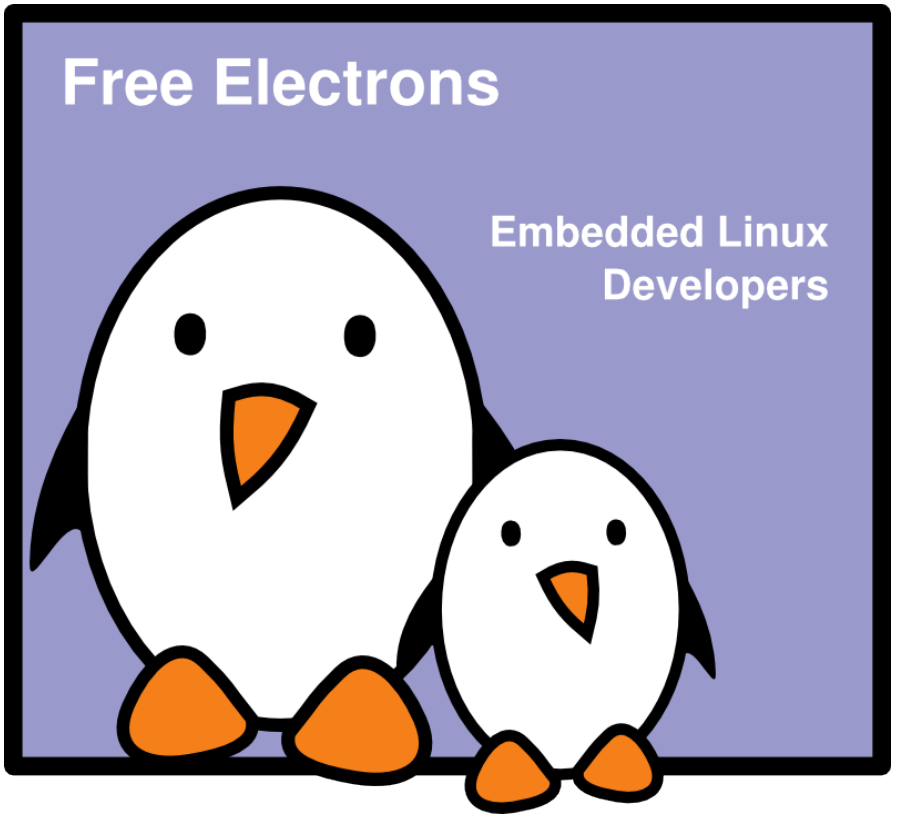
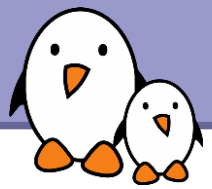
The Scratchbox development environment

Michael Opdenacker
Thomas Petazzoni
Free Electrons



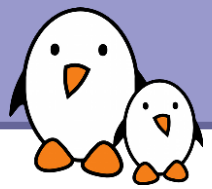


Rights to copy

Document sources, updates and translations:
http://free-electrons.com/docs/scratchbox

Corrections, suggestions, contributions and translations are welcome!

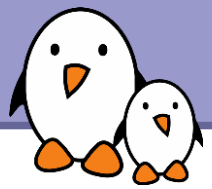
Latest update: Sep 15, 2009



Scratchbox

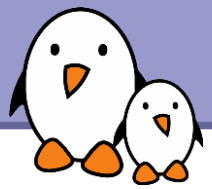
http://www.scratchbox.org

- Cross-compiling toolkit designed to make embedded Linux application development faster and easier.
- Provide a sandbox environment that emulates some characteristics of the target system
  - Dependencies are not mixed with host system's libraries
  - Transparent cross-compiling, making building tools believe they are doing a native compile job : no tweaking to support cross-compiling, and fast compiling on a cheap `x86` box.
- Supported architectures : `arm` and `x86`.
  Experimental support for `cris`, `mips` and `ppc`.



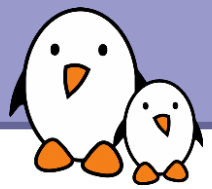
History

- Research started in 2002
- First public release in 2003
- Scratchbox 1.0 in February 2005, « Apophis »
  - Still maintained, with regular updates to various components from 2005 to 2008.
- Scratchbox 2.0
  - In development : version 1.99.0.23 released in February 2008.
- Developed by Movial, sponsored by Nokia.
- Used for the Maemo environment of the Nokia Internet tablets.



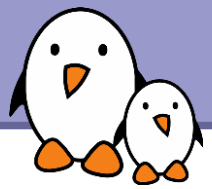
Features

- Chrooted environment
  - Running on the host, but only target files are visible.
- Transparent cross-compiling
- Transparent execution, either through Qemu or remote execution using `sbrsh`.
- Comes with ready-made cross-compiling toolchains and tools to build Debian packages.
- Supports both uClibc and glibc.
- Provides basic root filesystems.



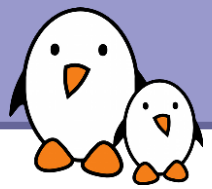
Installing Scratchbox

- On Debian/Ubuntu, add the Scratchbox repository to the package sources:
  `deb http://scratchbox.org/debian/ apophis main`
- On other distributions, download binary tarballs from
  http://scratchbox.org/download/files/sbox-releases/apophis/tarball/
- Install the following packages
  - `scratchbox-core`
  - `scratchbox-libs`
  - A toolchain package, for example
    `scratchbox-toolchain-arm-gcc4.1-uclibc20061004`
  - CPU transparency development kit
    `scratchbox-devkit-cputransp`



Configuring Scratchbox

- Scratchbox is installed in `/scratchbox`
- Add your account to the Scratchbox system
  `sudo /scratchbox/sbin/sbox_adduser <user>`
- This will
  - Add your user to the `sbox` group
  - Create files and directories inside the Scratchbox system for your user
- Need to log out and log in again for the group change to take effect
- Login using `/scratchbox/login`
- Scratchbox says « No current target »



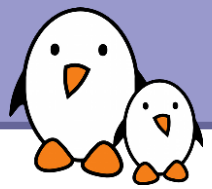
Configuring a target

- Two configuration tools
  - `sb-conf`, command line
  - `sb-menu`, semi-graphical curses interface
- Using sb-conf

```
sb-conf setup armdemo
  --compiler=arm-gcc4.1-uclibc20061004
  --devkits=cputransp
  --cputransp=/scratchbox/devkits/cputransp/bin/
          qemu-arm-0.8.2-sb2
```

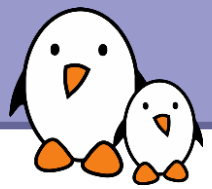
- Select the new target

```
sb-conf select armdemo
```



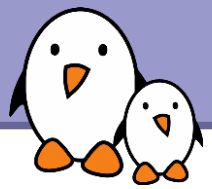
Exploring the target

- The prompt is now
  `[sbox-armdemo: ~] >`
- Inside the `armdemo` target, in your home directory
- Your target root filesystem is stored in `/targets/armdemo`
  - A set of symbolic links from `/` allows to think that you are actually running on the target
  - Some host tools, provided by Scratchbox, are still available
- The target root filesystem is empty. Let's ask to fill it with the C library, headers and basic /etc files
  `sb-conf install armdemo -c -e`
- Can also be done with `sb-menu`



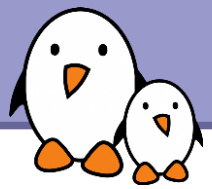
Using Scratchbox

- Test a simple program provided by Scratchbox
- Extract
```
tar xfz /scratchbox/packages/hello-world.tar.gz
```
- Configure and compile
```
cd hello-world
./autogen.sh
make
```
- Check that the program is compiled for ARM
```
file hello
hello: ELF 32 bit LSB executable, ARM [...]
```
- Run it: `./hello`



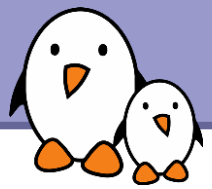
Using Scratchbox (2)

- Possible to cross-compile and install libraries in a transparent way
  - `./configure`
  - `make`
  - `make install`
- And then, to cross-compile programs using these libraries.
- Cross-compiling is a lot easier.



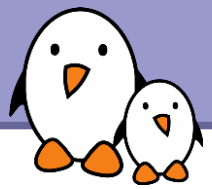
How Scratchbox works (1)

- The user is chrooted into `/scratchbox/users/<user>/`
- This directory looks like a regular root filesystem
  - Most directories are symbolic links to `target/links/<dir>`
  - These are again symbolic links to `target/<target>/<dir>`
  - They are switched when changing the target
  - The home directory in `/scratchbox/users/<user>/home/` is not target-specific
- Various host directories are remounted inside the target using the `--bind` option of `mount`: `/scratchbox, /tmp, /proc, /dev/, /dev/pts, /dev/shm, /sys`



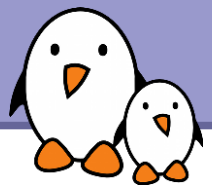
How Scratchbox works (2)

- Target root filesystem is stored in `/scratchbox/users/thomas/targets/<target>`
- Contains the filesystem hierarchy that should be used on the embedded devices
- Configuration file stored in `/scratchbox/users/thomas/targets/<target>.config`
  - Defines the architecture, CPU transparency method, cross-compiler, compiler and linker options, host compiler, host compiler options...
  - Many other variables can be defined to configure the target



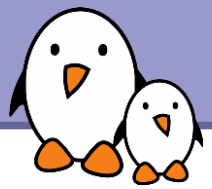
How Scratchbox works (3)

- Toolchain binaries are executed through a wrapper
- The `gcc` binary is a symlink to `sb_gcc_wrapper`, which runs the correct compiler depending on the target configuration.
- Build systems think that they are building natively.
- Outside of Scratchbox, the toolchain can be used in a normal way (`ARCH-linux-gcc`, etc.)



How Scratchbox works (4)

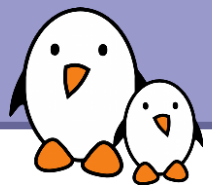
- Host tools take precedence over target binaries
- Host tools are hardwired to use libraries in `/scratchbox/host_shared/`
- `PATH` is set so that host binaries are used in preference over target binaries, but it is not enough for absolute paths.
- Scratchbox uses a technique called *binary redirection.*
- Using `LD_PRELOAD`, some libc functions are overriden
  - `exec()` and friends
  - `uname()` so that the target architecture is correctly returned.
  - etc.



How Scratchbox works (5)

CPU transparency

- Execute target binaries transparently on the host
- Uses the kernel `binfmt_misc` facility to run an interpreter when a target binary is run.
  - See `/proc/sys/fs/binfmt_misc/` for its configuration.
- The interpreter can then
  - Use qemu user emulation to run the binary
  - Use `sbrsh` to execute the binary directly on the target device using a network connection.

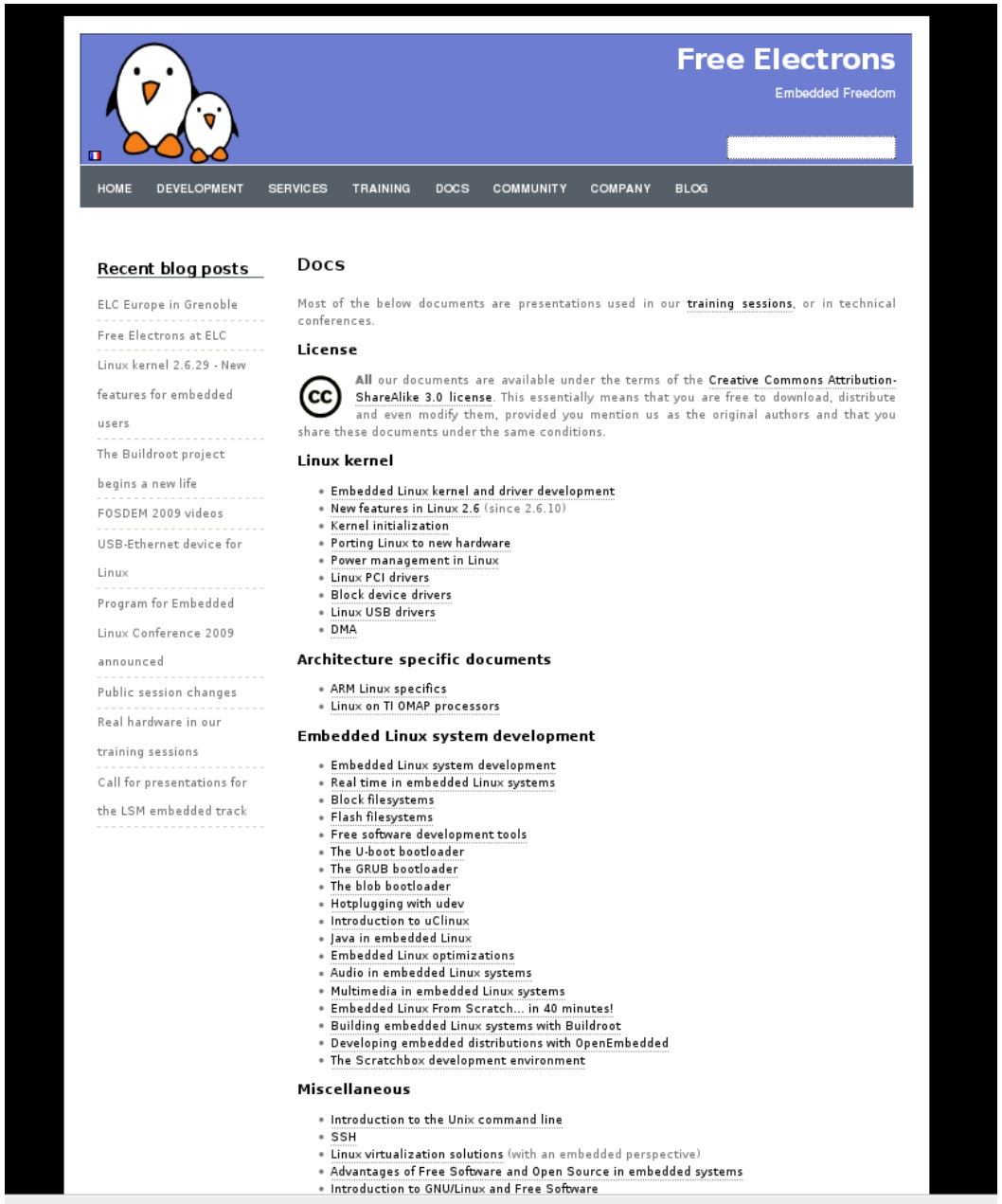


References

- Scratchbox
  http://www.scratchbox.org
- Scratchbox: Cross-compiling a Linux distribution
  http://www.embedded-kernel-track.org/2005/scratchbox-fosdem20

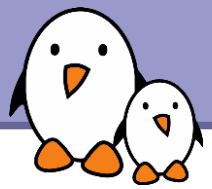
  FOSDEM 2005, Brussels
- Bringing Cross-Compiling to Debian
  http://nchipin.kos.to/debconf-sbox2.pdf
  Debconf 6, Mexico.

Related documents



All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs