Discovering Buildroot

Minimal Linux systems for small footprint embedded systems

Who I am

Michele Comignano

- Software Engineer doing Cloud Services and Maps processing for a living
- Not a professional embedded engineer
- Author of **Tonio** — free jukebox for children

Why embedded Linux

Building and running a focused and small footprint Linux system on hardware with limited resources.

Multiple options exist

Approach	Pros	Cons
Regular Distro (if the board allows)	Ready to use, good for quick prototyping	Slow boot, bloated for special purpose usage beyond prototyping, needs extra resources
Yocto	Maximum flexibility	Overly complex for tinkering and DIY projects
Linux system from Scratch	Good learning exercise	Impractical if you aim at higher level functionality
Buildroot	Quick start with low footprint	Not a distribution if that's what you're looking for

Buildroot for embedded Linux

As a very reasonable tradeoff.

Buildroot

The "compiler" of embedded Linux systems.

Buildroot is not a distribution. It is a **build system** (basically KConfig plus a set of Makefiles) that gets you through:

1. **Customizing the content of the system** in terms of packages, configs and Linux config
2. **Building a cross-compilation toolchain** targeting any supported hardware
3. **Building the Linux kernel** configured as you need
4. **Building a minimal rootfs** with only the software you need
5. **Assembling a bootable image** ready to flash

Buildroot by the numbers

- **More than 2900 packages** supported
- **Dozens of boards** with reasonable defaults to start with (Raspberry Pis, OrangePi and many more)
- Typical image can be **few to tens of MB** *vs* a handful of GB of a general purpose distro
- Initial build: **10–50 minutes** (depends on build host hardware and configuration)
- **LTS** release roughly every 2 years with 3-year support

If a board is not supported, that means you don’t have defaults. But with some work it will work for you too!

Last but not least, Buildroot is backed by an extremely competent and friendly community that welcomes contributions.

Anatomy of Buildroot

Quick start

1. Choose an existing defconfig for your board (or start from scratch)
2. customize with `make menuconfig` → choose toolchain, packages and much more
3. build with `make` (it does everything including rootfs images)
4. flash image to your device
5. boot!

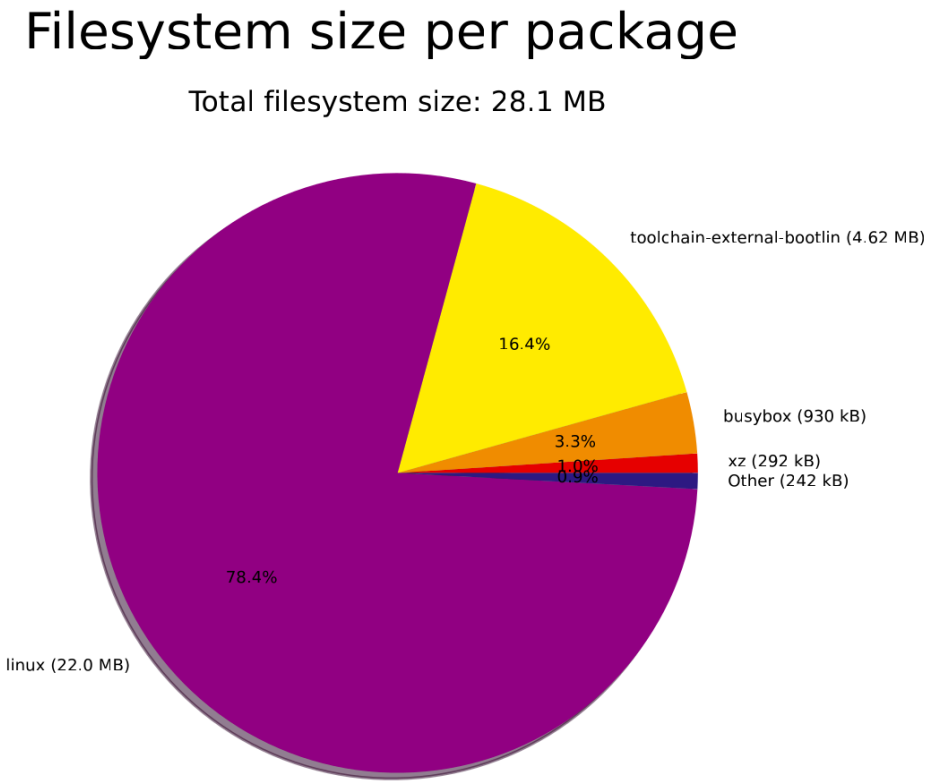
Demo: Buildroot for a Raspberry PI 3 in no time

Plus time to build.

```
# 1. Download Buildroot
$ wget https://www.buildroot.org/downloads/buildroot-2026.05.tar.gz
$ tar xzf buildroot-2026.05.tar.gz
$ cd buildroot-2026.05

# 2. Load default config for Raspberry Pi 3
$ make raspberrypi3_defconfig

# 3. Build!
$ make
```



Demo: flashing to the board

A bootable SD card image with:

- **Linux kernel** (fairly big one with all default drivers, more on that later)
- **BusyBox** (typical low footprint init+shell for embedded systems)
- **Rootfs** (**~28 MB** including a fat kernel)

```
# 4. Write to SD card
$ dd if=output/images/sdcard.img \
  of=/dev/sdX bs=4M \
  status=progress && sync
```

Deep dive: Buildroot directory tree

- `arch/` : supported architectures
- `board/` : board-specific files
- `boot/` : bootloaders
- `configs/` : preset defconfigs
- `linux/` : kernel recipes
- `package/` : all packages
- `system/` : rootfs skeleton
- `toolchain/` : internal/external toolchain
- `output/` : build results *(after build)*
  - `build/` : compiled sources
  - `host/` : toolchain + host tools
  - `staging/` : sysroot
  - `images/` : final images
- `Makefile` : the entry point

Deep dive: Kconfig (aka `$make menuconfig`)

Category	Description
Target options	architecture, FPU, ABI
Build options	ccache, output dir…
Toolchain	internal/external, C++, …
System config	hostname, init, console
Kernel	version, configuration
Target packages	all packages!
Filesystem	ext4, squashfs, initramfs

Buildroot external

How to build on top of Buildroot for real projects

The problem with working inside buildroot tree

```
$ git clone https://gitlab.com/buildroot.org/buildroot.git
# put my stuff inside
# sync with upstream forces to resolve conflicts continuously
# messy to share
```

Solution: **BR2_EXTERNAL** — or — keep the project as a **separate** extension of Buildroot

Buildroot External

The external tree

- `Config.in` : custom package menu
- `external.desc` : descriptor
- `external.mk` : Makefile for inclusions
- `configs/` :
  - `my_board_defconfig` : my configuration
- `board/` :
  - `my_board/` : overlay, scripts, kernel config
- `package/` :
  - `my-app/` : my custom package
    - `Config.in`
    - `my-app.mk`
    - `S99my-app` : init script
- `patches/` : specific patches

How to use it

```
# Go to a full buildroot tree (git or release)
$ make BR2_EXTERNAL=/path/to/my-project  my_custom_defconfig
$ make
```

Tonio: a case study

Let’s have a quick demo first

The project

An RFID-based **jukebox** for children:

- Place an **RFID token** → a playlist starts
- Buttons for **next/prev/volume**
- **Web interface** for configuration
- **Wi-Fi AP** (no network configuration needed)
- **Boots in ~5 seconds**

The Tonio philosophy

Inspired by **tonies®** (commercial toy) but:

- **Free** — GPLv3
- **No limits** — no DRM, no constraints
- **DIY** — start from scratch or from pre-built images
- **Affordable** — Raspberry Pi 3 + RFID reader + buttons + wiring + time

Tonio as BR2_EXTERNAL

- `external.desc` : name + description
- `external.mk` : include *.mk + flash helpers
- `Config.in`
- `configs/` : reference defconfig
- `board/raspberrypi/` :
  - `linux_3.config` : kernel config
  - `config_3.txt` : RPi firmware config
  - `genimage.cfg.in` : SD card layout
  - `flash.sh / flash-single.sh`
  - `post-build.sh / post-image.sh`
- `board/raspberrypi/` :
  - `rootfs_overlay/` : files to copy into `/`
    - `etc/default/tonio` : env vars
    - `etc/init.d/S41apsetup`
    - `etc/init.d/S90hostapd`
    - `hostapd.conf` , `dnsmasq.conf` ,...
- `package/tonio/` :
  - `Config.in` : menu entry
  - `tonio.mk` : cmake-package recipe
  - `S15tonio` : init script
  - `tonio-1.0/` : sources (inside)

Build and deploy

```
# Full build (with Buildroot)
$ cd buildroot-2026.05
$ make BR2_EXTERNAL=/path/to/buildroot-tonio tonio_raspberrypi3_defconfig
$ make

# Flash to SD card (from project directory)
$ make flash dev=/dev/sdc
$ make flash-boot dev=/dev/sdc   # boot only
$ make flash-rootfs dev=/dev/sdc # rootfs only

# Quick deploy (during development)
$ make tonio-deploy target=172.16.0.1
```

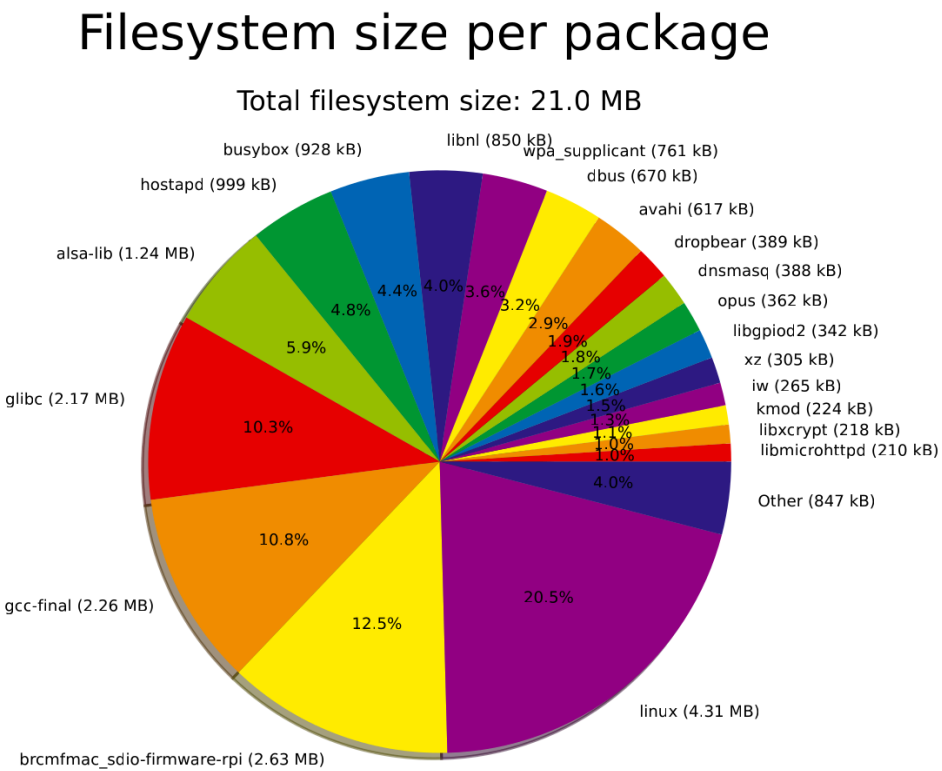


Image size

Component	Size
Linux kernel (zImage)	~4.3 MB
Total rootfs	**21 MB**

Measured boot time: **~5 seconds** from power-on to audio playback

What we learned

- Buildroot is **simple** — Kconfig + make
- **BR2_EXTERNAL** helps keep a focused project tree that extends the buildroot tree

Tonio demonstrates that:

- A **complete** uncompressed embedded Linux system can fit **~20 MB**
- You can make a **finished product** with Buildroot
- The barrier to entry is **very low**

Resources

- **buildroot.org** — Download, manual, mailing list
- **buildroot-2026.05** — Latest stable (non LTS)
- **Tonio** → github.com/comick/tonio (a ⭐ is much appreciated!)

The End

Thanks for your attention

Questions?