Getting started with Buildroot

Thomas Petazzoni
thomas.petazzoni@bootlin.com

bootlin

Formerly Free Electrons

Thomas Petazzoni

- Embedded Linux engineer at ~~Free Electrons~~ → Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
  - Freely available training materials
- Open-source contributor
- Living in **Toulouse**, France

Building an embedded Linux system

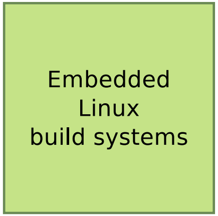
Pre-built binary Linux distributions

+ Readily available
- Large, usually 100+ MB
- Not available for all architectures
- Not easy to customize
- Generally require native compilation

Building an embedded Linux system

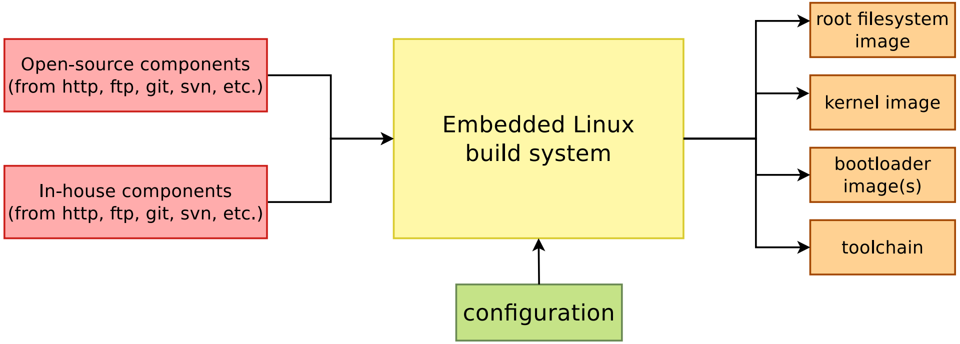
+ Smaller and flexible
- Very hard to handle cross-compilation and dependencies
- Not reproducible
- No benefit from other people's work

Building an embedded Linux system



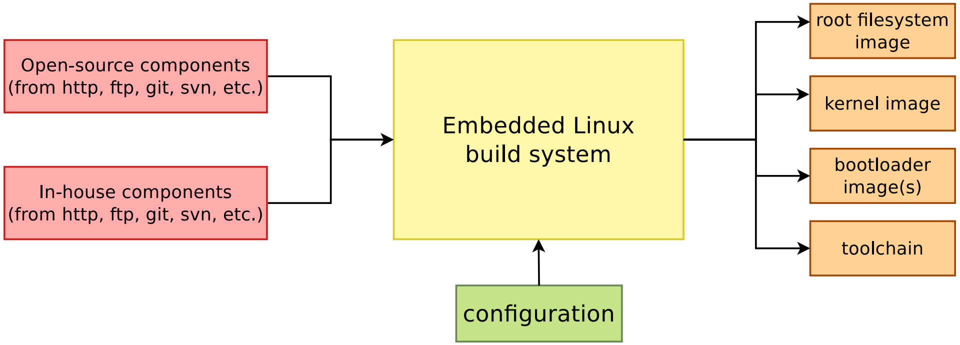
- + Small and flexible
- + Reproducible, handles cross-compilation and dependencies
- + Available for virtually all architectures
- - One tool to learn
- - Build time

Embedded Linux build system: principle



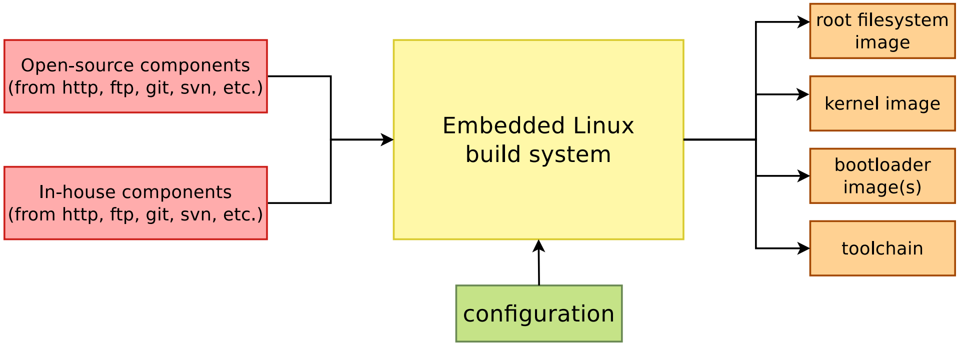
- Building from source $\rightarrow$ lot of flexibility

Embedded Linux build system: principle


Open-source components (from http, ftp, git, svn, etc.)
In-house components (from http, ftp, git, svn, etc.)
Embedded Linux build system
configuration
root filesystem image
kernel image
bootloader image(s)
toolchain

- Building from source → lot of flexibility
- Cross-compilation → leveraging fast build machines

Embedded Linux build system: principle


Open-source components (from http, ftp, git, svn, etc.)
In-house components (from http, ftp, git, svn, etc.)
Embedded Linux build system
configuration
root filesystem image
kernel image
bootloader image(s)
toolchain

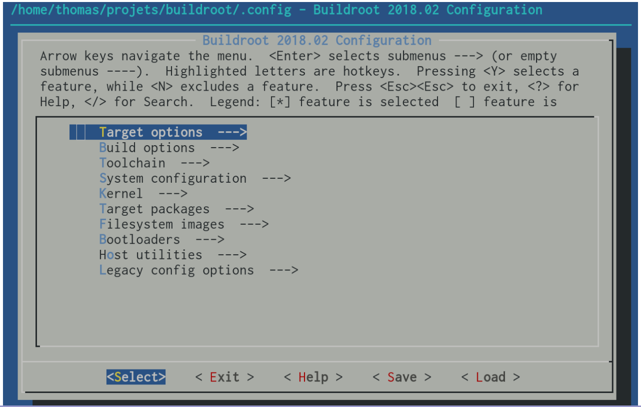
- Building from source → lot of flexibility
- Cross-compilation → leveraging fast build machines
- Recipes for building components → easy

Buildroot at a glance

- Is an **embedded Linux build system**, builds from source:
  - cross-compilation toolchain
  - root filesystem with many libraries/applications, cross-built
  - kernel and bootloader images
- **Fast**, simple root filesystem in minutes
- **Easy** to use and understand: kconfig and make
- **Small** root filesystem, default 2 MB
- More than **2300 packages** available
- Generates filesystem images, not a distribution
- Vendor neutral
- Active community, stable releases every 3 months
- Started in 2001, oldest still maintained build system
- `http://buildroot.org`

Getting started

```
$ git clone git://git.busybox.net/buildroot
$ cd buildroot
$ make menuconfig
```



```
/home/thomas/projets/buildroot/.config - Buildroot 2018.02 Configuration
                    Buildroot 2018.02 Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> selects a
  feature, while <N> excludes a feature.  Press <Esc><Esc> to exit, <?> for
  Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is

        Target options  --->
        Build options  --->
        Toolchain  --->
        System configuration  --->
        Kernel  --->
        Target packages  --->
        Filesystem images  --->
        Bootloaders  --->
        Host utilities  --->
        Legacy config options  --->

     <Select>    < Exit >    < Help >    < Save >    < Load >
```

Buildroot configuration

1. Target architecture

- Architecture
  ARC, ARM, AArch64, Blackfin, csky, m68k, Microblaze, MIPS(64), NIOS II, OpenRisc, PowerPC(64), SuperH, SPARC, x86, x86_64, Xtensa
- Specific processor
- ABI
- Floating point strategy

Buildroot configuration

1. Target architecture
2. Build options

- Download directory
- Number of parallel jobs
- Use of *ccache*
- Shared or static libraries
- etc.

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain

- Buildroot toolchain
  - Buildroot builds the toolchain
  - uClibc, glibc, musl
- External toolchain
  - Uses a pre-built toolchain
  - Profiles for existing popular toolchains
    Linaro, Sourcery CodeBench, etc.
  - Custom toolchains

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain
4. System configuration

- Init system to use: BusyBox, Sysvinit, Systemd
- `/dev` management solution: static, devtmpfs, mdev, udev
- Hostname, password, getty terminal, etc.
- Root filesystem overlay
- Custom post build and post image scripts
- etc.

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain
4. System configuration
5. Kernel

- Kernel source (stable version, Git tree, patches)
- Kernel configuration
- Support for kernel extensions: RTAI, Xenomai, aufs, etc.

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain
4. System configuration
5. Kernel
6. Target packages

- More than 2300 packages
- Qt4, Qt5, X.org, Gtk, EFL
- GStreamer, ffmpeg
- Python, Perl, Ruby, Lua, Erlang
- Samba, OpenSSL, OpenSSH, dropbear, lighttpd
- OpenGL support for various platforms
- And many, many more libraries and utilities

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain
4. System configuration
5. Kernel
6. Target packages
7. Filesystem images

- Major filesystem formats supported
- cloop
- cpio, for kernel initramfs
- cramfs
- ext2/3/4
- jffs2
- romfs
- squashfs
- tar
- ubifs

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain
4. System configuration
5. Kernel
6. Target packages
7. Filesystem images
8. Bootloaders

- Grub2
- Syslinux
- U-Boot
- Barebox
- and more platform-specific bootloaders: imx-bootlets, at91bootstrap, etc.

Buildroot configuration

1. Target architecture
2. Build options
3. Toolchain
4. System configuration
5. Kernel
6. Target packages
7. Filesystem images
8. Bootloaders
9. Host utilities

- Allows to build some native tools, useful for development.

Building and using

- To start the build: `make`
- Results in `output/images`:
  - `rootfs.ext4`, root filesystem in ext4 format
  - `zImage`, Linux kernel image
  - `am335x-pocketbeagle.dtb`, Linux kernel Device Tree blob
  - `u-boot.img`, U-Boot bootloader image
  - `MLO`, U-Boot bootloader image
- Ready to be flashed on your embedded system.

Exploring the build output

- All the output produced by Buildroot is stored in `output/`
- Can be customized using `O=` for out-of-tree build
- `output/` contains
  - `output/build`, with one sub-directory for the source code of each component
  - `output/host`, which contains all native utilities needed for the build, including the cross-compiler
  - `output/host/<tuple>/sysroot`, which contains all the headers and libraries built for the target
  - `output/target`, which contains *almost* the target root filesystem
  - `output/images`, the final images

Summarized build process

1. Check core dependencies
2. For each selected package, after taking care of its dependencies: download, extract, patch, configure, build, install
   - To `target/` for target apps and libs
   - To `host/<tuple>/sysroot` for target libs
   - To `host/` for native apps and libs
   - Filesystem skeleton and toolchain are handled as regular packages
3. Copy rootfs overlay
4. Call post build scripts
5. Generate the root filesystem image
6. Call post image scripts

Customizing the build

Besides the existing packages and options, there are multiple ways to customize the generated root filesystem:

- Create custom *post-build* and/or *post-image* scripts
- Use a *root filesystem overlay*, recommended to add all your config files
- Add your own packages

Adding a new package: Config.in

package/libmicrohttpd/Config.in

```
config BR2_PACKAGE_LIBMICROHTTPD
        bool "libmicrohttpd"
        depends on BR2_TOOLCHAIN_HAS_THREADS
        help
          GNU libmicrohttpd is a small C library that makes it easy to
          run an HTTP server as part of another application.

          http://www.gnu.org/software/libmicrohttpd/

comment "libmicrohttpd needs a toolchain w/ threads"
        depends on !BR2_TOOLCHAIN_HAS_THREADS
```

package/Config.in

```
[...]
source "package/libmicrohttpd/Config.in"
[...]
```

Adding a new package: <pkg>.mk, <pkg>.hash

package/libmicrohttpd/libmicrohttpd.mk

```
LIBMICROHTTPD_VERSION = 0.9.59
LIBMICROHTTPD_SITE = $(BR2_GNU_MIRROR)/libmicrohttpd
LIBMICROHTTPD_LICENSE = LGPL-2.1+
LIBMICROHTTPD_LICENSE_FILES = COPYING
LIBMICROHTTPD_INSTALL_STAGING = YES
LIBMICROHTTPD_CONF_OPT = --disable-curl --disable-examples

$(eval $(autotools-package))
```

package/libmicrohttpd/libmicrohttpd.hash

```
# Locally calculated
sha256 9b9ccd7d0b11b0e17...  libmicrohttpd-0.9.59.tar.gz
sha256 70e12e2a60151b9ed...  COPYING
```

Adding a new package: infrastructures

- In order to factorize similar behavior between packages using the same build mechanism, Buildroot has **package infrastructures**
  - `autotools-package` for autoconf/automake based packages
  - `cmake-package` for CMake based packages
  - `python-package` for Python Distutils and Setuptools based packages
  - `generic-package` for non-standard build systems
  - And more: `luarocks-package`, `perl-package`, `rebar-package`, `kconfig-package`, etc.

Defconfigs

- Pre-defined configurations for popular platforms
- They build a *minimal* system for the platform
- `make <foobar>_defconfig` to load one of them
- Some of the configs
    - Raspberry
    - BeagleBone
    - CubieBoard
    - PandaBoard
    - Many Atmel development boards
    - Several Freescale i.MX6 boards
    - Many QEMU configurations
    - and more...
- `make list-defconfigs` for the full list

Buildroot design principles

- **Cross-compilation only**: no support for doing development on the target.
- **No package management system**: Buildroot doesn't generate a distribution, but a firmware
- **Don't be smart**: if you do a change in the configuration and restarts the build, Buildroot doesn't try to be smart. Only a full rebuild will guarantee the correct result.

Documentation and support

- Extensive manual: `https://buildroot.org/downloads/manual/manual.html`
- 3-day training course, with freely available materials: `https://bootlin.com/training/buildroot/`
- Mailing list: `http://lists.busybox.net/pipermail/buildroot/`
- IRC channel: `buildroot` on Freenode

Today's lab

- **Step 1**: do a minimal build for the PocketBeagle, with just a bootloader, Linux kernel and minimal root filesystem. Generate a ready-to-use SD card image.
- **Step 2**: enable network over USB and SSH connectivity using Dropbear. Shows how to use a rootfs overlay and how to add packages.
- **Step 3**: customize the Linux kernel configuration, compile a small application that uses the GPIO, first manually, and then using a new Buildroot package
- Follow the instructions at `https://github.com/e-ale/Slides/blob/master/buildroot/buildroot-lab.pdf`
- **Don't hesitate to request help and ask questions!**