Buildroot vs. OpenEmbedded/Yocto Project: A Four Hands Discussion

Alexandre Belloni, Thomas Petazzoni
Free Electrons
alexandre.belloni@free-electrons.com
thomas.petazzoni@free-electrons.com

Alexandre Belloni

- The **Yocto Project guy** for this talk
- Embedded Linux engineer/trainer at Free Electrons since 2013
- Yocto Project/OE expert
- Maintainer of several Freescale boards in `meta-fsl`, strong contributor to `meta-atmel`
- OpenEmbedded setup for numerous customers
- Trainer for Free Electrons Yocto Project / OpenEmbedded course
- And also a kernel contributor: co-maintainer of the RTC subsystem and the Atmel ARM processor support

Thomas Petazzoni

- The **Buildroot guy** for this talk
- CTO and Embedded Linux engineer/trainer at Free Electrons since 2008
- Strong Buildroot contributor
- Interim Buildroot maintainer
- Trainer for Free Electrons Buildroot course
- And also a kernel contributor: support for Marvell processors

Common aspects

- Embedded Linux build systems
  - Goal is to build a complete, customized, embedded Linux system
  - Root filesystem, toolchain, kernel, bootloaders
- Build **from scratch from source**
- Using **cross-compilation**
- Very **actively maintained** and developed projects
- **Widely used** in the industry
- Well documented, training courses
- Free software

Buildroot: general philosophy

- Strong focus on **simplicity**
- Simple to use, but also simple to understand/extend
- Special use cases handled via **extension scripts**, rather than in Buildroot itself
- Re-use of **existing technologies**/languages: kconfig, make.
  - Learning effort worth the investment
- **Minimalist**: small by default
- Purpose-agnostic
- **Open community**, no vendor or bureaucratic/corporate management

Yocto Project: general philosophy

- Support for the major architectures
  - OpenEmbedded: only qemu
  - Yocto Project: adds support for a few machines
- Only provides **core** recipes, use **layers** to get support for more packages and more machines
- Custom modifications should stay in a separate layer
- **Versatile** build system: tries to be as flexible as possible and to handle most use cases.
- Open community but the project is still governed by the Yocto Project Advisory Board made up of its corporate sponsors.
- OpenEmbedded is an independent community driven project.

Buildroot: output

- Buildroot main product is a **root filesystem image**
  - But also: toolchain, kernel image, bootloaders, etc.
- Many formats supported: ext2/3/4, ubifs, iso9660, etc.
- **No binary packages**, no package management system
  - Some people call it a *firmware generator*
  - Updates are not possible via packages
  - Updates require a full system update, like Android
  - Belief that partial updates are harmful

Yocto Project: output

- Builds **distributions**, the main output is a package feed
  - the package management system is optional on the target
  - it is possible to install or update only part of the system
- Also generates **root filesystem images** by installing those packages. Supports the usual ext2/3/4, ubifs, iso9660, etc.. but also VM images: vmdk, vdi, qcow2
- Finally, images classes or a tool, `wic`, can be used to create **disk images**
- Also able to generate an **SDK** alongside the image to allow application developers to compile and test their applications without having to integrate it in the build. But, the SDK has to be kept in sync with the image.

Buildroot: configuration

- Re-uses *kconfig* from the Linux kernel
- Simple {*menu,x,n,g*}*config* interface
- Entire configuration stored in a single `.config`/`defconfig`
- Defines all aspects of the system: architecture, kernel version/config, bootloaders, user-space packages, etc.
- `make menuconfig`, `make`, profit!
- Building the same system for different machines: to be handled separately
  - A tool can generate `defconfig` from fragments
  - Doable, but not super easy
  - Complete separate build for each machine

Yocto Project: configuration

- The configuration is separated in multiple parts:
  - **Distribution** configuration (general package configuration, toolchain and libc selection,...)
  - **Machine** configuration (defines the architecture, machine features, BSP.)
  - **Image** recipe (what packages should be installed on the target.)
  - Local configuration (distribution and default machine selection, how many threads to use when compiling, whether to remove build artifacts,...)
- It is also necessary to gather the various **layers** that will be used and declare them.
- Allows to build the same image for different machines or using different distributions or different images for one machine.

Buildroot: layers

- Initially: no concept of layers
- All packages are maintained in the official repository
  - Allows for very high quality, thanks to review by experts
- Addition of `BR2_EXTERNAL`
  - Allows to store package definitions, configurations and other artefacts
  - One `BR2_EXTERNAL` only
  - Generally used for proprietary/custom packages and configurations
  - Can only add packages, not override the ones in Buildroot

Yocto Project: layers

- Layer mechanism allows to **modify or add** new package or image recipes.
- Clear separation between the core build system, the BSP and custom modifications.
- Scales properly, third parties provide their layers with BSPs or a set of recipes handling their dedicated applications
- Layers have to be compatible and use the same OE branch base.
- Beware of layer quality, reviews are not systematic.
- OpenEmbedded Metadata Index lists available layers, recipes, machines: `http://layers.openembedded.org/layerindex/`
- Also, there is a powerful **overrides** mechanism allowing to adjust recipe variables based on the machine or distribution.

Buildroot/Yocto Project: toolchain

Similar capabilities

- Building their own toolchains, based on gcc, a choice of C libraries (glibc, uClibc, musl)
- Using pre-built external toolchains
  - Somewhat easier in Buildroot, since it's built-in
  - Only really properly supported with additional vendor layers in the Yocto Project

Buildroot: new package

package/tmux/Config.in

```
config BR2_PACKAGE_TMUX
        bool "tmux"
        depends on BR2_USE_MMU # fork()
        select BR2_PACKAGE_LIBEVENT
        select BR2_PACKAGE_NCURSES
        help
          tmux is a terminal multiplexer, it enables a number of terminals
          (or windows) to be accessed and controlled from a single terminal.

          https://tmux.github.io/
```

package/tmux/tmux.mk

```
TMUX_VERSION = 2.1
TMUX_SITE = https://github.com/tmux/tmux/releases/download/$(TMUX_VERSION)
TMUX_LICENSE = ISC
TMUX_LICENSE_FILES = README
TMUX_DEPENDENCIES = libevent ncurses host-pkgconf

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

package/tmux/tmux.hash

```
# Locally computed:
sha256  31564e7bf4bcef2de...f6176  tmux-2.1.tar.gz
```

Yocto Project: new package

meta-oe/recipes-extended/tmux/tmux_2.1.bb

```
SUMMARY = "Terminal multiplexer"
HOMEPAGE = "http://tmux.sourceforge.net"
SECTION = "console/utils"

LICENSE = "ISC"
LIC_FILES_CHKSUM = "file://tmux.c;beginline=3;endline=17;md5=8685b4455...

DEPENDS = "ncurses libevent"

SRC_URI = "git://github.com/tmux/tmux.git;branch=master"
SRCREV ?= "310f0a960ca64fa3809545badc629c0c166c6cd2"

S = "${WORKDIR}/git"
B = "${WORKDIR}/build"

inherit autotools pkgconfig

PACKAGECONFIG ??= ""
PACKAGECONFIG[utempter] = \
    "ac_cv_header_utempter_h=yes,ac_cv_header_utempter_h=no,libutempter,"
```

Buildroot: complexity

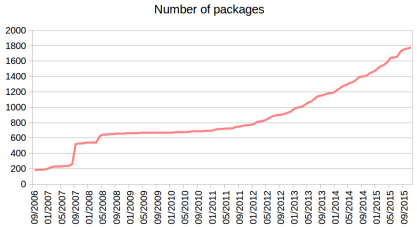
- Designed to be **simple**
- Every proposed feature for the core is analyzed in terms of **usefulness/complexity** ratio
- Core logic written entirely in make, less than 1000 lines of code including 230 lines of comments
  - Really easy to understand what is going on, why and how a package is built
  - Almost as simple as a shell script downloading, extracting, building and installing your software components one after the other
- Detailed documentation, lots of resources/talks
- One hour talk sufficient to describe all the internals (ELCE 2014)
- Typical feedback on IRC: *coming over from Yocto and very pleasantly surprised how much easier it is. This is the first thing that's stumped me*

Yocto Project: complexity

- Somewhat steeper learning curve.
- The core is `bitbake`, a separate project written in python (60kloc).
- A set of **classes** define the common tasks.
- Recipes are written in a mix of bitbake specific language, python and shell script.
- Logging and debugging allows to understand what is done for each task.
- Detailed documentation but many different configuration variables.
- It is not always easy to understand the best practices (e.g. Poky should not be used for production, distro/image modifications should not be done in local.conf, removing tmp/)
- People are still confused by the terminology (Yocto Project, Poky, OpenEmbedded, bitbake)

Buildroot: number/variety of packages

- 1800+ packages
- Graphics: X.org, Wayland, Qt4/Qt5, Gtk2/Gtk3, EFL
- Multimedia: Gstreamer 0.10/1.x, ffmpeg, Kodi, OpenGL support for numerous platforms
- Languages: Python 2/3, PHP, Lua, Perl, Erlang, Mono, Ruby, Node.js
- Networking: Apache, Samba, Dovecot, Exim, CUPS, lots of servers/tools
- Init systems: Busybox (default), initsysv, systemd
- No support for a toolchain on the target



Yocto Project: number/variety of packages

- Several thousand recipes: around 2200 for oe-core, meta-openembedded, meta-qt5. More than 8400 known by the Metadata Index (includes duplicates).
- Mostly the same packages as Buildroot
- Has layers for more languages: Java, Go, Rust, smalltalk
- Still has an active layer for Qt3
- meta-virtualization (Docker, KVM, LXC, Xen) and meta-openstack layers

Buildroot: approach of dependencies

- Minimalistic approach
  - if a feature can be disabled, it should be disabled by default
- Lots of automatic dependencies
  - i.e., if you enable OpenSSL, you will automatically get SSL support in all other packages you have enabled that can provide SSL support
- Leads to small root filesystems by default, with no effort

Yocto Project: approach of dependencies

- Package configuration is done at the distribution level
  - Enabling OpenSSL will enable it for all the packages
  - but it is still possible to disable it for a few packages
  - inversely, it is possible to enable a feature only for selected packages.
- Can be amended at the machine level but this should be avoided.
- Each recipe can define its own set of default features to lead to a sane default configuration.

Buildroot: updates/security

- Releases published every three months: two months of development, one month stabilization.
- Every release includes updates of package versions: both security updates and major updates.
- And also potentially changes in the core infrastructure.
- No LTS versions with only security updates so far, users are handling this on their own, only for the packages they care about
- Script to assess the unfixed CVEs in a given Buildroot configuration is being contributed.

Yocto Project: updates/security

- Releases published every six months: one release in April, one in October.
- Planning and roadmap is available on the wiki:
  `https://wiki.yoctoproject.org/wiki/Yocto_Project_v2.1_Status`
- Four milestones with 3 months between M1 and the final release
- At least the previous and the current releases have appointed maintainers, they get **security** and important fixes but no recipe upgrade.
- Older releases are maintained by the community, on a best effort basis.

Buildroot: detection of configuration changes

- Buildroot does not try to be smart
- When doing a change in the configuration, it does not try to detect what needs to be rebuilt
- Once a package has been built, Buildroot will not rebuild it unless you force it
- Major configuration changes require a full rebuild
  - Starts to be annoying for large systems
- Minor configuration changes can often be handled without a full rebuild
  - Requires understanding a bit what you're doing
- No way to share build artefacts between different configurations: one configuration, one build.

Yocto Project: detection of configuration changes

- `bitbake` maintains a shared State Cache (*sstate-cache*) to allow incremental builds.
- It detects changes in the inputs of task by creating a checksum of those inputs
- This cache is shared between all the builds. Building for similar machines is fast.
- It is possible to share that cache across hosts (for example a nightly build server and a developer machine), allowing to greatly speed up a full build.

Buildroot: architecture support

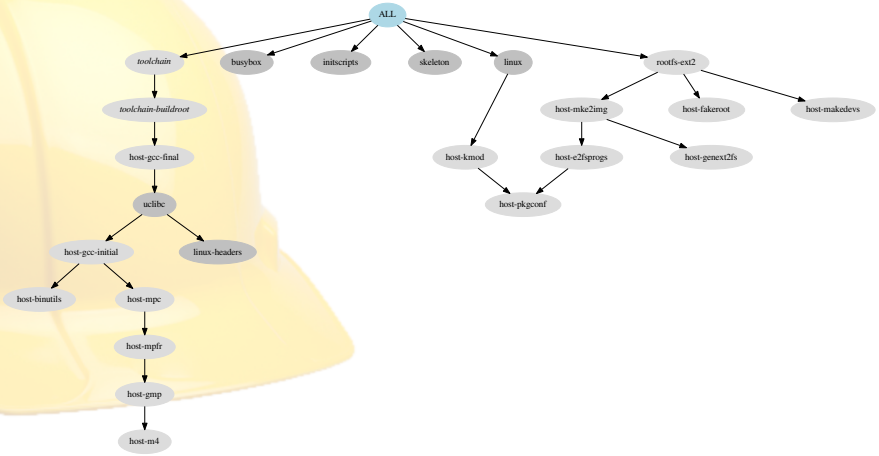
- Wide range of architecture support
- **Big ones**: ARM(64), MIPS, PowerPC(64), x86/x86-64
- But also numerous **more specific** architectures
  - Xtensa, Blackfin, ARC, m68k, SPARC, Microblaze, NIOSII
  - ARM noMMU, especially ARMv7-M
- Contributions from architecture vendors
  - MIPS by Imagination Technologies
  - PowerPC64 by IBM
  - ARC by Synopsys
  - Blackfin by Analog Devices

Yocto Project: architecture support

- In the core: ARM, MIPS, PowerPC, x86 and their 64-bit variants
- In separate layers:
  - Microblaze, NIOSII
- Usually **silicon vendors maintain** their own BSP layers:
  - meta-intel
  - meta-altera (both ARM and NIOSII)
  - meta-atmel, meta-fsl, meta-ti, ...
  - meta-xilinx (both ARM and Microblaze)
- But also community effort:
  - meta-rockchip, meta-sunxi, ...

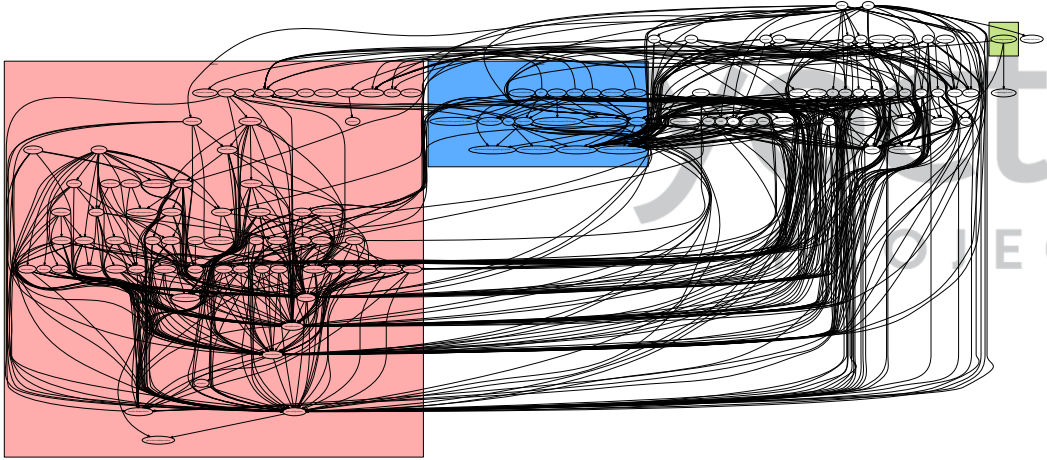
Buildroot: minimal build

Minimal build (`qemu_arm_versatile_defconfig`) takes 15 minutes and 25 seconds, image size is 2.2MB.

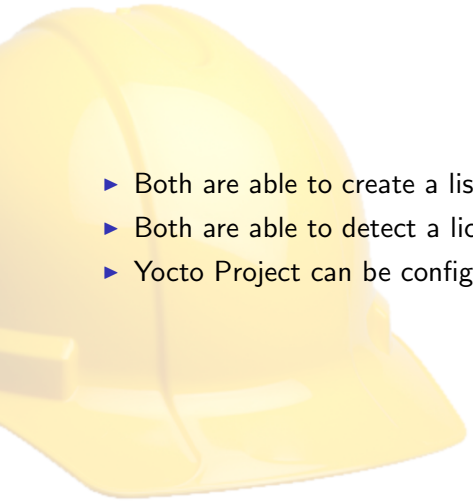


Yocto Project: minimal build

Minimal build (`core-image-minimal` for `qemuarm`) takes 50 minutes and 47 seconds, image size is 4.9MB. With an existing sstate-cache, it takes 1 minute 21 seconds.

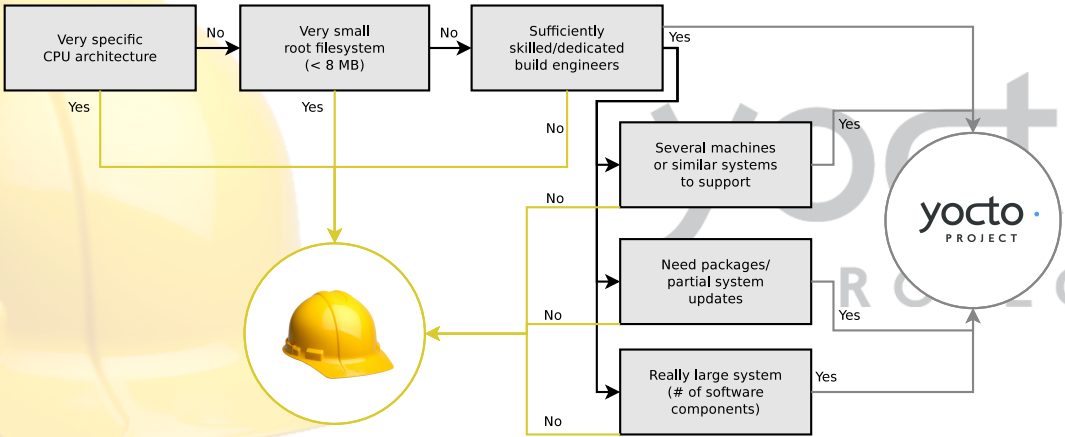


License compliance



- Both are able to create a list of used licenses
- Both are able to detect a license change
- Yocto Project can be configured to exclude GPLv3

Buildroot/Yocto Project: choosing



Questions?

Alexandre Belloni
Thomas Petazzoni

alexandre.belloni@free-electrons.com
thomas.petazzoni@free-electrons.com

http://free-electrons.com/pub/conferences/2016/elc/belloni-petazzoni-buildroot-oe/