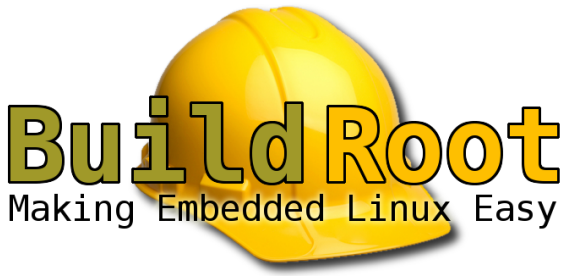
Rootfs made easy with Buildroot

How kernel developers can finally solve the rootfs problem.



Thomas Petazzoni
Free Electrons
thomas.petazzoni@free-electrons.com

Thomas Petazzoni

- CTO and embedded Linux engineer at Free Electrons
  - 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.
  - **We're hiring!**
  - `http://free-electrons.com`
- Contributing the **kernel support for the new Armada 370 and Armada XP** ARM SoCs from Marvell (widely used in NAS devices).
- Major contributor to **Buildroot**, an open-source, simple and fast embedded Linux build system
- Living in **Toulouse**, south west of France

Doing kernel development is awesome, but...

A kernel without a root filesystem is kind of useless

```
input: ImExPS/2 Generic Explorer Mouse as /devices/fpga:07/serio1/input/input1
Root-NFS: no NFS server address
VFS: Unable to mount root fs via NFS, trying floppy.
VFS: Cannot open root device "(null)" or unknown-block(2,0)
Please append a correct "root=" boot option; here are the available partitions:
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(2,0)
```

Solutions often used by kernel dev

- A **complete Linux distribution**
  - \+ Readily available
  - \- Large (can hardly be used as an initramfs)
  - \- Not available for all architectures
  - \- Not easy to customize.
- A **pre-built rootfs**
  - \+ Usually relatively small
  - \- Even less flexible
  - \- Not available for all architectures.
- A rootfs **built manually** with Busybox
  - \+ Smaller and flexible
  - \- Quite some work to create
  - \- Busybox is often not sufficient (testing sound? video? input devices? graphics? network?).
  - \- Difficult to reproduce for colleagues

→ There must be something better.

Embedded Linux build systems

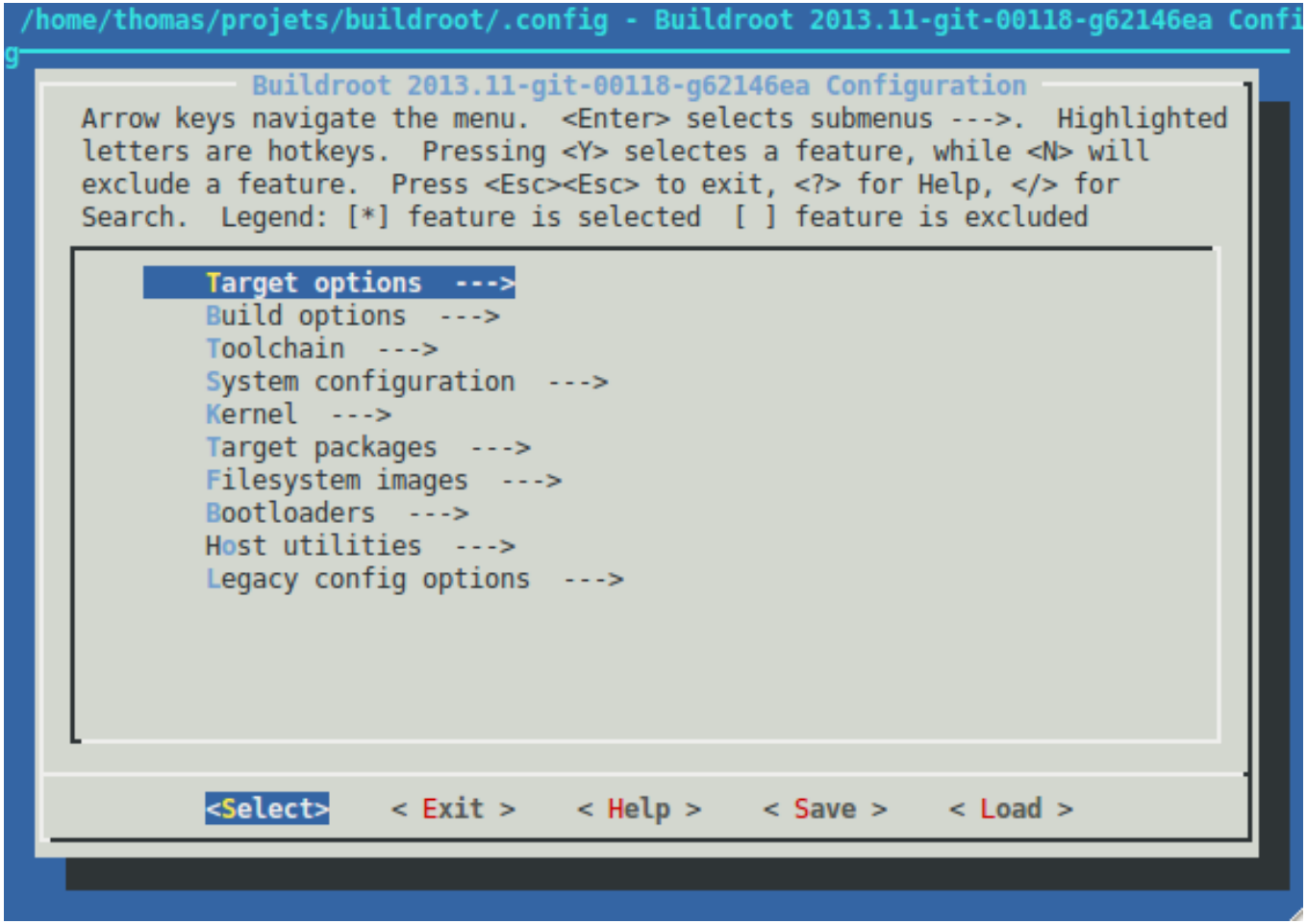
- **Embedded Linux build systems** are tools that automate the process of building a small (or not so small) Linux system **from source**.
- From source → lot of flexibility
- Generally used to build production embedded Linux systems, but they can also be used by kernel developers to build small root filesystems for testing purposes!
- Well-known build systems: OpenEmbedded, Yocto, **Buildroot**, PTXdist, etc.
- *For me, OE fails the easy to understand, quick to setup test. I now use Buildroot.* – Kevin Hilman, ARM kernel developer

Buildroot

- Can build a toolchain, a rootfs, a kernel, a bootloader
- **Easy to configure**: menuconfig, xconfig, etc.
- **Fast**: builds a simple root filesystem in a few minutes
- Easy to understand: written in make, extensive documentation
- **Small** root filesystem
- More than **1000 userspace libraries/apps** available, including important development and debug tools (see later)
- **Many architectures** supported: x86, ARM, PowerPC, MIPS, SuperH, Blackfin, ARC, Xtensa, Microblaze, Nios II.
- Active community, regular releases
- Used by many embedded system makers, including *Google* for the *Google Fiber* boxes.
- `http://buildroot.org`

Basic usage (1)

make menuconfig



Basic usage (2)

`make`

It builds...

Root filesystem image available in `output/images`.

initramfs use case: why

- *initramfs* is great for kernel development because your rootfs is fully in RAM, and has absolutely **no dependency on any kernel driver**. You don't need a storage driver or a network driver.
- However, since the root filesystem is loaded entirely in RAM, at every kernel boot, it **has to be small**. Buildroot's ability to generate really small filesystems is very useful here.

initramfs use case: Buildroot side

Example for an ARM Cortex-A8 platform.

- `make menuconfig`
  - **Target options**, select the *ARM* architecture, the *EABIhf* ABI and the *Cortex-A8* architecture variant.
  - **Toolchain**, select *External toolchain*, and then the *Linaro* toolchain.
  - **System configuration**, select the *devtmpfs* `/dev` management method and ensure the serial port for the *getty* is correct.
  - **Filesystem images**, select the *cpio* format.
- `make`
- Your CPIO image is in `output/images/rootfs.cpio`.
  - Contains just Busybox.
  - 3 MB uncompressed.
  - 2 minutes and 46 seconds of complete build time, on a 2 years old laptop.

initramfs use case: kernel side

The kernel configuration should have:

- `CONFIG_INITRAMFS_SOURCE="/path/to/buildroot/output/images/rootfs.cpio"`
- `CONFIG_INITRAMFS_COMPRESSION_GZIP=y` or some other compression algorithm
- `CONFIG_DEVTMPFS=y`, to get *devtmpfs* support, to provide a dynamic */dev*
- Note that Buildroot does this configuration automatically when it is responsible for building the kernel. However, when doing active kernel development, one typically builds the kernel outside of Buildroot.

NFS use case: how

- In Buildroot, select the *tar* filesystem image format.
- After the build, uncompress as root the image tarball to your NFS exported directory:
  `sudo tar -C /nfsroot -xf output/images/rootfs.tar`
  Note: this can be automated with a *post-image script*.
- Configure your kernel to mount `/nfsroot`, and make sure you have `CONFIG_DEVTMPFS`, `CONFIG_DEVTMPFS_MOUNT`, `CONFIG_NFS_FS` and `CONFIG_ROOT_NFS` enabled.
- Do **not** directly use `output/target` as the NFS-exported directory. Permissions and ownership are not correct.

Extending your rootfs

Doing kernel development that needs some specific userspace tools? Buildroot already has a good number of them:

- A must is Dropbear for SSH.
- Benchmarks/testing programs: dhrystone, iozone, bonnie++, LTP, netperf, ramspeed, stress, lmbench, iostat, memtester, etc.
- Debugging tools: latencytop, LTTng, trace-cmd, etc.
- Hardware interaction: evtest, i2c-tools, devmem2, pciutils, usbutils, libv4l, yavta, alsa-utils, linux-firmware, mii-diag, etc.
- Many more packages already available in Buildroot.
- And can be extended with more packages: easy to do, and well documented in the Buildroot manual.

Configuration example 1

Used for my kernel development on Marvell Armada 370/XP.

```
BR2_arm=y
BR2_cortex_a8=y
BR2_TOOLCHAIN_EXTERNAL=y
BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_MDEV=y
BR2_PACKAGE_LINUX_FIRMWARE=y
BR2_PACKAGE_LINUX_FIRMWARE_IWLWIFI_5000=y
BR2_PACKAGE_LINUX_FIRMWARE_MWIFIEX_SD8787=y
BR2_PACKAGE_LINUX_FIRMWARE_RTL_8192=y
BR2_PACKAGE_LINUX_FIRMWARE_RTL_8712=y
BR2_PACKAGE_I2C_TOOLS=y
BR2_PACKAGE_PCIUTILS=y
BR2_PACKAGE_USBUTILS=y
BR2_PACKAGE_USBUTILS_ZLIB=y
BR2_PACKAGE_DROPBEAR=y
BR2_PACKAGE_ETHTOOL=y
BR2_PACKAGE_IPERF=y
BR2_PACKAGE_IW=y
BR2_PACKAGE_MII_DIAG=y
BR2_PACKAGE_WIRELESS_TOOLS=y
BR2_TARGET_ROOTFS_CPIO=y
```

- For ARM Cortex-A8
- Uses *mdev* for firmware loading
- Has a few firmwares
- i2c-tools
- pciutils
- usbutils
- dropbear
- and various network/wireless related tools.
- Size: 5.6 MB, 13 minutes build time.

Configuration example 2

- Used for my kernel development on Marvell Armada 370/XP in Big Endian mode.
- A more minimal variant, with just Dropbear, but shows that doing rootfs for 'exotic' architectures is also possible.

```
BR2_armeb=y
BR2_cortex_a8=y
BR2_TOOLCHAIN_EXTERNAL=y
BR2_TOOLCHAIN_EXTERNAL_PATH="/home/thomas/x-tools/armebv7-marvell-linux-gnueabi-softfp_i686/"
BR2_TOOLCHAIN_EXTERNAL_CUSTOM_PREFIX="armeb-marvell-linux-gnueabi"
BR2_TOOLCHAIN_EXTERNAL_CUSTOM_GLIBC=y
BR2_TOOLCHAIN_EXTERNAL_CXX=y
BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_DEVTMPFS=y
BR2_PACKAGE_DROPBEAR=y
BR2_TARGET_ROOTFS_CPIO=y
```

Conclusion

Questions?

http://buildroot.org

Thomas Petazzoni
thomas.petazzoni@free-electrons.com
Thanks to the Buildroot community (Arnout Vandecappelle, Thomas De Schampheleire, Peter Korsgaard) and to Kevin Hilman for reviewing the slides.

http://free-electrons.com/pub/conferences/2013/kernel-recipes/rootfs-kernel-developer/