 MB RAM, 512 MB flash
 1 USB 2.0 host
 1 USB device
 HDMI / DVI-D video output, audio I/O
 100 Mbit Ethernet port, Wifi, Bluetooth
 Expansion port, JTAG port, etc.



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.

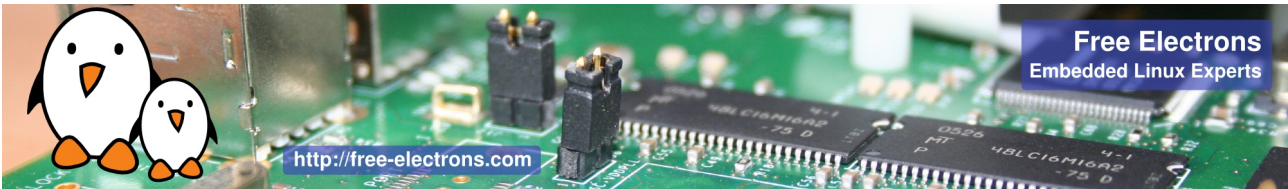
Day 1 - Afternoon

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



<p>Lab - Bootloader and U-boot</p>	<p>Lecture - Linux kernel</p>
<p>Set up serial communication with the board. Configure, compile and install the first-stage bootloader and U-Boot on the IGEPv2 board. Become familiar with U-Boot environment and commands. Set up TFTP communication with the board. Use TFTP U-Boot commands.</p>	<p>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</p>

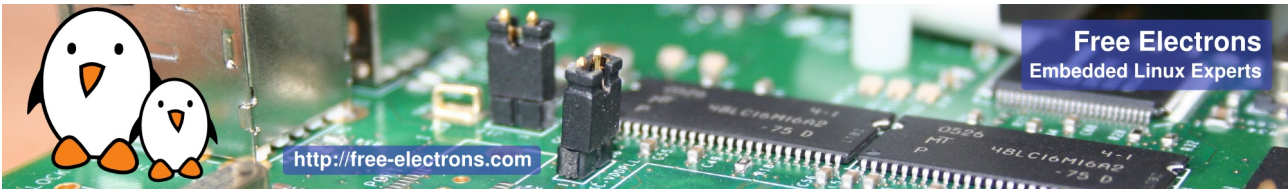
Day 2 - Morning

<p>Lab - Kernel sources</p>	<p>Lecture - Configuring and compiling a Linux kernel</p>
<p>Downloading kernel sources Apply kernel patches</p>	<p>Kernel configuration. Useful settings for embedded systems. Native compiling. Generated files. Using kernel modules</p>

<p>Lecture - Kernel cross-compiling</p>	<p>Lab - Kernel cross-compiling and booting</p>
<p>Kernel cross-compiling setup. Using ready-made configuration files for specific architectures and boards. Cross-compiling Linux.</p>	<p><i>Using the IGEPv2 ARM board</i> 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.</p>

Day 2 - Afternoon

<p>Lecture - Root filesystem in Linux</p>	<p>Lecture - BusyBox</p>
<p>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.</p>	<p>Detailed overview. Detailed features. Configuration, compiling and deploying.</p>



Lab - Tiny root filesystem built from scratch with BusyBox

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 developing a sample application.

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 IGEP 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.

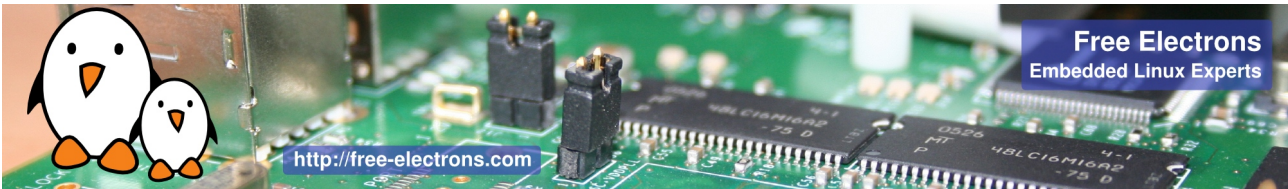
Day 3 - Afternoon

Lecture - Flash filesystems

The Memory Technology Devices (MTD) filesystem.
Filesystems for MTD storage: JFFS2, Yaffs2, UBIFS.
Kernel configuration options
MTD storage partitions.
Mounting MTD filesystem images.

Lab - Flash filesystems

Using the IGEPv2 ARM board
Creating partitions in your internal flash storage.
Formatting the main partition with JFFS2 in read-only mode.
Using JFFS2 for system data.



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, DirectFB, Gtk, Qt, etc.), system utilities, network libraries and utilities, multimedia libraries, etc.

Example of a typical consumer electronic product leveraging many open-source components.

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 a graphical system based on DirectFB, running in Qemu.

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

Building a system with a graphical system based on DirectFB, running in Qemu.

Using Buildroot to rebuild the same system as in the previous lab.

Seeing how easier it gets.

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.

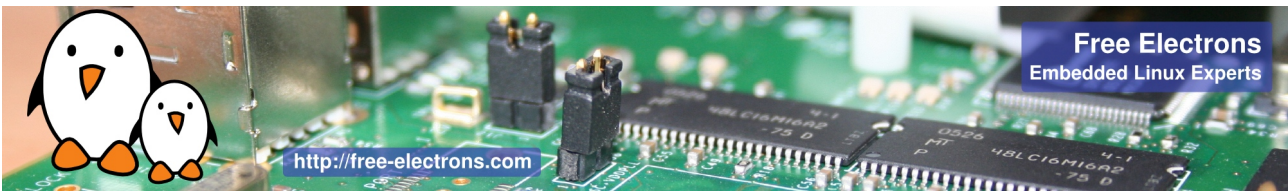
Build system for your application, how to use existing libraries in your application.

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.

Developing on Windows.



Lab - Application development and debugging

In Qemu and on the IGEP ARM board.

Develop and compile an application relying on the DirectFB 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 brought by Linux 2.6.

Understanding and using the latest RT preempt patches for mainstream Linux.

Real-time kernel debugging. Measuring and analyzing latency.

Xenomai, a hard real-time solution for Linux: features, concepts, implementation and examples.

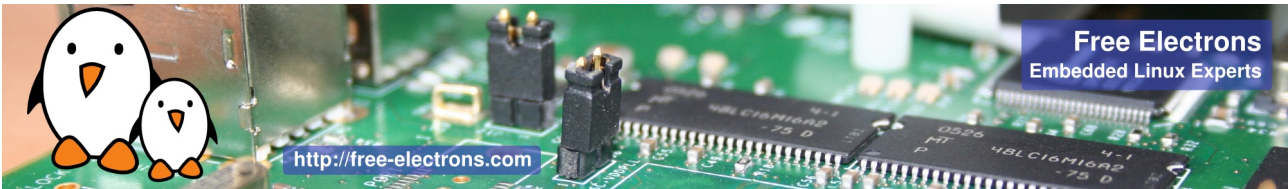
Lab - Linux latency tests

Tests performed on the IGEPv2 ARM board

Latency tests on standard Linux.

Setting up Xenomai.

Latency tests with Xenomai.



Additional topics

These topics can be covered at the end of the training if enough time is left.

Lecture - Hotplugging	Lab - Hotplugging
<p>Udev: handling hardware events from user-space: creating and removing device files, identifying drivers, notifying programs and users.</p> <p>Using BusyBox mdev, a simpler implementation.</p>	<p>Using BusyBox mdev to populate the /dev directory with all available devices.</p> <p>Adding rules for mdev.</p> <p>Making mdev automatically mount the partitions of an external USB disk when it is inserted.</p>