# Embedded Linux kernel and driver development training

**On-line seminar, 7 sessions of 4 hours**

<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Embedded Linux kernel and driver development training</th>
</tr>
</thead>
</table>
| **Overview** | Understanding the Linux kernel  
Developing Linux device drivers  
Linux kernel debugging  
Porting the Linux kernel to a new board  
Working with the kernel development community  
Practical demos with the ARM-based BeagleBone Black board (or with its Wireless variant). |
| **Materials** | Check that the course contents correspond to your needs:  
| **Duration** | Seven half days - 28 hours (4 hours per half day).  
80% of lectures, 20% of practical demos. |
| **Trainer** | One of the engineers listed on  
https://bootlin.com/training/trainers/ |
| **Language** | Oral lectures: English  
Materials: English. |
| **Audience** | People developing devices using the Linux kernel  
People supporting embedded Linux system developers. |
| **Prerequisites** | **Familiarity with C programming**  
In particular, participants should understand complex data types and structures, pointers and function pointers.  

**Familiarity with UNIX or GNU/Linux commands**  
People lacking experience on this topic could get trained by themselves, for example with our freely available on-line slides (https://bootlin.com/blog/command-line/).  

**Familiarity with embedded Linux development**  
Taking our Embedded Linux course (https://bootlin.com/training/embedded-linux/) first is not a requirement, but it will definitely allow to understand the development environment and board manipulations, allowing to concentrate on kernel code programming. |
## Required equipment

- 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

## Materials

Electronic copies of presentations, demo instructions and data.

## 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 to illustrate the development of Linux device drivers:

- A Wii Nunchuk, which is connected over the I2C bus to the BeagleBone Black board. Its driver will use the Linux `input` subsystem.
- An additional UART, which is memory-mapped, and will use the Linux `misc` subsystem.

While our explanations will be focused on specifically the Linux subsystems needed to implement these drivers, they will always be generic enough to convey the general design philosophy of the Linux kernel. The information learnt will therefore apply beyond just I2C, input or memory-mapped devices.
### Half day 1

<table>
<thead>
<tr>
<th>Lecture - Introduction to the Linux kernel</th>
<th>Lecture - Kernel sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Kernel features</td>
<td>• Specifics of Linux kernel development</td>
</tr>
<tr>
<td>• Legal constraints with device drivers.</td>
<td>• Coding standards</td>
</tr>
<tr>
<td>• Kernel user interface (/proc and /sys)</td>
<td>• Retrieving Linux kernel sources</td>
</tr>
<tr>
<td>• User space device drivers</td>
<td>• Tour of the Linux kernel sources</td>
</tr>
<tr>
<td></td>
<td>• Kernel source code browsers: cscope, Elixir Cross Referencer, VS Code...</td>
</tr>
</tbody>
</table>

### Demo - Kernel sources

- Making searches in the Linux kernel sources: looking for C definitions, for definitions of kernel configuration parameters, and for other kinds of information.
- Using the UNIX command line and then kernel source code browsers.

### Half day 2

<table>
<thead>
<tr>
<th>Lecture - Configuring, compiling and booting the Linux kernel</th>
<th>Demo - Kernel configuration, cross-compiling and booting on NFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Kernel configuration.</td>
<td>Using the BeagleBone Black board</td>
</tr>
<tr>
<td>• Native and cross compilation. Generated files.</td>
<td>• Configuring, cross-compiling and booting a Linux kernel with NFS boot support.</td>
</tr>
<tr>
<td>• Booting the kernel. Kernel booting parameters.</td>
<td></td>
</tr>
<tr>
<td>• Mounting a root filesystem on NFS.</td>
<td></td>
</tr>
</tbody>
</table>
## Lecture - Linux kernel modules

- Linux device drivers
- A simple module
- Programming constraints
- Loading, unloading modules
- Module dependencies
- Adding sources to the kernel tree

## Demo - Writing modules

*Using the BeagleBone Black board*
- Write a kernel module with several capabilities.
- Access kernel internals from your module.
- Set up the environment to compile it

---

## Half day 3

---

## Lecture - Linux device model

- Understand how the kernel is designed to support device drivers
- The device model
- Binding devices and drivers
- Platform devices, Device Tree
- Interface in user space: `/sys`

## Demo - Linux device model for an I2C driver

*Using the BeagleBone Black board*
- Implement a driver that registers as an I2C driver
- Modify the Device Tree to list an I2C device
- Get the driver called when the I2C device is enumerated at boot time

---

## Lecture - Introduction to the I2C API

- The I2C subsystem of the kernel
- Details about the API provided to kernel drivers to interact with I2C devices
### Half day 4

#### Lecture - Pin muxing
- Understand the *pinctrl* framework of the kernel
- Understand how to configure the muxing of pins

#### Demo - Communicate with the Nunchuk over I2C
*Using the BeagleBone Black board*
- Configure the pin muxing for the I2C bus used to communicate with the Nunchuk
- Extend the I2C driver started in the previous lab to communicate with the Nunchuk via I2C

#### Lecture - Kernel frameworks
- Block vs. character devices
- Interaction of user space applications with the kernel
- Details on character devices, *file_operations*, `ioctl()`, etc.
- Exchanging data to/from user space
- The principle of kernel frameworks

#### Lecture - The input subsystem
- Principle of the kernel *input* subsystem
- API offered to kernel drivers to expose input devices capabilities to user space applications
- User space API offered by the *input* subsystem

#### Demo - Expose the Nunchuk functionality to user space
*Using the BeagleBone Black board*
- Extend the Nunchuk driver to expose the Nunchuk features to user space applications, as a *input* device.
- Test the operation of the Nunchuk using `evtest`
## Lecture - Memory management

- Linux: memory management - Physical and virtual (kernel and user) address spaces.
- Linux memory management implementation.
- Allocating with `kmalloc()`.
- Allocating by pages.
- Allocating with `vmalloc()`.

## Lecture - I/O memory

**Demo - Minimal platform driver and access to I/O memory**

*Using the BeagleBone Black board*
- Implement a minimal platform driver
- Modify the Device Tree to instantiate the new serial port device.
- Reserve the I/O memory addresses used by the serial port.
- Read device registers and write data to them, to send characters on the serial port.

- I/O memory range registration.
- I/O memory access.
- Read / write memory barriers.

## Lecture - The misc kernel subsystem

**Demo - Output-only serial port driver**

*Using the BeagleBone Black board*
- Extend the driver started in the previous lab by registering it into the `misc` subsystem
- Implement serial port output functionality through the `misc` subsystem
- Test serial output from user space

- What the `misc` kernel subsystem is useful for
- API of the `misc` kernel subsystem, both the kernel side and user space side
<table>
<thead>
<tr>
<th>Lecture - Processes, scheduling, sleeping and interrupts</th>
<th>Demo - Sleeping and handling interrupts in a device driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Process management in the Linux kernel.</td>
<td>• Adding read capability to the character driver developed earlier.</td>
</tr>
<tr>
<td>• The Linux kernel scheduler and how processes sleep.</td>
<td>• Register an interrupt handler.</td>
</tr>
<tr>
<td>• Interrupt handling in device drivers: interrupt handler registration and programming, scheduling deferred work.</td>
<td>• Waiting for data to be available in the <code>read()</code> file operation.</td>
</tr>
<tr>
<td></td>
<td>• Waking up the code when data is available from the device.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lecture - Locking</th>
<th>Demo - Locking</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Issues with concurrent access to shared resources</td>
<td>• Add locking to the current driver</td>
</tr>
<tr>
<td>• Locking primitives: mutexes, semaphores, spinlocks.</td>
<td></td>
</tr>
<tr>
<td>• Atomic operations.</td>
<td></td>
</tr>
<tr>
<td>• Typical locking issues.</td>
<td></td>
</tr>
<tr>
<td>• Using the lock validator to identify the sources of locking problems.</td>
<td></td>
</tr>
</tbody>
</table>

*Using the BeagleBone Black board*
### Half day 7

#### Lecture - Driver debugging techniques
- Debugging with printing functions
- Using Debugfs
- Analyzing a kernel oops
- Using kgdb, a kernel debugger
- Using the Magic SysRq commands

#### Lecture - Power management
- Overview of the power management features of the kernel
- Topics covered: clocks, suspend and resume, dynamic frequency scaling, saving power during idle, runtime power management, regulators, etc.

#### Lecture - ARM board support and SoC support
- Understand the organization of the ARM support code
- Understand how the kernel can be ported to a new hardware board

#### Demo - Investigating kernel faults

*Using the BeagleBone Black board*
- Studying a broken driver.
- Analyzing a kernel fault message and locating the problem in the source code.

#### Lecture - The Linux kernel development process
- Organization of the kernel community
- The release schedule and process: release candidates, stable releases, long-term support, etc.
- Legal aspects, licensing.
- How to submit patches to contribute code to the community.
- Kernel resources: books, websites, conferences
<table>
<thead>
<tr>
<th>Lecture - If time left</th>
<th>Questions and Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>• DMA</td>
<td>• Questions and answers with the audience about the course topics</td>
</tr>
<tr>
<td>• mmap</td>
<td>• Extra presentations if time is left, according what most participants are interested in.</td>
</tr>
</tbody>
</table>

Possible extra time

Extra time (up to 4 hours) may be proposed if the agenda didn’t fit in 7 half days, according to the time spent answering questions from participants.