Building Debian-Based Products: Experiences in Collaboration

Kazuhiro Hayashi, Toshiba Corporation
(Baurzhan Ismagulov, ilbers GmbH)
Japan Technical Jamboree 61
Jun 23, 2017
Motivation

• Deby and Isar:
  – Both use Debian
  – Have common goals

• Seek working with community

• Benefits
  – Avoid effort duplication
  – Achieve more
Contents

• What is Deby
• What is Isar
• Comparison
• What we do
• Summary
What is Deby?

• A reference Linux distribution for embedded system
• “Shared Embedded Linux Distribution” project
  – One of the activities of CELP (Core Embedded Linux Project)
    • https://www.linuxfoundation.jp/projects/core-embedded-linux
  – Goals
    • Create an industry-supported embedded Linux distribution
    • Provide supports for long term
• Based on the two projects
  – Debian GNU/Linux
    • Cross-built from Debian source packages
  – Yocto Project
    • Cross-built with Poky build system and metadata for Debian source packages (meta-debian)
• Origin of the name
  – Debian + Poky
  – Debian-like
Deby: Purposes

• Providing features required in embedded systems, including civil infrastructure
  – Stability
    • Well-tested software set
  – Long-term support
    • 10+ years, especially for security fixes
  – Customizability
    • Changing configure options, compiler optimizations, etc.
  – Wider hardware support

• Contribution and collaboration with other communities
  – Debian, Debian-LTS
  – Yocto Project
  – Similar Debian-based projects like Isar
Deby: How it works

Debian source packages

- Fetch
- poky (Recipes)
  - meta-x (Custom layer)
  - meta-debian
    - A.bb
    - B.bb
    - C.bb
    - debian-package.bbclass
- meta (OE-Core)
  - .conf
  - .bbclass
  - .bb

Extra sources

- Hello

Cross build (bitbake)

- sysroots (Shared binaries)

apt repository

- A.deb
- B.deb
- C.deb
- hello.deb

apt-get install (bitbake)

- rootfs
- SDK
- kernel

X.bb defines how to build Debian source package “X”

Common function for Debian sources

Same buildflow as poky’s
Deby: How to use

• Repository
  – https://github.com/meta-debian/meta-debian

• Quick start

• Example: Build the minimal images and run on QEMU

```bash
$ git clone -b morty git://git.yoctoproject.org/poky.git
$ cd poky
$ git clone -b morty https://github.com/meta-debian/meta-debian.git
$ cd ..
$ export TEMPLATECONF=meta-debian/conf
$ source ./poky/oe-init-build-env
$ bitbake core-image-minimal
$ runqemu qemux86 nographic
```
Deby: Current development status

<table>
<thead>
<tr>
<th><strong>Debian version</strong></th>
<th>8 jessie (the latest stable)</th>
</tr>
</thead>
</table>
| **Yocto Project version** | 2.2 morty (stable)  
                           | 2.3 pyro (development) |
| **Kernel**           | 4.4 LTS / 4.4 CIP            |
| **BSP**              | QEMU: x86 (32bit, 64bit), ARM, PowerPC, MIPS  
                           | BeagleBoard, PandaBoard, MinnowBoard  
                           | BeagleBone Black, Raspberry Pi 1/2, Intel Edison |
| **init manager**     | busybox, systemd            |
| **Package manager**  | dpkg / apt                   |
| **Supported packages** | Approx. 600                   |
What is Isar?

- **Image generation for embedded systems**
  - Installs Debian binary packages as a base system
  - Builds and installs product’s software packages
  - Creates ready-to-use firmware images
  - Just a build system, **not a distribution**

- **Origin**
  - Predecessor system at Siemens
  - Developed by ilbers GmbH
  - Sponsored by Siemens

- **Uses:**
  - BitBake: Recipes for building and installing packages
  - Yocto: Structure, layering, workflow (**doesn’t rely on poky** code base)
  - Debian: Binary packages (**not included in Isar**)

- **Name**
  - Integration **System for Automated Root filesystem generation**
  - A river in Munich
Isar: Goals

• **Product build system**
  – One-command, on-demand building
  – Reproducibly create ready-to-use firmware images
  – Integrate product applications and customizations
  – Multiple upstreams, **multiple products**, strong reuse
  – Easy for beginners, familiar and powerful for advanced

• **Customer requirements**
  – Low effort: Native builds, no massive changes to upstream packages
  – Scale from small to big
  – Security updates
  – Maintenance: 10+ years
  – Legal clearing
Isar: How it works

- **Debian apt**
  - Create armhf build chroot
- **hello.git**
  - Build custom packages
  - Create armhf rootfs
  - Install custom packages
- **buildchroot**
  - **hello.deb**
  - **rootfs**
  - **U-Boot**
  - **kernel**
  - **Create target image**
  - **isar-image-base**
Isar: How to use

- Repository
  - [https://github.com/ilbers/isar](https://github.com/ilbers/isar)

- Quick start

- Example: Build a minimal image and run under QEMU

```bash
$ su -c "apt-get install dosfstools git mtools multistrap parted python3 qemu qemu-user-static sudo"
$ su -c "echo -e $USERYYYYtALL=NOPASSWD:¥ ALL >>/etc/sudoers"
$ git clone [https://github.com/ilbers/isar](https://github.com/ilbers/isar)
$ cd isar
$ . isar-init-build-env ../build
$ bitbake isar-image-base
$ start_armhf_vm  # User: root, password: root
```
## Isar: Current development status

<table>
<thead>
<tr>
<th><strong>Debian versions</strong></th>
<th>8 “Jessie”, 9 “Stretch”</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architectures</strong></td>
<td>i386, amd64, armhf</td>
</tr>
<tr>
<td><strong>Boards</strong></td>
<td>QEMU: pc (i386, amd64), virt (armhf) Raspberry Pi, Siemens Nanobox</td>
</tr>
<tr>
<td><strong>Boot</strong></td>
<td>U-Boot, grub, rpi boot loader, UEFI</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Disk image, filesystem image, ...</td>
</tr>
</tbody>
</table>
| **Base system**     | Debian-based distro (not a part of Isar), e.g.:  
  - Debian:  
    - Init: sysvinit, busybox, systemd  
    - Package manager: dpkg, apt  
    - Source packages: 25432 (Stretch)  
  - Raspbian: ...  
  - ... |
## Comparison of Isar and Deby

<table>
<thead>
<tr>
<th></th>
<th>Isar</th>
<th>Deby</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base system</strong></td>
<td>Debian <strong>binary</strong> packages (<strong>no rebuilding</strong>)</td>
<td>Binary packages <strong>cross-built</strong> from Debian <strong>source</strong> packages</td>
</tr>
<tr>
<td><strong>Build system</strong></td>
<td><strong>bitbake</strong></td>
<td><strong>poky</strong> (bitbake + OE-Core)</td>
</tr>
<tr>
<td><strong>Host tools</strong></td>
<td>Debian: multistrap, dpkg-buildpackage, qemu</td>
<td><strong>poky</strong></td>
</tr>
</tbody>
</table>
| **Metadata** (bitbake recipes) | ✓ Class and recipes for building product packages  
✓ Recipes for image generation  
✓ Debian packages **not included** | ✓ Common function to unpack Debian source packages (debian-package.bbclass)  
✓ **Full recipes** for cross-building every Debian source package |
| **Compilation**     | Native                                                                | Cross                                                                |
| **Benefits**        | ✓ Re-use Debian binaries and QA  
✓ Fast (re-use, parallel builds)  
✓ Lower development costs | ✓ Affinity with Poky recipes  
✓ Fully customizability  
✓ No need to keep binary pkgs |
| **Common features** | ✓ Based on Debian packages (stability, long-term maintenance)  
✓ Build packages and images with bitbake recipes  
✓ Generate images by installing binary packages  
✓ Manage multiple products as a set of layers |
Deby: Interaction points

Debian

- ack-grep
- acpid
- ...
- zip
- zlib

Yocto Project

- bitbake
- oe-init-build-env
- runqemu
Isar: Interaction points

Debian
- 0ad
- 0ad-data
- ... zzz-to-char
- zzzseekspinx

Yocto Project
- bitbake
- wic
- oe-init-build-env
- runqemu

Isar
- bitbake
- wic
- isar-init-build-env
- runqemu
### History of Debian-based projects

<table>
<thead>
<tr>
<th>Year</th>
<th>debian-cross (cross)</th>
<th>Emdebian</th>
<th>Debian (native)</th>
<th>SLIND (cross)</th>
<th>Isar (native)</th>
<th>Deby (cross)</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2006</td>
</tr>
<tr>
<td>2009</td>
<td>v1.0</td>
<td>v1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2009</td>
</tr>
<tr>
<td>2013</td>
<td>ML created</td>
<td>v3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2013</td>
</tr>
<tr>
<td>2014</td>
<td>rebootstrap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2014</td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v0.1</td>
<td></td>
<td>2015</td>
</tr>
<tr>
<td>2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>v0.2</td>
<td></td>
<td>2016</td>
</tr>
</tbody>
</table>

- **Contribution**: Use of `debian/rules` and `Cross-building`.
- **1st release**: Published.
- **Share ideas, code**: Rebootstrapped.

---

**Notes**:
- Year 2013: ML created, rebootstrapped.
- Year 2015: Published, v0.1 and v0.2.
- Year 2016: Rebootstrapped, v0.1 and v0.2.

---

**Additional Notes**:
- Debian-based projects:
  - debian-cross: Cross-building.
  - Emdebian: ML created, rebootstrapped, v1.0.
  - Debian: v1.0, v3.1.
  - SLIND: v0.1, v0.2.
  - Isar: v0.1.
  - Deby: v0.2.

---

**Japan Technical Jamboree 61**
Ideas for collaboration

• As the first step
  – Share the current benefits and issues of the both projects
  – Find features that could be shared
  – Create a proof of concept of the common features
  – List up issues, then define the next iteration

• Main topics
  – Both projects build Debian packages. Build time for subsequent builds can be improved by re-using previous build results
    • Binary package caching
      – Massive changes like cross-building is better done as a community
    • Cross-building of packages
      – Both projects require features to summarize license information in generated images
    • Support license clearing
Binary package caching 1/2

• Motivation
  – Improve build time by re-using previous build results

• Common features
  – After building a package: Save built packages for later use
  – Before building a package: If a pre-built version exists, skip building
  – During package installation: Install from the project’s apt repo

• Approach
  – Share functions to re-use built packages
  – Goal: Implement a common layer providing binary package caching

• What we did
  – Isar released the first implementation of binary package caching
  – Deby implemented a proof of concept of binary package caching, referring to the results of Isar
Lessons learned

- Deby
  - Requires two architectures (not only target but also native)
    - Poky always builds native binaries required for cross-building
  - Need to adapt binary package caching to sysroots
    - All built binaries are shared in sysroots for building others
    - Isar: Very divergent code bases, much glue, little common code

Next steps

- Deby
  - Design ways how to support multiple architectures and adapt sysroots in binary package caching
  - Or, consider changing the current sysroot based build flow to another one which has better affinity with Debian packages
    - Isar: Propose a common layer
Isar: Scripts

Debian
- 0ad
- 0ad-data
- ...
- zzz-to-char
- zzzzeeksphinx

Isar
- bitbake
- wic
- isar-init-build-env
- runqemu

Yocto Project
- bitbake
- wic
- oe-init-build-env
- runqemu
Cross-building of packages 1/3

• Motivation
  – Isar
    • Experience in cross-building Debian packages
  – Deby
    • Developing and maintaining full recipes for cross-building Debian packages without debian/rules costs too much
    • Planning to cross-build packages with debian/rules in recipes (.bb)
      – Implement common functions to handle debian/rules
      – Create patches for debian/rules to support cross-building
  – Debian 10 (buster)
    • A lot of efforts to support cross-building in debian/rules
    • Discussed in https://lists.debian.org/debian-cross/
Cross-building of packages 2/3

• **Common features**
  – debian/rules based package build (Deby: planning)
  – Supporting cross-build in community makes big sense

• **Approach**
  – Share existing resources for supporting cross-building
  – Contribute to debian-cross
    • Support cross-building not in-house but in Debian community

• **What we did**
  – Isar provided examples of
    • Common function (.bbclass) to cross-build Debian package
    • Source packages with patches to support cross-building
  – Deby
    • Implemented proof-of-concept recipes which cross-build packages with debian/rules, referring to the example of Isar
    • Identified 2191 of 3035 packages that don’t support cross-building
    • Added cross-building to libxinerama, reported [#861073](https://bugs.debian.org/861073)
Cross-building of packages 3/3

• **Lessons learned**
  - Deby
    • debian/rules of several packages in Debian buster work with the Deby’s cross toolchain without modification
    • Issue: debian/rules depends on commands and data in native system ignoring sysroots
  - Isar:
    • Initially released native building under QEMU to avoid massive changes; re-adding cross-building due to performance
    • ELBE reports issues with distcc, good experiences with icecc

• **Next steps**
  - Deby
    • Consider new design to adapt debian/rules to sysroots
    • Keep creating patches for debian/rules to support cross-building
  - Isar
    • Merge cross-building
    • Implement automatic cross-dependency installation in a Debian way
Deby: Interaction points (Current)

Debian

- ack-grep
- acpid
- ...
- zip
- zlib

Yocto Project

- bitbake
- oe-init-build-env
- runqemu

Deby
Deby: Interaction points (Future)

Debian

- ack-grep
- acpid
- ...
- zip
- zlib

Yocto Project

- bitbake
- oe-init-build-env
- runqemu

Deby
Support License Clearing 1/2

• Motivation
  – As general issues, examining and summarizing license information in generated images take time and require carefulness
  – As long as using the same Debian source packages, such efforts should be shared in related projects

• Approach
  – Share results of license examining and summarizing by using the common tools
    • Improve the quality of the output
    • Reduce costs for examining and summarizing
  – Support machine readable license data in Debian package level
    • DEP-5 formatted debian/copyright
      – [https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/](https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/)
    • First, keep accurate license data in Debian community
      – Contribute to Debian by posting patches for debian/copyright
    • Second, effectively summarize license information according to debian/copyright by sharing common tools
Support License Clearing 2/2

• What we did
  – Setup tools for investigating and summarizing license information
    • Scanning & Clearing: FOSSology
    • Summarizing: sw360
  – Provided DEP-5 copyright for zlib, reported #862260
    • Initial output from FOSSology, manual editing

• Lessons learned
  – Need to clear licenses and copyright holder name in “debian” directory even if no copyright holder name is detected by scanning tool

• Next steps
  – Keep posting patches for debian/copyright to support DEP-5 with clarifying policies of contribution
  – Share the tools and results of license investigation for Debian packages with related projects
  – Work with sw360 and ELBE on BoM and release notes generation
Summary

- **Common goals**
  - Package building, image generation and customization, licensing support

- **Divergent goals**
  - Deby: Max customizability
  - Isar: Min modifications

- **Current and future work**
  - Converge towards debian/rules and cross-building
  - Provide tools to support license clearing
  - Cross-building: Provide patches to Debian
  - Licensing: Move to DEP-5 and provide patches to Debian

- **Lessons (re-)learned**
  - Provide an implementation
  - Upstream your work
  - Bigger changes require community work
  - Providing a common layer for disparate code bases is a challenge
  - Proper license clearing costs time
  - Performance does matter