



# Linux on TI OMAP processors Training lab book

*Michael Opdenacker*  
*Free Electrons*  
<http://free-electrons.com>





## About this document

This document is part of an embedded Linux training from Free Electrons. It actually replaces Lab 3 in the lab book of our Embedded Linux kernel and driver development training, with an OMAP specific lab.

You will find the whole training materials (slides, lab book and lab data) on <http://free-electrons.com/articles/omap>

## Copying this document

© 2005, Michael Opdenacker, [michael@free-electrons.com](mailto:michael@free-electrons.com)



This document is released under the terms of the [Creative Commons Attribution-ShareAlike 2.0 license](https://creativecommons.org/licenses/by-sa/2.0/). This means you are free to download, distribute and even modify it, under certain conditions.

Document updates and translations available on <http://free-electrons.com/articles/omap>

Corrections, suggestions, contributions and translations are welcome!

## Document history

Unless specified, contributions are from the original author, Michael Opdenacker.

Sep 15, 2009. Latest update. Corrections and minor improvements.

June 26, 2005. First public release.

## Training setup

Instructions for the instructor, or for self training people.

See lab instructions on <http://free-electrons.com/training/drivers>. This lab replaces Lab 3 in that generic course.



## Lab 1 – Cross-compiling

Objective: Learn how to cross-compile a kernel for another target platform: OMAP 2420 H4 board.

After this lab, you will be able to

- Set up a cross-compiling environment
- Configure the kernel Makefile accordingly
- Cross compile the kernel for an arm target platform
- Postprocess kernel and initrd files for U-boot
- Configure a tftp server
- Download the kernel and initrd files on the board RAM and eventually boot your home-made Linux kernel!

### Prerequisites

This lab should be run as a replacement of Lab 3 in our embedded Linux kernel and drivers training, available on <http://free-electrons.com/training/drivers>. So, you should follow this training first!

To run this lab, you need an OMAP 2420 H4 development board. Of course, these instructions can be easily adapted to different OMAP based hardware.

### Setup

Go to the `/mnt/labs/omap/lab1` directory.

### Getting the sources

Get the latest sources of the Linux kernel for the H4 board from <http://linux.omap.com/>

### Cross-compiling environment setup

To cross-compile Linux, you need to add the cross-compiling toolchain to your `PATH` environment variable:

```
export PATH=/usr/local/arm-unknown-linux-gnu/gcc-3.4.2-glibc-2.2.5/bin:$PATH
```

### Makefile setup

Modify the toplevel Makefile file to cross-compile for the arm platform using the above toolchain.

Also enable the use of `ccache` to speed up recompiling.

### Linux kernel configuration

Pick up the default Linux 2.6 configuration for OMAP H4 2420 released inside the kernel sources.

Don't hesitate to visualize the new settings by running `make xconfig` afterwards!

Note that the default gcc 3.3.2 toolchain on <http://linux.omap.com> didn't manage to compile the linux-2.9 kernel supplied on this site.



## Cross compiling

Try to compile your kernel. You will soon see whether there are issues in your setup or not.

With 2.6.9 and the default config file, you should face an error. Find the corresponding configuration setting. Go back to the configuration interface and switch this setting off.

Once compiling is successful, where is the compressed Linux kernel image created?

## Interacting with the bootloader

We are now going to start to interact with the hardware. The U-boot bootloader is supposed to have been installed on the OMAP 2420 boards.

Connect your PC to the serial port of the debug board.

Make the serial port device (usually `/dev/ttyS0`) readable and writable by all users.

From the `knoppix` user, run `minicom -s` to configure the `minicom` terminal emulator. Make sure that the right serial port is selected, and that serial port settings are set to `115200 8N1`. Save your setup as the default settings, and choose `Exit`, to exit the configuration mode and start using `minicom`.

Power up the board. Do you get the messages and the prompt from U-boot on the serial terminal? If you don't, try to find the reason why. Don't hesitate to ask your instructor for help!

Type `help` and take your time to review the commands available in the U-boot prompt.

## Preparing boot images

With the Linux Makefile, generate a kernel image file suitable for das U-boot.

Pickup an `ext2` ramdisk image from the OMAP Linux community site.

Compress this image with `gzip` and post process it to make it usable by U-boot. Name this file `uInitrd` (just an example name).

## Setting up a tftp server

Copy the U-boot compatible kernel and ramdisk images to the root directory of the tftp server configured on your workstation.

Enable the tftp server and start it.

See the OMAP / U-boot slides for details!

## Configuring the bootloader for network boot

Connect the ethernet port of the board to the network.

If your network uses a dhcp server, run the below 2 commands in the U-boot command line to learn more about valid network settings:

```
setenv ethaddr xx:xx:xx:xx:xx:xx
dhcp
[Ctrl C] (to interrupt)
```

Of course, any means of finding the file to display is valid!

Though U-boot supports loading and booting a kernel image through the serial port (using the kermit protocol), this is a very slow way compared to downloading images from a tftp server on the network.

`ethaddr` is the MAC address of the board's network adapter. It is supposed to be written on a label on the back of the multiport debug board.



Now that you know more about valid network settings, configure U-boot accordingly. Change the below values to valid ones:

```
setenv ipaddr 192.168.1.102
setenv gatewayip 192.168.1.1
setenv netmask 255.255.255.0
```

Find the IP address of your workstation (which is running the tftp server), and configure U-boot accordingly as in the below example:

```
setenv serverip 192.168.1.101
```

Now configure the kernel command line for a Linux boot from RAM:

```
setenv bootargs root=/dev/ram0 rw console=ttyS0,115200n8
init=/bin/sh (in just 1 line)
```

Save these settings to flash to avoid having to configure them over and over again!

### Downloading images and booting

You are ready to go!

```
tftpboot 80000000 uImage
tftpboot 84000000 uInitrd
bootm 80000000 84000000
```

### Interacting with the target through a serial console

If everything goes right, and if you didn't make any mistake, your kernel should boot fine!

Great job!

The U-boot command line help is your friend.

Other valid addresses may be used, of course. Unfortunately, I haven't found any official recommendations yet.