How to combine Debian and Yocto/Bitbake?

Manuel Traut <manuel.traut@linutronix.de>
ELCE 2017 - Prague
What's next?

1. Why Yocto?
2. About Debian
3. Benefit of a combination
4. Existing solutions
5. Perfect combination
1) Why Yocto?

- What is Yocto
- Typical usage
- Limitations
What is Yocto?

- Tooling for building your own Linux distribution and SDK
- Defines a format that eases sharing of compile recipes and patches
- Powerful configuration management for different but similar images
- Based on Open-Embedded
- Example distribution “Poky” available
Typical usage

- Use Poky example distribution
- Add meta-layers from chip and/or hardware vendor
- Add 3rd party layers, e.g. for QT5
- Add own layer with image customization and own applications
Limitations

- Recipes from different layers might be incompatible
- Packages need to be built before they can be used
- Quality of recipes is hard to verify
- Security tracking/updates need to be done
- No LTS/updates available
- Reproducibility is not completely given (host dependencies)
2) About Debian

- The universal OS
- Debian and embedded?
- Usage
- Limitations
The universal OS

- Debian provides more than a pure OS, it comes with over 51,000 packages.
- The infrastructure, documentation, and build-tools are open-source.
- Debian takes security very seriously.
- Many security advisories are coordinated with other free software vendors and are published the same day a vulnerability is made public.
Debian and embedded?

Packages are available for

amd64  arm64  armel  armhf  i386  mips  mips64el  mipsel
powerpc (not in stretch)  ppc64el  s390x

Also **cross-toolchains** for different architectures are available in Debian/stretch
Usage

- Debootstrap embedded RFS (e.g. for arm) into a directory
- Use pbuilder or a cross-compiler to build own applications and copy to RFS-dir
- Remove unneeded files (man-pages, i18n, ...) from RFS-dir
- Build FS (ext4, etc) or disk / UBI images using some tools and scripting
- Extract licence information and retrieve source-code of all used packages
Limitations

- Only limited number of HW architectures supported
- No HW specific binary packages like special gstreamer plugins are available

The following limitations are given in Debian but are solved with E.L.B.E.

- SDCard / UBI / etc. image generation
- SDK generation and licence information and source package extraction
- Reduce image footprint
- Own application integration
3) Benefit of a combination

★ Yocto + Debian = ?
Yocto + Debian = ?

Use from Yocto

- Task scheduling
- Configuration management
- (cross-) compile from source if necessary
- SDK generation

Use from Debian

- Well maintained packages
- Security tracking
- Binary packages if available and useful
- Source packages if necessary
4) **Existing solutions**

- meta-debian
- meta-isar
- nmeta-elbe
- Comparison table
meta-debian

- ~600 .bb recipes, using sources from Debian/jessie
- Build rules optimized for embedded and retrieved from 'debian/rules'
- Long-term Linux kernel from CIP (Civil Infrastructure Project)
- Supports SDK generation
- Very active ~2000 commits on github
- Not compatible with existing Debian binary packages
meta-isar

- Uses Debian binary packages from stretch, jessie, wheezy or raspian-jessie
- Optional: building Debian source packages in a chroot (with qemu-user)
- Needs 'sudo' with nopasswd for several tasks
- Default image size ~300MB
- ~100 commits on github
nneta-elbe

- Proof of concept E.L.B.E. frontend (9 commits on github)
- Uses Debian stretch binary pkgs (tested with armhf)
- Optional: build binary pkgs from source within elbe-pbuilder
- Source and binary pkgs built with pbuilder available in a signed Debian repository
- Bitbake generates elbe-xml and schedules elbe-pbuilder and elbe-image-build jobs
- Builds licence information
- SDK generation currently not implemented, but easy because available in E.L.B.E.
<table>
<thead>
<tr>
<th>Feature</th>
<th>meta-debian</th>
<th>meta-isar</th>
<th>nmeta-elbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yocto-style config management &amp; app integration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW-specific SW like kernel / bootloader buildable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use Debian sources</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default footprint / reducible?</td>
<td>10MB</td>
<td>300MB / with Yocto methods</td>
<td>300MB / not yet</td>
</tr>
<tr>
<td>Non-Debian archs buildable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>For specific HW optimized binaries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Export used source code</td>
<td>download dir</td>
<td>easy to develop</td>
<td></td>
</tr>
<tr>
<td>SDK generation available</td>
<td>easy to develop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate licence information</td>
<td>csv</td>
<td>XML &amp; plain-text</td>
<td></td>
</tr>
<tr>
<td>Reproducibility</td>
<td>pkg v. by git tags</td>
<td>no VM / shared chroot for all builds</td>
<td>VM / pbuilder</td>
</tr>
<tr>
<td>Bitbake file per Debian package needed</td>
<td>+ a git repo</td>
<td>not for bin-pkg</td>
<td>not for bin-pkg</td>
</tr>
<tr>
<td>Use Debian binary packages</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of available Debian packages</td>
<td>limited / ~600</td>
<td>all</td>
<td>all</td>
</tr>
<tr>
<td>Effort needed to adapt buildsystem to new Debian release</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate signed Debian repos of self built packages</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5) Perfect combination

★ My personal wishlist
★ Conclusion
★ Your ideas
My personal wishlist

- Collaborate with 'rebootstrap.sh'
  to bootstrap Debian with settings from Bitbakes machine config
- Use Debian multiarch for cross-compiling any (modified) src pkg
  for a self bootstrapped architecture
- Mix usage of cross-built Debian pkgs via Bitbake
  with official Debian binary pkgs (for official supported architectures)
- Having reproducible builds for all Debian packages
Conclusion

- 3 implementations but only 2 use-cases
  - meta-debian is good for architectures that are NOT available in Debian
  - meta-isar and nmeta-elbe can only be used if the architecture is available in Debian
  - nmeta-elbe is a proof-of-concept but it's already very powerful thanks to the E.L.B.E backend

- Porting Debian bootstrapping to Bitbake might be interesting for Debian and Yocto
Your ideas

To improve the usage of Debian in embedded Linux projects?
References

nneta-elbe / E.L.B.E.

http://elbe-rfs.org

http://github.com/linutronix/nneta-elbe

http://github.com/linutronix/nneta-elbe-extended


meta-isar

http://github.com/ilbers/meta-isar


meta-debian

http://github.com/meta-debian

https://elinux.org/images/2/2e/MiniDebianConfJap an-Yoshi.pdf

Debian

https://wiki.debian.org/HelmutGrohne/rebootstrap

https://wiki.debian.org/ReproducibleBuilds
Contact

Manuel Traut
<manuel.traut@linutronix.de>

Linutronix GmbH
Bahnhofstraße 3
88690 Uhldingen
Germany