Yocto Project and Android

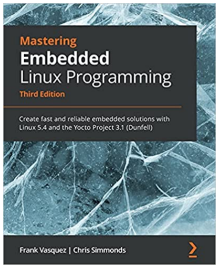
compare and contrast

Chris Simmonds

Embedded Linux Conference 2021

License

About Chris Simmonds



- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at `https://2net.co.uk/`

@2net_software

`https://uk.linkedin.com/in/chrisdsimmonds/`

Agenda

- Yocto and Android?
- Overview of Yocto Project
- Overview of AOSP
- Development process
- Community
- Conclusion

Previously

In the past I have spoken about Yocto Project and Debian[1]

I have also blogged about using Android as an embedded operating system [2]

This time I want to talk about Yocto Project and Android. Are they comparable? When would you use one or the other?

[1] https://elinux.org/images/8/89/Debian-and-yocto-csimmonds-elce-2020.pdf

[2] https://www.linkedin.com/pulse/7-reasons-consider-embedded-android-your-next-project-chris-simmonds/

Compare

Both Yocto Project and the Android Open Source Project (AOSP) are:

- open source
- flexible build systems
- designed to produce a full operating system

Both can be (and are) used to build fully functional embedded devices

Contrast

Yocto Project and AOSP are very different beasts

- Yocto Project is a community driven project, designed to build flexible Linux operating systems
- AOSP is driven by Google specifically for the market segments that they care about (mostly smart phones)

The core differences are ownership, control and community

But first, let's look at the technical differences...

- Yocto and Android?
- Overview of Yocto Project
- Overview of AOSP
- Development process
- Community
- Conclusion

Yocto Project

https://www.yoctoproject.org/

- Yocto Project is a build system that creates packages from source code
  - based on Bitbake and OpenEmbedded meta data
  - Yocto Project and OpenEmbedded have been used to create the software running on many millions of devices
- Allows you to create your own tailor-made distro
- You only need to build and deploy the packages you need

Getting Yocto Project

```
$ git clone git://git.yoctoproject.org/poky
```

The download is about 250 MiB, of which

- 55 MiB is tools, documentation and meta data
- 195 MiB is git history

It does not contain the upstream code that will compile and construct the images for your chosen platform

Setting up the environment

Begin by sourcing this script

```
$ cd poky
$ source oe-init-build-env [build dir]
```

- Creates a working directory for your project, default `build/`
- Changes into that directory

Local configuration

- Local configuration is in `[build dir]/conf/local.conf`
- Can contain many configuration variables, including

Variable	Description	Example
`MACHINE`	Target board	`MACHINE = "beaglebone"`
`DISTRO`	Distribution	`DISTRO = "poky"`
`PACKAGE_CLASSES`	Package format	`PACKAGE_CLASSES = "package_rpm"`
`EXTRA_IMAGE_FEATURES`	Additional features	`EXTRA_IMAGE_FEATURES = "debug-tweaks"`

Recipes

- The core meta data consists of recipes grouped together into **meta layers**
- The recipes are processed by a task scheduler named **BitBake**
- Recipes generate binary packages
- Packages combine to make images which can be copied to a device

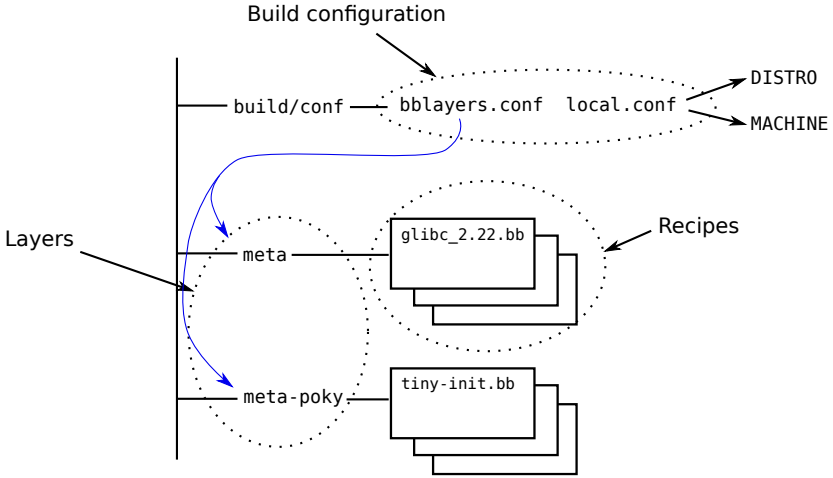
Recipes

- Here is a simple recipe that builds a "helloworld" program

poky/recipes-skeleton/hello-single/hello_1.0.bb

```
DESCRIPTION = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
    ${CC} ${LDFLAGS} helloworld.c -o helloworld
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 helloworld ${D}${bindir}
}
```

Config, layer and recipe

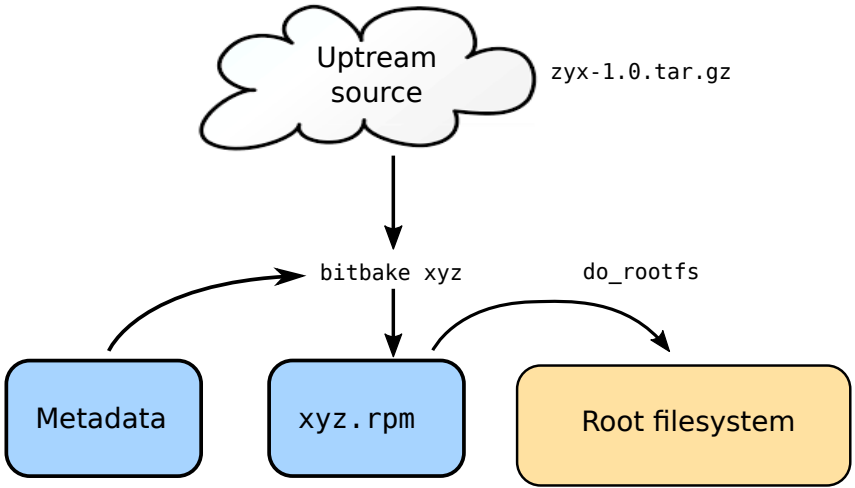


Building an image

- An **image recipe** generates the binaries that you load onto the target
- Typically includes bootloader, kernel, devicetree and root filesystem
- Example image recipes:
  - `core-image-minimal`: small, console-based
  - `core-image-full-cmdline`: a fully-functional console-only image
  - `core-image-x11`: basic graphics with an X11 server

```
$ bitbake core-image-minimal
```

Building a rootfs with Yocto Project



Meta layers

BitBake recipes are organised into meta layers

- Makes it easy to extend by adding new layers

Many layers are listed at
http://layers.openembedded.org/layerindex/branch/master/layers

- Machine (BSP), e.g. meta-raspberrypi
- Distribution, e.g. meta-agl
- Software, e.g. meta-qt5
- Miscellaneous

Many SoC and dev board vendors publish meta layers for their chips and boards

What can you do with Yocto?

Applications of Yocto Project range far and wide, for example:

- ticket machines for trams and trains
- water purity testing machines
- smart entry phones
- Automotive IVI systems, using the Automotive Grade Linux layer, meta-agl
- 4G and 5G base stations

... and many more

- Yocto and Android?
- Overview of Yocto Project
- Overview of AOSP
- Development process
- Community
- Conclusion

Overview of AOSP

- AOSP is the open source component of Android, consisting of
  - build system
  - base operating system (native layer)
  - Android framework and run-time (ART)
  - some demo apps, including Launcher3, Settings App, Desk Clock
- All Android devices are based on AOSP
- Mostly permissive licenses[1]: BSD and Apache 2.0

[1] Some debug and build tools have GPL license, but they are not present in user builds

Android without Google

- Since AOSP is open source you can modify and deploy it as you wish
- For example, the Amazon Fire branded devices are based on AOSP
- Or, you can build it into an embedded system ("Embedded Android")
- Typical use cases include
  - Test and measurement
  - Digital advertising
  - Marine navigation (!)

The AOSP build system: repo and manifests

- AOSP is composed of 100's of git repositories
- Managed via a **manifest** xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <remote  name="aosp"
           fetch=".." />
  <default revision="refs/tags/android-11.0.0_r42"
           remote="aosp"
           sync-j="4" />
[...]
  <project path="art" name="platform/art" groups="pdk" />
  <project path="bionic" name="platform/bionic" groups="pdk" />
  <project path="bootable/recovery" name="platform/bootable/recovery" gro
[...]
```

The AOSP build system: repo and manifests

- **repo** is a tool that parses a manifest and clones the repositories

Initialise: download the manifest and install tools

```
$ repo init -u https://android.googlesource.com/platform/manifest -b android-11.0.0_r42
```

Synchronise: clone the git repos

```
$ repo sync
```

Note: Yocto Project downloads from upstream at as needed during the build; repo downloads everything at sync time

`repo sync` will download >100 GiB

The AOSP build system: toolchains

AOSP comes with toolchains for the supported compiled languages:

- C/C++: `prebuilts/clang`
- Rust (since R/11): `prebuilts/rust`

... and the supported architectures:

- ARM: v7a (32-bit) and v8a (64-bit)
- x86 and x86_64

The toolchains are supplied as pre-built binaries

- saves time building them from source (as Yocto does)
- but, each binary toolchain is many MiB
- plus there are several versions for backwards compatibility
- so the whole of `prebuilts/` is currently 42 GiB

The AOSP build system: soong

- The build tool is named **soong**[1]
- soong parses recipes written in **Blueprint**
- Each bp file produces one or more **modules** named by the `name:` property

This is a simple Android.bp file:

```
// Build Low Memory Killer Daemon, lmkd
cc_binary {
    name: "lmkd",
    srcs: ["lmkd.c"],
    shared_libs: [
        "liblog",
        "libcutils",
    ],
    cflags: ["-Werror"],
}
```

[1] Since 7.0; previously it was GNU make

The AOSP build system: external modules

- Directory `external/` contains about 300 upstream projects
- Note: these are local copies (forks) of upstream, each with their own Android.bp
- Necessary because soong does not support autotools, cmake, make, etc

```
$ ls -1 external/
[...]
bc
bcc
blktrace
boringssl
bouncycastle
brotli
bsdiff
bzip2
caliper
capstone
catch2
cblas
cbor-java
chromium-libpac
chromium-trace
chromium-webview
clang
[...]
```

The AOSP build system: board support packages

- The BSPs are in directories `devices/` and `vendor/`
- Consist of
- `AndroidProducts.mk`
  - adds the device to the lunch menu
  - selects a device makefile
- `BoardConfig.mk`
  - type of CPU
  - flash memory layout and image sizes
  - plus other board-related stuff

Note: still using GNU makefile fragments. No Blueprint here

The AOSP build system: product type

- `AndroidProducts.mk` points to the device makefile
- Here you select the type of Android device

```
$(call inherit-product, $(SRC_TARGET_DIR)/product/aosp_base_telephony.mk)
```

- Options include
- `aosp_base_telephony.mk`: phone
- `aosp_base.mk`: tablet
- `atv_base.mk`: TV
- `car.mk`: Automotive OS

The AOSP build system: lunch

You select the device to build using the **lunch** command

The menu is populated by parsing all the `AndroidProducts.mk` files in `device/` and `vendor/`

```
$ source build/envsetup.sh
$ lunch

You're building on Linux

Lunch menu... pick a combo:
     1. aosp_arm-eng
[...]
     35. beagle_x15-userdebug
     36. beagle_x15_auto-userdebug
     37. car_x86_64-userdebug
     38. db845c-userdebug

Which would you like? [aosp_arm-eng] 38
```

kernel

- AOSP does not include a Linux kernel
- In most cases the kernel comes from the SoC vendor (Qualcomm, NXP, etc)
  - based on the Android Common Kernel
  - with many (1000's) of in-house patches
  - usually lags mainline Linux by 2 or 3 years
- But, since R/11 we are transitioning to the GKI (Generic Kernel Image)

GKI

- GKI is a single kernel code base which all Android devices should use
- Intended to reduce the duplicated effort and lag that results from today's vendor kernels
- Kernel and core modules are controlled by Google
- Vendors responsible for vendor modules (mostly binary-only)

Can you make a 100% open source Android product?

- Not really because you always need **binary blobs**:
  - vendor OpenGL ES libraries
  - vendor HALs
  - binary kernel modules (e.g. for WiFi, Bluetooth)
  - firmware binaries
- Binary blobs are supplied from SoC vendors, often under NDA
- Tie you in to a particular version of software
- Binary blobs are a pain, a drag on the rate of innovation

Why Android is not a Linux distro

AOSP is not a distro because you can't take an application, library, or script from another distro (e.g. Debian) and run it on Android.

Not only is it not binary compatible (which is understandable), it is not source code compatible either

Why Android is not a Linux distro

Incompatible and incomplete set of shared libraries

- libc is a stripped down C library called called bionic which omits many POSIX functions
- there is a very limited set of other system libraries

Non standard library loader, ld-android.so

- the VNDK (O/8) introduced complex rules for loading libraries
- enforces library load paths based on namespaces
- restricts libraries that can be added

Why Android is not a Linux distro

Enforced security policy (selinux, etc)

- the policy is written for use cases supported by Google (phone, tablet, TV, IVI)
- enforced at build-time by soong
- enforced at run-time by Android init
- can’t be disabled

Blinkered build system

- soong does not understand autotools, or even Makefiles, and so is not able to build upstream projects directly

AOSP is not a stable platform

Internal ABIs, sepolicy, and linker namespaces are all liable to change from one release to the next

Any code you add that depends on these things will likely break the next time round

The only ABIs that are stable are

- support for Android applications
- the vendor HAL (Hardware Abstraction Layer)

How far can I go with Embedded Android?

I define **Embedded Android** as AOSP running on a custom board (I am excluding off-the-shelf phones, tablets and TVs)

Assuming you have such a board

- don't mess with the base OS: you will introduce bugs and it will be hard to move the patches to later releases
- add native services (daemons) if you have to (*)
- add system services (extensions to the Android OS) if you have to (*)
- keep as much as possible in the application APK and use NDK to bundle libraries and low level stuff

(*) in both cases there will be some breakage from one release to another, but so long as your additions are simple and self contained it is likely acceptable

- Yocto and Android?
- Overview of Yocto Project
- Overview of AOSP
- Development process
- Community
- Conclusion

Development process: Yocto Project

The Yocto Project has a very open development process

- you can see all commits to the Yocto Project repositories
- you can contact the developers and repost bugs via the mail lists
- there is a weekly project status report

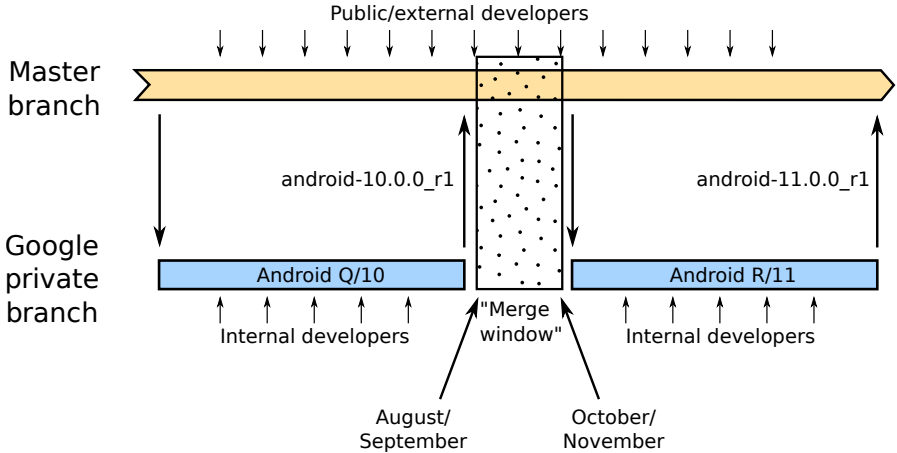
Development process: AOSP

In contrast, AOSP has a very opaque development process

- closed development
- roadmap is not published
- the publicly visible git main branch is not the next release!

What you get is what you get

How is AOSP developed?



- Yocto and Android?
- Overview of Yocto Project
- Overview of AOSP
- Development process
- Community
- Conclusion

Community: Yocto Project

Yocto has a strong, mutually supporting community

The development is done in the open

the goals are set by a steering committee

contributions are invited from all

Community: AOSP

AOSP, once again, is at the opposite end of the spectrum to Yocto

driven by the commercial needs of the main contributor, Google

development done behind closed doors

difficult to contact Android developers: open forums (e.g. Android Porting[1]) are not regularly monitored

[1] https://groups.google.com/g/android-porting

Resourcing

Both Yocto and Android are developed at a loss - there are no license fees for either

But Android provides a (large) revenue stream to Google in various indirect ways, so it makes sense for Google to sponsor the AOSP developers

Meanwhile, Yocto survives on sponsorship from Linux Foundation and donations

The net effect is that Android evolves much more quickly, but only in the direction that is compatible with the commercial goals of Google

- Yocto and Android?
- Overview of Yocto Project
- Overview of AOSP
- Development process
- Community
- Conclusion

Conclusion

With AOSP, what you get is what you get. If it fits your use case well, then go ahead

But if not, then you can waste a lot of time hacking around on AOSP, when a ground up distro built by Yocto would have been much easier

Maybe we should all give community based projects a little love, and try to promote awareness up the decision tree of whichever organisation we work for

Questions?

Slides at
https://2net.co.uk/slides/elc2021/yocto-and-android-csimmonds-elc-2021.pdf