Buildroot: a nice, simple and efficient embedded Linux build system

Thomas Petazzoni

Bootlin

thomas.petazzoni@bootlin.com
Thomas Petazzoni

- Embedded Linux engineer and trainer at **Bootlin** since 2008
  - **Embedded Linux development**: kernel and driver development, system integration, boot time and power consumption optimization, consulting, etc.
  - **Embedded Linux training**, Linux driver development training and Android system development training, with materials freely available under a Creative Commons license.
  - [https://www.bootlin.com](https://www.bootlin.com)

- **Major contributor to Buildroot**, an open-source, simple and fast embedded Linux build system

- **Speaker** at Embedded Linux Conference, Embedded Linux Conference Europe, FOSDEM, Libre Software Meeting, etc.

- Living in Toulouse, south west of France
To create an embedded Linux system, one has multiple choices:

1. Use a **pre-built binary distribution** such as Debian, Ubuntu or Fedora
   - Quick to set up, but not very flexible: support for only a few architectures, no flexibility on package configuration, no easy way to rebuild the entire system automatically.

2. Build **all system components manually**
   - Highly flexible, but painful and inefficient: need to handle complex cross-compilation issues, understand inter-package dependencies, not reproducible.

3. Use an **automated build system**, that builds the entire system from source
   - Automated, flexible, handle most cross-compilation issues
   - Examples: Buildroot, OpenWRT, PTXdist, OpenBricks, OpenEmbedded, Yocto, etc.
Buildroot main characteristics

- Based on **well-known technologies**: `kconfig` for the configuration interface and language, `make` for the build logic. These technologies are familiar to all embedded Linux developers.

- **Very simple to use**, and **easily hackable** code base. The core infrastructure is a few hundred lines of `make` code.

- **Fast**. It does really build only what’s necessary. The base system, composed only of BusyBox, takes less than 3 minutes to build with an external toolchain.

- **Designed for small to medium sized embedded systems**. There is no runtime package management system (dpkg, rpm). Complete rebuilds are often required. Well-suited for systems with a limited number of components.

- [Visit Bootlin](https://bootlin.com) for kernel, drivers and embedded Linux - Development, consulting, training and support.
Buildroot vs. OpenEmbedded/Yocto

Root filesystem generator

Distribution generator

Buildroot

OpenEmbedded / Yocto

Number of software components

1 10 100 1000

No runtime package management needed. Upgrades are done system-wide. Fixed-functionality systems, like industrial systems.

Runtime package management needed. Upgrades are done per-package. Systems on which functionality needs to be dynamically added (apps, etc.)
December 2001

Buildroot is created by *uClibc* developers as a way of building small embedded Linux systems to test *uClibc*
Starting around 2005

Buildroot really starts to be used as an embedded Linux build system for production devices.

The number of developers increases with everybody having write-access to the repository, and the maintainer is no longer active. No stable releases, no design.

The code slowly gets crappier over the years.
January 2009

Peter Korsgaard becomes the new maintainer. Start of a new period for the project:

- Stable releases every three months. First release 2009.02, 2012.08 due at the end of august.
- Huge cleanup effort: code base reduction from 5.2 MB to 2.2 MB, while many packages are added, updated and many new features are added.
- Increase in the number of contributors and users. Regular Buildroot Developer Days.
- Linux model: only the maintainer has write-access. Allows to keep a very good consistency in the design decisions.
Context: quick Buildroot history

Number of commits per month

Number of e-mails per day
Three years of change

After three years of major changes in *Buildroot*, it’s a good time to have a fresh look at it.
Buildroot: general principle

Open-source components (from http, ftp, git, svn, etc.)

In-house components (from http, ftp, git, svn, etc.)

Buildroot

Makefiles build procedures

Config.in configuration options

.config your configuration

root filesystem image

kernel image

bootloader image(s)

toolchain
Using Buildroot

$ wget http://buildroot.org/downloads/buildroot-2012.05.tar.bz2
$ tar xjf buildroot-2012.05.tar.bz2
$ cd buildroot-2012.05

$ make [menu|x|n|g]config

▶ No need to run as root. Ever.
▶ No need to symlink /bin/sh to bash
▶ Very limited set of dependencies: a native compiler, and basic utilities like awk, bison, patch, gzip, tar, wget, etc.
▶ Out-of-tree build is possible using O=, exactly like the Linux kernel.
Target architecture

Allows you to define:

- **The target architecture**, i.e ARM, x86, PowerPC, MIPS. Buildroot supports non-MMU architectures such as Blackfin or soon Microblaze, thanks to its uClibc support.

- **The target architecture variant**, such as ARM926 or Cortex-A8 on ARM. Allows to automatically add the appropriate \(-\text{mcpu}, -\text{march}, -\text{mtune}\) arguments to gcc.

- **Target architecture ABI**
Build options

- The **download directory**, where tarballs are saved for future builds. Defaults to $(TOPDIR)/dl. Can also be overridden using the BUILDROOT_DL_DIR environment variable.

- The **host directory**, where all host utilities are installed, including the *toolchain* and its *sysroot*. Defaults to $(O)/host, but can be changed to generate an SDK in a different directory.

- The **number of jobs**. Buildroot build the different components sequentially, but uses `make -j` to compile the individual components.

- Use of **ccache**

- Other build options: build with debugging symbols, install documentation on target, install development files on target, etc.
Toolchain

Buildroot provides three toolchain back-ends:

► An **internal Buildroot toolchain back-end**, which has been available since Buildroot’s creation. Buildroot will directly build a *uClibc* toolchain and use it for cross-compiling all packages.

► An **external toolchain back-end**, which allows use existing pre-built *uClibc*, *glibc* or *eglibc* toolchains, such as Sourcery CodeBench toolchains, Linaro toolchains, or toolchains that have previously been built using Crosstool-NG or Buildroot. Using an external toolchain removes the toolchain build time.

► A **Crosstool-NG back-end**, which tells Buildroot to build a cross-compiling toolchain with Crosstool-NG. This allows to benefit from all Crosstool-NG advantages, such as support for *glibc* or *eglibc*.

→ Buildroot often had the reputation of being limited to *uClibc*, but it is no longer the case, for several years now.
System configuration

Allows the definition of various system-wide parameters:

- The /dev management solution:
  - static, where device nodes are created statically at build time according to a device table
  - devtmpfs
  - devtmpfs + BusyBox’ mdev
  - devtmpfs + udev

- The serial port for the console

- Location of a post-build script that gets run after all packages have been built, but before the filesystem images are created.

- Some various other parameters (hostname, etc.)
Packages

Package Selection for the target

Arrow keys navigate the menu. <Enter> selects submenus -->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search.

Legend: [*] feature is selected [ ] feature is excluded

BusyBox

- customize
  - Audio and video libraries and applications -->
  - Compressors and decompressors -->
  - Debugging, profiling and benchmark -->
  - Development tools -->
  - Games -->
  - Graphic libraries and applications (graphic/text) -->
  - Hardware handling -->
  - Interpreter languages and scripting -->
  - Libraries -->
  - Miscellaneous -->
  - Networking applications -->
  - Package managers -->
  - Real-Time -->
  - Shell and utilities -->
  - System tools -->
  - Text editors and viewers -->
A selection of more than 750 open-source components typically used in embedded systems:

- Audio and multimedia: gstreamer, mplayer, pulseaudio, many codec libraries, alsa, etc.
- Graphics: full X.org stack, Gtk, Qt, EFL, DirectFB, SDL, etc.
- System tools, filesystem utilities, hardware utilities and libraries
- Networking applications: dropbear, avahi, bluez, samba, pppd, connman, etc.
- Development, debugging: oprofile, lttng, etc.
- Interpreters: Python, PHP, Ruby, Perl
Filesystem images

Buildroot can generate filesystem images in multiple formats:

- tar
- cpio
- ext2
- jffs2
- ubi/ubifs
- cramfs
- cloop
- iso9660
- squashfs
- initramfs built into the kernel
Kernel and bootloader

- Automates the **Linux kernel build process**. Builds the kernel using a *defconfig* or a specified configuration file. Takes care of installing the kernel modules into the root filesystem. It is also capable of patching the kernel for real-time extensions like Xenomai or RTAI.

- Includes support for the **most popular bootloaders**: U-Boot, Barebox, Grub, syslinux, but also architecture-specific bootloaders: AT91Bootstrap, X-Loader, lpc32xxcdl, etc.
Once the configuration is defined and saved in `.config`, the compilation is triggered using:

```bash
$ make
...
$ ls output/images/
dataflash_at91sam9m10g45ek.bin rootfs.tar
rootfs.ubi rootfs.ubifs
u-boot.bin u-boot-env.bin
uImage
```

Buildroot will automatically download, extract, patch, configure, compile and install the selected components, taking care of following the necessary dependencies.
In the output directory (by default, output/, but can be changed using out-of-tree build), Buildroot generates:

- **build**, with one sub-directory per component built. This is where Buildroot extracts and builds the various components of the system.

- **host**, with a typical Unix organization, in which Buildroot installs all utilities compiled for the host, including the cross-compiler. It also contains, in host/usr/<tuple>/sysroot the toolchain sysroot, with all the headers and libraries built for the target.

- **images**, with the final images (root filesystem, kernel, bootloaders)

- ...
Build output (2/2)

- ...;
- **stamps**, a few stamp files used internally by Buildroot to keep track of what has been built;
- **target**, the target root filesystem (but with an empty `/dev` and invalid permissions, those are fixed using `fakeroot` in the final image);
- **toolchain**, where the different toolchain components are built in the case of the internal back-end. Otherwise unused.
Build-time: basic examples

- Building a simple ARM system with simply BusyBox, using an already extracted CodeSourcery glibc toolchain, and generating a .tar filesystem image takes **39 seconds**. The target system size is **4.2 MB**.

- Building a simple ARM system with simply BusyBox, using a tarball CodeSourcery glibc toolchain, and generating an UBIFS filesystem image takes **2 minutes 57 seconds**. Identical system size.

- Building the same simple ARM system, using an internal Buildroot toolchain (with largefile, locales, C++ support), takes **9 minutes 59 seconds**. The target system size is **2.2 MB**.

Note: durations measured with the *real* output of the `time` command. Download times are excluded.
The device is an ARM AT91-based platform with GPS, RFID readers, GSM modem, Ethernet and USB.

The Buildroot configuration:
- CodeSourcery ARM glibc toolchain
- Linux kernel
- BusyBox for the basic system
- Dropbear for SSH access (debugging)
- Qt with only QtCore, QtNetwork and QtXml, no GUI
- QExtSerialPort
- zlib, libxml2, logrotate, pppd, strace, a special RFID library, popt library
- The Qt application
- JFFS2 root filesystem

Filesystem size: 11 MB. Could be reduced by using uClibc.

Build time: 10 minutes on a fast build server (quad-core i7, 12 GB of RAM)
Dependencies: real-world example 1
Build time: real-world example 2

- An x86-based system, with an OpenGL application for vehicle navigation system.
  - External glibc toolchain generated with crosstool-NG
  - The Grub bootloader
  - Linux kernel, of course
  - BusyBox
  - A large part of the X.org stack (the server, a few drivers, and some client libraries), including libdrm, Mesa
  - The fglrx ATI proprietary OpenGL driver
  - ALSA utils, ALSA library, V4L library, Flashrom, LM Sensors, Lua, Dropbear, Ethtool
  - The OpenGL application and its data

- Filesystem size: 95 MB, with 10 MB of application (binary + data) and 45 MB (!) of fglrx driver.
- Build time: 27 minutes on a fast build server (quad-core i7, 12 GB of RAM)
Dependencies: real-world example 2

Just a small part of it...
Overall logic: preparation

Buildroot starts by:

- Creating all output directories
- Copying the target root filesystem skeleton, located in `fs/skeleton` in the source tree, into the `target/output` directory. This `skeleton` contains the basic directories and configuration files for the target system. One can specify a different skeleton through a configuration option.
Buildroot continues by building the toolchain:

► For **internal** toolchains, it extracts and builds the different elements (binutils, gcc, uClibc, etc.) in the toolchain directory, with *stamp* files to keep track of what has already been done. It installs the results in the `host/usr/` directory (for binaries) and `host/usr/<tuple>/sysroot` directory (for libraries and headers).

► For **external** toolchains, Buildroot copies the original toolchain sysroot to the `host/usr/<tuple>/sysroot` directory and creates wrappers for the toolchain binaries in `host/usr/bin`.

► For **Crosstool-NG toolchains**, Buildroot installs *Crosstool-NG* in `host/`, and then uses it to generate a toolchain with binaries in `host/usr/bin` and sysroot in `host/usr/<tuple>/sysroot`. 
Buildroot continues by building the packages. Packages encapsulate the build procedure of userspace libraries and programs but also the Linux kernel or bootloaders build procedures.

The build method of each package is described using an infrastructure: AUTOTARGETS for autotools-based packages, CMAKETARGETS for CMake-based packages and GENTARGETS for other packages.

Buildroot follows the dependencies expressed in the package recipes, and triggers the configuration, compilation and installation steps as described in the recipes.
The package source code is extracted in build/<pkg>-<version>/, and stamp files are created after each step to let Buildroot know of what has already been done. Commands like make <foo>-reconfigure and make <foo>-rebuild allow to restart the build of a package if needed.

Host packages are installed in host/, while target packages are installed in target/ (usually stripped, no headers). Target libraries are also installed in host/usr/<tuple>/sysroot so that they are found by the cross-compiler (unstripped, with headers and static libraries).
Finally, Buildroot generates the *root filesystem image(s)*:

- It uses *fakeroot* to generate this root filesystem image without having to be root
- A *makedevs* utility is used to adjust the file permissions and ownership, and to create device files if a static /dev was chosen.
Source code organization

▶ **board**, board-specific patches and configuration files
▶ **boot**, bootloaders recipes
▶ **configs**, default configuration files for various platforms. Same concept as kernel *defconfigs*
▶ **docs**
▶ **fs**, recipes for generating root filesystem images in various formats, and also root filesystem skeleton in *fs/skeleton*
▶ **package**, all user-space packages (for the host and the target)
▶ **support**, misc scripts and tools
▶ **target**, legacy, almost empty directory
▶ **toolchain**, toolchain handling code (build recipes for internal and Crosstool-NG back-ends, integration recipes for external toolchains)
Adding packages: configuration

- Each package has its own directory in package/. Let’s say package/foo for our package.
- A package/foo/Config.in file needs to be created to declare at least one configuration option for the package. The syntax is identical to the \textit{kconfig} syntax:

```plaintext
config BR2_PACKAGE_FOO
  bool "foo"
  select BR2_PACKAGE_ZLIB
  help
  This is package foo

  http://foo-project.org
```

- This package.foo/Config.in file must be included from package/Config.in:

```plaintext
source "package/foo/Config.in"
```
The recipe must be written in the `package/foo/foo.mk` file.

It consists of:

- Variable declarations to define the package location, version, and the steps to be done to build the package.
- A call to one of the AUTOTARGETS, GENTARGETS or CMAKETARGETS macro to expand the package recipe.
Adding packages: minimal recipe

Basic recipe for autotools based package

FOO_VERSION = 1.3
FOO_SOURCE = foo-$(FOO_VERSION).tar.bz2
FOO_SITE = http://foo-project.org/downloads
FOO_DEPENDENCIES = zlib host-pkg-config

$(eval $(call AUTOTARGETS))

The AUTOTARGETS infrastructure:

- knows how to configure, build and install the package
- handles the common cross-compilation issues with autotools based packages (libtool problems, passing the right arguments and environment variables to ./configure, etc.).
Adding packages: various source methods

**From Git**

FOO_VERSION = this-branch
FOO_SITE = git://git.foo-project.org/foo.git

**From Subversion**

FOO_VERSION = 12345
FOO_SITE = http://foo-project.org/svn/foo/trunk/
FOO_SITE_METHOD = svn

**From a local directory**

FOO_SITE = /home/thomas/projects/foo/
FOO_SITE_METHOD = local

And also from Mercurial, from Bazaar, etc.
Packages: GENTARGETS

For packages that use a special build system (not autotools, not CMake).

FOO_VERSION = 1.3
FOO_SOURCE = foo-$(FOO_VERSION).tar.bz2
FOO_SITE = http://foo-project.org/downloads
FOO_DEPENDENCIES = zlib

define FOO_CONFIGURE_CMDS
    echo "HAS_ZLIB=YES" >> $(@D)/config
endef

define FOO_BUILD_CMDS
    $(MAKE) -C $(@D) \
    CC="$(TARGET_CC)" CFLAGS="$(TARGET_CFLAGS)" \
    all
endef

define FOO_INSTALL_TARGET_CMDS
    $(INSTALL) -D -m 0755 $(@D)/foo $(TARGET_DIR)/usr/bin
endef

$(eval $(call GENTARGETS))
Packages: other important mechanisms

- **Patches** in the package/foo directory are automatically applied if they are named `<number>-<something>.patch` or `<version>/<number>-<something>.patch`

- Need to set `FOO_INSTALL_STAGING = YES` for **packages that install libraries**, so that headers and static library are installed in the toolchain *sysroot*

- Can add *hooks* to **execute custom actions** before/after the different steps, especially for *AUTOTARGETS* packages

- Can set `FOO_AUTORECONF = YES` for autotools-based package to **autoreconfigure** them, useful when the `configure.ac` or `Makefile.am` files are patched.

- The **documentation** contains tutorials and a reference with all details about the package infrastructure `docs/manual/manual.pdf`
Buildroot simplicity

- After a configuration modification, Buildroot **does not even try to apply the configuration changes** during the next `make` invocation.
- Tracking the consequences of configuration modifications is very complicated (when a toolchain setting is changed, everything need to be rebuilt; when a library is removed, all reverse dependencies needs to be rebuilt)
- Buildroot remains simple: it **simply executes the build procedure of the selected packages, and does not try to track the files installed by each package.**
- Relies on the **user’s knowledge** about the configuration change to know what needs to be done. Since building is very fast, full rebuilds are not problematic.
- This focus on simplicity is a **very strong design decision** of Buildroot. Some features, such as binary packages generation, are not implemented in order to keep the build system simple.
Customizing for a project

Buildroot offers multiple mechanisms to handle all the particularities of a given project:

▶ A board/<company>/<project> directory to store all project-specific files: kernel and bootloaders patches, scripts and configuration files, etc.

▶ Storage of the project configuration as a minimal defconfig file in the configs/ directory.

▶ A hookable post-build script that gets called after all packages are installed but before the root filesystem image is created.

▶ The possibility of easily adding packages for custom software, including software coming from local repositories or directories.

See the slides of the Using Buildroot for real projects talk given at the Embedded Linux Conference Europe 2011.
Testing effort

- Buildroot provides more than 2000 config options, which creates a huge number of possible configurations.
- We are running 24/24 random configuration builds on 4 different machines, targeting 40+ combinations of architectures/toolchains configurations.
- Results visible live at http://autobuild.buildroot.org
- Daily summary sent to the Buildroot mailing list
## Buildroot build tests

<table>
<thead>
<tr>
<th>Date</th>
<th>Status</th>
<th>Commit ID</th>
<th>Submitter</th>
<th>Arch</th>
<th>Failure reason</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-07-07 14:51:13</td>
<td>NOK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>arm</td>
<td>tremor-18153</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 14:35:31</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>sh4</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 14:25:14</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Peter Korshgaard (gcc110)</td>
<td>i686</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 14:08:40</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Peter Korshgaard (gcc14)</td>
<td>powerpc</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 13:59:31</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>arm</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 13:24:02</td>
<td>NOK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>arm</td>
<td>libffi-3.0.11</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 13:20:53</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>arm</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 13:14:37</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Peter Korshgaard (gcc14)</td>
<td>powerpc</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 13:07:48</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Peter Korshgaard (gcc110)</td>
<td>i686</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 13:07:32</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Peter Korshgaard (gcc110)</td>
<td>arm</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 12:53:53</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>powerpc</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 12:42:59</td>
<td>OK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>mips</td>
<td>none</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
<tr>
<td>2012-07-07 12:26:57</td>
<td>NOK</td>
<td>37aa01f2</td>
<td>Thomas Petazzoni (Free Electrons build server)</td>
<td>x86_64</td>
<td>bind-9.6-ESV-R7-P1</td>
<td>dir, end log, full log, config, defconfig</td>
</tr>
</tbody>
</table>
Future directions

- Implement a mechanism for automatic generation of a licensing report, detailing all components used in the system and their license
- Add more packages for SoC-specific software (hardware codecs, 3D acceleration, special bootloaders)
- Improve the documentation with more tutorials
- Finalize the cleanup effort in the remaining areas
References

- Website: http://buildroot.org
- Mailing-list: buildroot@uclibc.org
  http://buildroot.org/lists.html
- IRC channel: #buildroot on Freenode
- Bug tracker: https://bugs.uclibc.org
Discover *Buildroot* by practice during this Libre Software Meeting!

**Half-day tutorial organized on Thursday 12th July, from 14:00 to 17:00**, in room 2150

Creating a first project configuration (BusyBox+kernel), creating additional packages, and more if time permits.

Tutorial takes place on **IGEPv2 boards**, based on OMAP3 ARM processors

Participants need to come with their own laptop, equipped with any reasonably recent GNU/Linux distribution.
Conclusion

- **Nice**: uses well-known technologies and languages, `kconfig` and `make`. Active community.
- **Simple**: only a few hundreds lines of code to understand the core infrastructure. Easy to get started, and easy to fully understand the internal mechanisms.
- **Efficient**: very reasonable build times, only what’s necessary gets built.

Of course, Buildroot is not suitable for *all* embedded Linux projects, but it is well adapted to industrial-like projects which have a limited number of software components and fixed functionality.
Questions?

Thomas Petazzoni

thomas.petazzoni@bootlin.com

Slides under CC-BY-SA 3.0. PDF and sources will be available on
https://bootlin.com/pub/conferences/2012/lsm/