# Deciding between Buildroot & Yocto

By **Nathan Willis**
April 6, 2016

ELC

Often, two or more free-software projects that have similar goals prefer to ignore the fact that they compete for users and adoption. The developers and advocates choose to let the rivalry remain the unaddressed elephant in the room whenever they cross paths in public. But that was not the case in at least one session at the 2016 Embedded Linux Conference in San Diego, when a developer from Yocto and one from Buildroot shared a podium for a discussion of where the two projects overlapped and where they differed. Both are used to build an embedded Linux system from scratch (or, at least, from the source code up) and both are popular, but that is about all that they have in common.

Before diving into the details, the two speakers introduced themselves. Alexandre Belloni is a Yocto developer and trainer, maintaining the Yocto layers for several boards and contributing elsewhere (including to the kernel, where he is co-maintainer of the real-time clock subsystem). Thomas Petazzoni is an embedded Linux engineer who works on the Buildroot project. Both are employed by Free Electrons, which consults on Yocto and Buildroot projects for various customers.

**The great divide**

At the basic level, Petazzoni began, Yocto and Buildroot can both give you the same end product: a root filesystem image for your embedded device, a kernel, a bootloader, and a compatible toolchain. But, while many people have subjective opinions on which projects has the better approach, there are more than enough objective differences in how the two projects function to discuss.

Buildroot, he said, focuses on simplicity. The core Buildroot tool is kept as small and as simple as possible, which makes it easy to use and understand. All special cases are handled in extensions, and Buildroot re-uses existing tools like kconfig wherever possible. Thus, he said, learning Buildroot means learning tools that are applicable in other scenarios. The OS image Buildroot creates is also minimal by default, making builds fast, and delivering a "purpose agnostic" system, not one tailored to any particular use case.

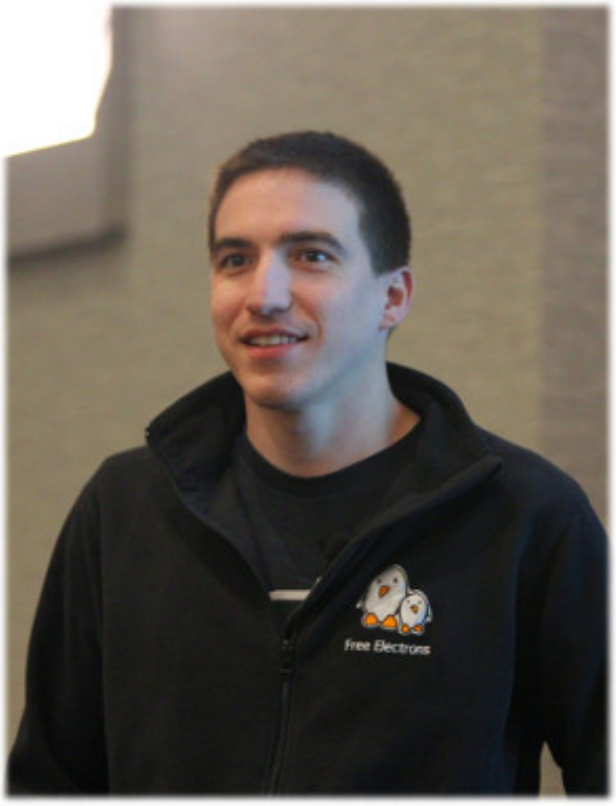
In contrast, Belloni continued, Yocto tries to be versatile and support a wide range of embedded systems. It does that by defining builds in recipes, which specify what software to build and how to build it, and by supporting layers, which are recipe collections written and maintained by the development community. Layers allow the community to support new boards or architectures, define new application stacks, or support entirely new use cases. By relying on cleanly defined layers, the default system image defined can remain small

defined can remain small.

The projects provide a tangible difference in what they generate, Petazzoni said. Buildroot's output is a root filesystem image, nothing more; that has even led some to call it a "firmware generator." Whenever you need to update the system, you must regenerate the entire image, or somehow build your own update mechanism (such as a binary diff system). This restriction is by choice, he said; the project believes embedded systems should not be updated like desktops and servers are, since the outcome of a failed or partial update can be disastrous.

Yocto's output is "a distribution," Belloni said. Specifically, it is a package feed, although providing a full-scale package-management system for the target device is optional and generating full disk images is certainly supported. But individual packages can be rebuilt as needed, individual packages can be updated or removed, and updating one package (say, to apply an OpenSSL security fix) is faster than a full rebuild. Yocto can also build a software development kit (SDK) to let application developers build and test their software for the device without rebuilding the OS image.

Builds made with Buildroot and Yocto are configured in differing manners as well, which Petazzoni called one of the clearest distinctions between the two. Buildroot puts all configuration information in one file, which can be edited using any of the interfaces from the kernel's kconfig tool (e.g., xconfig or menuconfig). That file specifies the architecture, kernel, bootloader, and user-space packages to include. He likes to describe the build process as "`make menuconfig; make; profit`." The drawback is that similar machines cannot share any configuration information: one must build every target separately.




Yocto, Belloni said, keeps configuration information in multiple parts. The distribution configuration dictates general packages, toolchains, and libraries; the machine configuration describes the architecture and hardware features; the image recipe lists which packages should be pre-installed; the local configuration controls how the build is performed. This approach means one may have to gather a lot of layers together before beginning a build, but it also makes it easy to build variations with little additional work. Building for variants of the same board can require only a small tweak.

Yocto, as described earlier, defines a build with a collection of layers. Buildroot has historically eschewed them, Petazzoni said, largely because the out-of-tree configurations the project sees have tended to be of mediocre quality only suitable for single use cases—and, thus, not suitable for sharing. But, recently, a number of companies have started pushing for a Buildroot feature to let them keep certain components outside of their main configuration, and the project has added a feature to support that behavior. It is called the `BR2_EXTERNAL` option, and it allows a build to incorporate "secret sauce" for package definitions, configurations, or other artifacts that will not be shared. `BR2_EXTERNAL` is limited, he said: a configuration can only use the feature one time, and it can only be used to add (not override) settings.

Yocto layers, Belloni said, can add or modify packages or even recipes. Layers enable a strict separation between the core of the build, the board support package (BSP), and custom modifications; that allows a configuration to scale properly. A third party can modify an existing layer by adding a layer of its own

that enables support for new hardware, without rewriting the original layer. That said, he noted, there

are many layers published in the OpenEmbedded Metadata Index (much of Yocto being based on work that originated in OpenEmbedded), but it is up to the user to look at them and notice if they are unmaintained or contradict another layer.

The complexity of the two systems is quite different. Buildroot strives to remain simple; Petazzoni said every change proposed for the core is analyzed in terms of its "usefulness to complexity ratio." The core logic is implemented entirely in Makefiles totaling under 1000 lines of code. That amount of code can be read and understood by an individual, he said. Yocto, Belloni conceded, has a bit more challenging of a learning curve. The main utility, BitBake, is a 60,000-line Python program, and recipes are written in a combination of Python, shell scripts, and a BitBake-specific language. Still, BitBake does provide logging and debugging aides, which are useful. The project's complexity is a known issue, he said: "best practices" are not always easy to find or understand, and many people seem to find the terminology (such as the distinctions between Yocto, OpenEmbedded, Poky, and BitBake) hard to grasp.

**Lesser distinctions**

The differences discussed so far encompass the major, large-scale distinctions between Yocto and Buildroot, but there are still quite a few important details that differ as well, which Petazzoni and Belloni turned to next.

First, Buildroot currently supports a smaller set of available packages—a little over 1800. The list includes all of the core options one would expect, from graphics layers to application toolkits to multimedia frameworks and programming languages. But it is a curated list of packages and, notably, it does not include any development tools (e.g., compilers and linkers) to run on the target system. Buildroot targets are meant to be the end product, not the development system. Yocto, however, does provide target toolchains—and much more. About 8400 packages are available in total, including some of questionable value for current systems, like Qt 3. But there are packages for more programming languages, including some like Go and Rust that are not available for Buildroot, and there are layers available for many containerization and virtualization frameworks (use cases that are outside of Buildroot's scope).

Dependency handling differs as well. Buildroot again takes a minimalist approach: whenever a new package is added, it is built with as many features disabled as possible. That allows the creation of small footprints: the minimal image is just 2MB. But Buildroot does include a number of automatic dependencies, Petazzoni said. For instance, adding OpenSSL to the configuration will automatically add SSL support for other packages that include an SSL option. In Yocto, package configuration is done at the distribution layer: enabling SSL support enables it for every package by default, although it can still be disabled for individual packages (or enabled for individual packages, if it is set to be off by default). These dependencies can be overridden at the machine-configuration level, but that is considered bad practice.

The two projects differ somewhat in their update schedules and security practices. Buildroot releases on a three-month cycle; each release includes package updates, new releases, and security updates. There are no "long-term support" releases with guarantees of security updates; users must handle that on their own. Yocto releases every six months, with four milestones in between each stable release. The current release and the previous release both have officially appointed maintainers and receive security and major updates. In addition, various community members make updates for older releases.

One final distinction to be aware of is that Buildroot provides no mechanism to detect configuration changes between builds. All rebuilds are full-image rebuilds, Petazzoni noted, so, by design, all rebuilds must be manually initiated. In contrast, BitBake maintains a shared cache between builds, so Yocto can skip rebuilding individual packages if they have not changed. Consequently, Belloni said, building slight variants for several different machines can be quite fast.

slight variants for several different machines can be quite fast.

There are number of factors where Buildroot and Yocto offer more-or-less comparable features. The toolchains they support are essentially the same, utilizing GCC and the user's choice of C libraries (glibc, uClibc, and musl). They support the same processor architectures, they both provide license-compliance tools, and they produce minimal builds of fairly similar specifications. Buildroot's default build is 2.2MB and can be completed in 15 minutes, the speakers said, while Yocto's default is 4.9MB and can be completed in 50 minutes from scratch or in just over one minute by using the package-build cache.

Ultimately, they said, the decision of which system to use is not a simple (much less single-issue) choice. There are a number of factors one should consider when choosing. The interesting criteria, as they saw matters, include whether or not the architecture needed is something strange or needs a very small root filesystem (factors that Buildroot is better suited for), whether or not you must support several machine configurations or need to provide package-management and updates (scenarios that Yocto is better for). The size of the development project also influences the choice: Buildroot is good for small teams with limited resources, while Yocto is good for large systems composed of many components. In the end, one size does not fit all where embedded Linux build systems are concerned.

*[The author would like to thank the Linux Foundation for travel assistance to attend ELC 2016.]*

---



**Deciding between Buildroot & Yocto**
Posted Apr 9, 2016 7:53 UTC (Sat) by **HIGHGuY** (subscriber, #62277) [Link]

As a user of an OE-derived and yocto-based platform I must say that it has matured a lot over the last few years.
The first versions I had to work with left a very bad taste in my mouth with regards to maintainability (why does this happen? where does that variable suddenly come from? Why can't I enable this for everything at once? ...)

However, as attested in the article, some debugging features were added to relieve those issues. One can now track where each part of a variable comes from, there's pkgconfig support and more.

Still, one must question if the addition of these debugging features aren't just a plaster on an open leg wound. One must truly understand how bitbake parses and operates, one must become very familiar with the layers used before you can effectively and efficiently make an image out of it. There's a very steep learning curve to go through to make a proper image. Apparently that's one of the prices to pay for the extreme flexibility it offers.

Another downside is that there's a ton of legacy. It's true that there are many layers and many packages in them. But you can scrap a lot of them as unusable as-is because they contain legacy cruft due to a lack of love and care.

Finally, one must see that, as far as I know, yocto is the go-to platform in the industry. All big embedded-distro vendors are using it. This is probably a result of the design-choices stated in the article.
My advice would be this: if this is for a hobby project and buildroot can fill your needs, use it. If it's for business and you have time/money to invest or if you require a lot of flexibility, go yocto.

**Deciding between Buildroot & Yocto**
Posted Apr 11, 2016 11:37 UTC (Mon) by **BlueLightning** (subscriber, #38978) [Link]

Posted Apr 11, 2016 11:37 UTC (Mon) by **BlueLightning** (subscriber, #38978) [Link]

> Still, one must question if the addition of these debugging features aren't just a plaster on an open leg wound. One must truly
> understand how bitbake parses and operates, one must become very familiar with the layers used before you can effectively and
> efficiently make an image out of it. There's a very steep learning curve to go through to make a proper image. Apparently that's
> one of the prices to pay for the extreme flexibility it offers.

I think we do try to improve things where we can, and we've come a long way since the Yocto Project started - of course there's only so much you can do to simplify a complex system, and there's no denying that OE is complex. I know you've been a long-term user based on your comments here so if you have any suggestions on how we might improve we would definitely like to hear them.

> Another downside is that there's a ton of legacy. It's true that there are many layers and many packages in them. But you can
> scrap a lot of them as unusable as-is because they contain legacy cruft due to a lack of love and care.

"A lot of them"? Quality does vary across layers, that is true - it depends on how diligent the maintainer is. I know this is probably too small a space to get into specifics but can you elaborate on where you've found lack of maintenance?

**Deciding between Buildroot & Yocto**
Posted Apr 9, 2016 18:09 UTC (Sat) by **ghane** (subscriber, #1805) [Link]

Perhaps it was the format of the talk, or the Author's style of writing, but this is one of the most lucid articles I have read on a topic I know nothing about, on LWN in a decade.

Thank you, sirs!

**Deciding between Buildroot & Yocto**
Posted Apr 11, 2016 11:29 UTC (Mon) by **BlueLightning** (subscriber, #38978) [Link]

I regret not being able to make it to ELC this year, sounds like this was one of the good talks. As someone who works on OpenEmbedded for the Yocto Project If there's anyone out there I'd trust to make an objective comparison of Buildroot and OE it's Free Electrons - and that's even though they themselves maintain Buildroot. Once again they prove that they're totally awesome :)

**Deciding between Buildroot & Yocto**
Posted Apr 12, 2016 9:10 UTC (Tue) by **tpetazzoni** (subscriber, #53127) [Link]

Thanks for the nice comment! To be completely correct, Free Electrons does not maintain Buildroot. The Buildroot maintainer is Peter Korsgaard, who is not and has never been a Free Electrons person, he has been working for Barco since many years. I do however act as an interim maintainer for Buildroot, helping the main maintainer to merge the significant amount of contributions we receive. But I'm by far not the official maintainer, and therefore do not take the big design / architecture decisions, though I can certainly influence them. In any case, stating that Free Electrons maintains Buildroot is incorrect.

**Deciding between Buildroot & Yocto**
Posted Apr 12, 2016 23:42 UTC (Tue) by **BlueLightning** (subscriber, #38978) [Link]

> OK - thanks for correcting that misconception on my part; I guess I'd just seen your name associated with Buildroot and made incorrect assumptions.

**Deciding between Buildroot & Yocto**
Posted Apr 14, 2016 2:52 UTC (Thu) by **xxiao** (guest, #9631) [Link]

Used both in projects and ended up going with Openwrt for final products. In short yocto/OE is the gentoo for embedded, and it's heavy and complicated, while buildroot is way more user friendly, I mean _way_ more.

Yocto is good for companies like Wind River or chip vendors like Freescale/NXP, if you're doing your own board bring up, it's much easier to go with Buildroot instead, or Openwrt if you want to do a network oriented product fast.

Buildroot also has the best documentation among all.

**Link to the presentation slides**
Posted Apr 22, 2016 8:33 UTC (Fri) by **michaelo** (subscriber, #23907) [Link]

http://free-electrons.com/pub/conferences/2016/elc/bellon...