# ARM Linux specifics
# Training lab book

*Thomas Petazzoni*
*Free Electrons*
*http://free-electrons.com*

## About this document

This document is part of an embedded Linux training from Free Electrons.

You will find the whole training materials (slides and lab book)
on http://free-electrons.com/docs/arm-linux.

Lab data can be found on http://free-electrons.com/labs/embedded_linux.tar.bz2.

## Copying this document



Document updates and translations available on http://free-electrons.com/docs/arm-linux.

Corrections, suggestions, contributions and translations are welcome!

## Training setup

See the training labs on http://free-electrons.com/docs/kernel for setup instructions, which are shared with these practical labs.

# Lab 1 – Producing ARM Thumb binaries with Scratchbox

Objective: Set up a cross-compiling toolchain able to produce ARM Thumb binaries and that supports interworking, and use it to generate a DirectFB based demonstration

---

After this lab, you will be able to

- Create a cross-compilation toolchain supporting interworking with Buildroot
- Integrate this toolchain inside Scratchbox
- Use it to compile a DirectFB demonstration

## Setup

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

Make sure you have at least 2 GB of free disk space and that Scratchbox is properly installed on your system. If not, follow the instructions of the previous Scratchbox lab.

Install the `libncurses5-dev` package (needed for the Buildroot configuration tool).

## Set up an host target in Scratchbox

Install the `scratchbox-toolchain-host-gcc` package, which contains a host compiler for Scratchbox and the `scratchbox-devkit-debian` package.

Starting from now, the following steps are done from Scratchbox, so you should log into Scratchbox now.

Create an HOST target:

```
sb-conf setup HOST --compiler=host-gcc \
        --devkits=debian-etch
```

Install the devkit and /etc files to the target:

```
sb-conf install HOST --devkits --etc
```

Select the new target:

```
sb-conf select HOST
```

## Generating the cross-compilation toolchain with Buildroot inside Scratchbox

The goal of this step is to generate the cross-compilation toolchain that we will then integrate to the Scratchbox build system. In order for this toolchain to work properly under Scratchbox, it has to be compiled in the Scratchbox environment.

Download our Buildroot snapshot from http://www.free-electrons.com/labs/tools/. This lab was designed for `buildroot-20080326.tar.bz2`. After download, uncompress the Buildroot tarball inside your Scratchbox home directory `/scratchbox/users/<user>/home/<user>/`, and apply the `data/thumb-interworking-support.patch patch` to Buildroot.

The Thumb support for Buildroot has been contributed by Free Electrons. The patch has been sent for inclusion into the official version of Buildroot.

Run the Buildroot configuration tool outside Scratchbox:

```
make menuconfig
```

Make the following configuration choices:

- Target architecture variant: `arm720t`
- Target ABI: `EABI`
- Toolchain and header file location: `/scratchbox/compilers/arm-linux-gcc4.2-uclibc0.9.29-interwork`
- Number of jobs to run simultaneously: 2
- strip : sstrip
- Kernel headers: `Latest Linux 2.6.23.x kernel headers`
- Binutils version: `binutils 2.18.50.0.1`
- Install `sstrip` for the target system: `yes`
- **Use software floating point by default: `yes`**
- **Enable interworking support: `yes`**
- Generate Thumb binaries: `yes`
- Include target utils in cross toolchain: `no`
- Package selection for the target: *unselect all*
- Target filesystem options: *unselect all*
- **Kernel type: `none`**

The ncurses library, used by the configuration tool is unfortunately not available inside Scratchbox.

Your instructor will give the URL of a local webserver that contains tarballs of various tools built by Buildroot. Download them, and put them in a directory named `dl/` inside Buildroot. It will save the download time, which can be huge considering the size of the tarballs.

Change the owner of `/scratchbox/compilers/` so that Buildroot can write to it:

```
sudo chown <user> /scratchbox/compilers
```

Before starting the compilation, we need to compile texinfo inside Scratchbox, because it is used by the toolchain compilation process. Get the texinfo tarball from the location given by your instructor, uncompress it inside Scratchbox, and then run the usual `./configure ; make ; make install`.

Texinfo is available as part of the `scratchbox-devkit-doctools` package, but unfortunately, the version available in this package is too old to compile a recent Binutils.

Finally, compile the toolchain inside Scratchbox by running `make`.

If something fails during the compilation and you want to start again from scratch, remove the `build_arm`, `project_build_arm` and `toolchain_build_arm` directories.

## Playing with the generated toolchain

Outside Scratchbox, write a simple « Hello World » program as follows:

```
#include <stdio.h>

int main(void) {
      printf("Hello World\n");
      return 0;
}
```

Make sure the toolchain is available in your PATH:

```
export PATH=/scratchbox/compilers/arm-linux-gcc4.2-uclibc0.9.29-interwork/usr/bin
```

Compile the program in Thumb mode for ARM, statically:

```
arm-linux-gcc -o test test.c -mthumb -static
```

You can run this program with Qemu user emulation:

```
qemu-arm test
```

You can check that the program code is compiled in Thumb mode and the library code in ARM mode by looking at the disassembled version of the program:

```
arm-linux-objdump -d test
```

The code of the `main()` function is in Thumb mode, while the uClibc code is in ARM mode.

### Integrating the toolchain inside Scratchbox

The goal of integrating the toolchain into Scratchbox is to allow the creation of a Scratchbox target that uses our toolchain to cross-compile libraries and applications.

The procedure is documented at http://www.scratchbox.org/wiki/ForeignToolchains. We reproduce the procedure below, with a few modifications to get it working properly in our configuration.

First, make sure your user has write permission on `/scratchbox/device_tools`, preferably by changing its owner to your user.

Download the toolchain integration scripts for Scratchbox tarball `sb-toolchain-extras.tar.gz` from the location given by your instructor. Uncompress it inside Scratchbox, in your home directory.

Usually, these tools are fetched from a Darcs repository, using the `darcs` tool. We provide a tarball to speed up the download process.

Apply the `data/sb-toolchain-extras-uclibc-fix.patch` patch inside `sb-toolchain-extras`. The patch contains a small fix for the *fakeroot* tool to allow it to be compiled with an uClibc without large file support.

Download the tools needed by the toolchain integration scripts, available as a `sb-toolchain-extras-downloads.tar.gz` tarball at a location given by your instructor.

These files are automatically downloaded by the compilation process. But to speed up the process, we provide a tarball with the needed files.

We then need to do some minor fixes to the toolchain generated by Buildroot. Everything must be done from the toolchain directory, `/scratchbox/compilers/arm-linux-gcc4.2-uclibc0.9.29-interwork/.`

- Create a symbolic link `arm-linux-uclibcgnueabi/` to `usr/arm-linux-uclibcgnueabi`
- Create a symbolic link `include/` to `usr/arm-linux-uclibcgnueabi/include/`
- Inside `bin/`, create symbolic links using the following command:
  ```
  for i in ../usr/bin/arm-linux-uclibcgnueabi-* ; do
  ln -s $i ; done
  ```

Still from the toolchain directory, run the following command to create a configuration file for the toolchain:

```
~/sb-toolchain-extras/confhelper/create_toolchain_conf.py
> ~/sb-toolchain-extras/meta/alien-tc/arm-linux-gcc4.2-
uclibc0.9.29-interwork.conf
```

Edit this file to modify the following variables:

- `COMPILER_PACKAGE`, that should be set to `scratchbox-toolchain-arm-linux-gcc4.2-uclibc0.9.29-interwork`. This will be the name of the tarball or Debian package containing the toolchain
- `VENDOR`, that should be empty
- `BINUTILS_FULLVER`, which should be set to `2.18.50.0.1.20070908`

Go back to the `~/sb-toolchain-extras/` directory, and run the following commands to integrate the toolchain into Scratchbox:

```
make CONFIG=meta/alien-tc/arm-linux-gcc4.2-uclibc0.9.29-interwork.conf -C meta/alien-tc all-sums
```

```
make CONFIG=meta/alien-tc/arm-linux-gcc4.2-uclibc0.9.29-interwork.conf -C meta/alien-tc
```

After these steps, the new toolchain should be visible using the command:

```
sb-conf list --compilers
```

### Configuring the target for Thumb compilation

Create a new target with the new compiler:

```
sb-conf setup armthumb
  --compiler=arm-linux-gcc4.2-uclibc0.9.29-interwork
  --devkits=cputransp
  --cputransp=/scratchbox/devkits/cputransp/bin/qemu-arm-0.8.2-sb2
```

Then, select this target:

```
sb-conf select armthumb
```

And install the necessary files:

```
sb-conf install armthumb --clibrary --etc
```

In your home directory, create an Hello World program, and compile it simply with gcc. It should run properly, and if you look at the binary with `objdump -S`, the `main()` function should be in Thumb mode.

### Creating a Debian package containing the toolchain

Optionally, you can create a Debian package containing the toolchain, if you want to distribute it to others. Simply run the following command from `~/sb-toolchain-extras/`:

```
make CONFIG=meta/alien-tc/arm-linux-gcc4.2-uclibc0.9.29-interwork.conf deb
```

### Compile the DirectFB demonstration

Create the /scratchbox/users/<user>/targets/armthumb.environment file with the following contents:
`export CFLAGS=-mthumb`

It will allow to automatically compile programs and libraries in Thumb mode.

Then, take the instructions from the Scratchbox lab to recompile the

DirectFB demonstration inside this new Scratchbox target. Use the provided `run_qemu` script to start Qemu, after configuring NFS to make it export the target root filesystem directly from the Scratchbox directory.