Debian + YoctoProject Based Projects: Collaboration Status

Kazuhiro Hayashi, Toshiba Corporation
Japan Technical Jamboree 63
Dec 1, 2017
Background

• 3 Debian-based projects for embedded products
  – ELBE: https://elbe-rfs.org/ [1]
  – Deby: https://github.com/meta-debian/meta-debian
  – Developed individually

• Introduced in ELCE 2016 @ Berlin
  – The three projects have
    • Common features
    • Different approaches
      – e.g. native-build v.s. cross-build

• Started collaboration
  – Made presentation in OSSJ2017 [3]

• Met again in ELCE 2017 @ Prague
  – Status updates of ELBE [4]
  – Collaborative discussion

http://events.linuxfoundation.org/events/archive/2016/embedded-linux-conference-europe/program/schedule
Motivation

• Common requirements
  – Stability
  – long-term maintenance
  – Reproducibility
  – Customization for embedded systems
  – License clearing

• Based on the same resources
  – Debian packages
  – Debian tools (e.g. debootstrap)
  – Bitbake, OE-Core

• Seek working with community

• Benefits of collaboration
  – Avoid effort duplication
  – Achieve more
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](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 ELBE and Isar
Deby: How it works

Debian source packages

A  B  C  …

Fetch

poky (Recipes)

meta-x (Custom layer)

C.bbappend  hello.bb

meta-debian

debian-package.bbclass

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

Common function for Debian sources

meta (OE-Core)

Cross build (bitbake)

sysroots (Shared binaries)

apt repository

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

Same buildflow as poky’s

apt-get install (bitbake)

rootfs  SDK  kernel
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
```
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

Extra sources

Fetch

isar (Recipes)

meta-isar

hello.bb

dpkg.bbclass

meta (Isar’s own)

build.sh

buildchroot.bb

dpkg-buildpackage + QEMU

buildchroot (Shared binaries)

apt repository

hello.deb

Debian apt repository

hello.deb

rootfs

multistrap

multistrap

isar-image-base.bb

Extra sources

hello

Debian apt repository
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
```
ELBE

• **Image generation tool for embedded systems**
  – Create bootloader, kernel, and rootfs images for specific architecture
  – Build natively on “build VM” (chroot, QEMU)
  – Use pre-build Debian binary packages directly
  – Customize everything by defining one XML file

• **OE-Core adaptation available**
  – [https://github.com/Linutronix/nneta-elbe](https://github.com/Linutronix/nneta-elbe)
  – Work as front-end of ELBE
    • Control ELBE core functions
    • Automatically generate ELBE XML from bitbake variables

• **Name**
  – ELBE: The Embedded Linux Build Environment
ELBE: How it works

Build VM

Debian apt repository

Extra repo

hello

XML

poky (Recipes)

meta-elbe
Settings

meta (OE-Core)

Control

VM

Installation + customization

elbe-init

elbe-chroot

elbe-control

kernel

bootloader

rootfs

installer

VM image
ELBE: How it works

```xml
<ns0:RootFileSystem xmlns:ns0="https://www.linutronix.de/projects/Elbe"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    created="2009-05-20T08:50:56" revision="6"
    xsi:schemaLocation="https://www.linutronix.de/projects/Elbe dbsfed.xsd">
    <project>
        <name>ARMexample</name>
        <version>08.15</version>
        <description>full featured debian system</description>
        <buildtype>armel</buildtype>
        <mirror>
            <primary_host>debian.tu-bs.de</primary_host>
            <primary_path>/debian</primary_path>
            <primary_proto>http</primary_proto>
        </mirror>
        <url-list>
            <url>
                <url>
                    http://d
                </url>
            </url>
        </url-list>
        <noauth />
        <suite>wheezy</suite>
        <buildimage>
            <kinitrd>elbe-bootstrap</kinitrd>
        </buildimage>
    </project>
    <target>
        <hostname>myARM</hostname>
        <domain>tec.linutronix.de</domain>
        <passwd/foo</passwd>
        <console/ttyS0,115200</console>
        <package>
            <tar>
                <name>nfsroot.tar.gz</name>
            </tar>
        </package>
        <finetuning>
            <pkg-list>
                <pkg>bash</pkg>
                <pkg>openssl-server</pkg>
            </pkg-list>
        </finetuning>
    </target>
</ns0:RootFileSystem>
```
ELBE: How to use

• Repository
  – https://github.com/Linutronix/nneta-elbe

• Quick start

• Example: Build an image

```bash
$ git clone http://git.yoctoproject.org/git/poky
$ cd poky
$ git reset --hard 924e576b8930fd2268d85f0b151e5f68a3c2afce
$ git clone https://github.com/Linutronix/nneta-elbe
$ git clone https://github.com/Linutronix/nneta-elbe-ext
$ TEMPLATECONF=nneta-elbe/conf ./oe-init-build-env build-elbe
$ elbe initvm create
$ elbe initvm start
$ bitbake simple-image
```
# Comparison

<table>
<thead>
<tr>
<th></th>
<th>ELBE</th>
<th>Isar</th>
<th>Deby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yocto-style development</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Kernel / Bootloader</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Use Debian source package</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Default footprint</td>
<td>300MB</td>
<td>300MB</td>
<td>10MB</td>
</tr>
<tr>
<td>Non-Debian archs support</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Export used source code</td>
<td>N</td>
<td>N</td>
<td>Y (download dir)</td>
</tr>
<tr>
<td>Yocto-style SDK</td>
<td>N (possible)</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Generate license info</td>
<td>N (possible)</td>
<td>N</td>
<td>Y (csv)</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>VM, pbuilder</td>
<td>Shared chroot</td>
<td>Use git tags</td>
</tr>
<tr>
<td>Use Debian binary package</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td># of Available packages</td>
<td>All</td>
<td>All</td>
<td>Limited (600 .dsc)</td>
</tr>
<tr>
<td>Effort for adapting (update)</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Signed apt repository</td>
<td>Y</td>
<td>deb:N dsc:Y</td>
<td>N</td>
</tr>
<tr>
<td>Feature</td>
<td>ELBE</td>
<td>Isar</td>
<td>Deby</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Yocto-style development</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Kernel / Bootloader</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Use Debian source package</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Default footprint</td>
<td>300MB</td>
<td>300MB</td>
<td>10MB</td>
</tr>
<tr>
<td>Non-Debian archs support</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Export used source code</td>
<td>N</td>
<td>N</td>
<td>Y (download dir)</td>
</tr>
<tr>
<td>Yocto-style SDK</td>
<td>N (possible)</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Generate license info</td>
<td>N (possible)</td>
<td>N</td>
<td>Y (csv)</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>VM, pbuilder</td>
<td>Shared chroot</td>
<td>Use git tags</td>
</tr>
<tr>
<td>Use Debian binary package</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td># of Available packages</td>
<td>All</td>
<td>All</td>
<td>Limited (600 .dsc)</td>
</tr>
<tr>
<td>Effort for adapting (update)</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Signed apt repository</td>
<td>Y</td>
<td>deb:N dsc:Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Source: Japan Technical Jamboree 63
Ideas for collaboration

• **Topics**
  – Place to develop together
  – Policy to share our existing resources
  – What we need to change
  – Clarify requirements: Common / different goals
  – Debian way v.s. YoctoProject (OE-Core) way
  – Native build v.s. Cross build
  – License clearing and software component catalogue generation

• **Workflow**
  – 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
Collaboration status (1)

• Created a common repository
  – [https://github.com/manut/meta-combo](https://github.com/manut/meta-combo)
  – Project layers included
    • meta-elbe (for ELBE)
    • meta-debian (for Deby)
    • meta-isar (for Isar) => Under discussion
  – Imported from original repositories with combo-layer
Collaboration status (2)

• Shared ideas about how to develop the common layer
Collaboration status (2)

• Shared ideas about how to develop the common layer

Plan: Find out the features which could be shared and put them into meta-common

Plan: Under discussion
Idea: 1. Move Isar specific features to meta-isar
2. Put the common features in meta-isar into meta-common
Collaboration status (3)

• Clarify required features in our build systems
  – Debian binary package pool
    • Need to reuse binary packages to reduce the build time
    • Deby wants to avoid full-building
  – Native package builder
    • ELBE and Isar usually need native build which is officially supported in Debian (more stable than cross-building)
  – Cross package builder
    • Deby needs cross-building and is already doing now
    • ELBE and Isar sometimes want cross-building to build quickly big packages or kernels
  – Build environment for extra sources (Cross/Native)
    • Kernel, bootloader, application, etc.
  – Image builder
    • Generate rootfs and SDK from pre-built binaries in above builders
    • Provides customization and adaptation for embedded boards
  – YoctoProject-based layer structure in metadata
    • All recipes should be included in proper project (product) layer
Collaboration status (4)

• Deby: Provides new experimental metadata set
  – Purpose: Add new features required to satisfy our requirements with existing resources in ELBE/Isar
    • Currently designed for OE-Core infrastructure
  – Name: meta-debian-next
    • https://github.com/meta-debian/meta-debian/tree/next-poc/meta-debian-next
  – Very small, but the first step
  – Features
    • Provides a class file for cross-building Debian source packages with pbuilder through bitbake
    • Depends on meta-elbe
      – Reuse existing bitbake configuration in meta-elbe
Future plan

- **Deby (meta-debian-next)**
  - Implement remaining features (PoC)
    - Build environment for extra sources
      - Debian binary packages + Current Deby way (Yocto-based)
    - Image builder
      - Try to reuse existing functions in Isar / ELBE
- **meta-combo**
  - Create example that uses multiple layers
    - e.g. meta-debian + meta-elbe
  - Isar integration
- **Clarify our requirements (more)**
  - Concrete requirements in our products
  - What is needed in the market
  - Available resources for collaboration are not unlimited
    - Sharable benefits required to change our current ways
References


2. Isar Build Debian-Based Products with BitBake - FOSDEM 2017
