CE Workgroup
Shared Embedded Linux Distribution Project

Yoshitake Kobayashi
CE Workgroup, the Linux Foundation (Toshiba Corporation)
Embedded Linux Conference Europe 2015
5-7 Oct, 2015
Overview

- Introduction of CE Workgroup
- Introduction of Shared Embedded Linux Distribution Project
Introduction of CE Workgroup
The Linux Foundation
Role of CEWG

• **Bridge** between OSS (like Linux) developer communities and embedded system industry who wish to collaborate with those communities.
  • Building the relation of trust and co-creation is so essential to realize the value of OSS
  • However it is so difficult achieving such ideal relationship solely by any company because of the diversified, groval and huge scale of the active OSS communities
• CEWG is a community of people who belong to the industry that wish to become a citizen of grater OSS communities and perform co-creation of innovative software
Industry domain

(Past) CEWG worked mainly for consumer electronics industry
- Through more than 10 years activities many technological challenges of OSS development went well and recently OSS is commonly used in CE industry

(Now) CEWG extending the industry domain not only for consumer electronics but also for industrial appliances, medical equipment, transportation/traffic control system and more wider embedded system industry domain
- We are aiming at the wider use and collaboration of OSS for embedded systems
- CEWG will bridge to the OSS community and the unique technological challenges in each industry domain
Governance Structure of CEWG

- **Steering Committee**
  - Decide the budget, community supporting program and more

- **Architecture Group**
  - Discussion board of the OSS technologies for the embedded systems
  - Open Project candidates are selected by this group
Activities of CEWG (eLinux Wiki)

• You will find bunch of precious technological information in the eLinux Wiki
• CEWG supports for the site
• It is the technological information portal of OSS for embedded system developers
Activities of CEWG (Linux, Long term support)

• An extended support program based on industry requirement (Long Term Support Initiative) is carried out together with the Linux community
  • Generally Linux community terminate supporting Linux when “the next to next version” released (it is about 5 month). It had been a great headache for the industry people who require the longer term support.
  • The initiative pick up one release of Linux annually and extend the support term to 2 years
  • The automated test environment have also been prepared and we are aiming at more sophisticated quality
Activities of CEWG (Open Project)

• CEWG Open Project is a open community funding program to reinforce the OSS development which worth for embedded system developers
  • For example, we have funded a project up-streaming development result of Android to the original OSS development communities such as Linux community
  • Solicit the project open to everyone
  • The Architecture Group of CEWG select the solicited projects. Based on the Steering Committee approval, the funding will be carried out

• Some example of past projects
  • Device tree documentation, Linux-tiny, DirectFB enhancement, Squashfs
Activities of CEWG (CEWG Project)

• CEWG Project aim to study and to solve issues on embedded system

• Linux in Civil Infrastructure (was called Social Infrastructure)
  • Goals: Solve problems with Linux for use in civil infrastructure systems
  • Status: Project scope presented at ELC 2015 and LinuxCon Japan 2015

• Shared Embedded Distribution
  • Goals: Create an industry-supported distribution of embedded Linux and provide support for long term
  • Status: Created an Yocto layer by using Debian source code
  • Presented at ELC, LCJ and ELCE

• Device Mainlining
  • Goals: Study obstacles to mainlining, and work to reduce obstacles
  • Status: SIG meetings at ELCE and ELC
  • Presentations about overcoming obstacles at ELCE 2014, ELC 2015, and LCJ 2015
  • White paper (published at LCJ – June 2015)
Introduction of Shared Embedded Distribution

• For more information about this project
  – Shared Embedded Linux Distribution
    • http://elinux.org/Shared_Embedded_Linux_Distribution
  – CE Workgroup Linux Foundation
    • http://www.linuxfoundation.org/collaborate/workgroups/celf
Motivation

• Linux is running on many kind of embedded systems
  – Including the systems in civil infrastructure

• Things to be considered to choose a base distribution
  – The number of supported packages
  – Package versions
  – Supported hardware
  – Stability, number of bugs were fixed
  – The frequency of security updates and supported timespan
  – How to compile and customize packages
In our case

• **What we want to do**
  – Make custom embedded Linux environments

• **What we need**
  – Wider hardware support
  – Stability
    • Well tested packages are required
    • Many embedded developer are still want to use stable version
  – Long-term support
    • Over 15 years support required, especially for security fixes
    • (This is what we would like to contribute something)
  – Fully customizable build system
Our solution

**Yocto Project "poky"**
- One of the most popular reference distributions for embedded Linux
- Fully customizable build system
- Supports numerous embedded boards including modern ones
- Can be extended by meta-layer

**Debian GNU/Linux**
- Support many kind of CPUs: x86, ARM, PowerPC, MIPS (32bit/64bit)
- Release a stable version after two years of testing
- Long-term support for 5 years by Debian-LTS project

**meta-debian**
What is meta-debian?

- A meta layer for the Poky build system

Main feature
- Allows cross-building Linux images using Debian source packages

Implemented as an independent "layer"
- Completely separated from OpenEmbedded-Core and other layers
- Source code: https://github.com/meta-debian/meta-debian.git
Build system structure (poky)

Upstream source code

Fetch

poky build system

meta (OpenEmbedded-Core)

Board-specific metadata

Build

A  B  C

A  B  C
Build system structure (poky + meta-debian)

Upstream source code

Debian source packages

Fetch

poky build system

meta-debian

meta (OpenEmbedded-Core)

Board-specific metadata

A

B

C

Build

A

B

C
Target versions of meta-debian

Upstream source code

Debian source packages

Debian 8.0 jessie

poky build system

Fetch

Yocto Project 1.6 daisy

meta-debian
meta (OpenEmbedded-Core)

Board-specific metadata

Build

A
B
C

A
B
C
Purpose of meta-debian

• **Create embedded Linux environments with**
  – Wide embedded CPU support
  – Stability
  – Long-term support
  – Fully customizable build system

• **Provide a common place for developers having the same needs**

• **Contribute to upstream (Debian and Yocto Project)**
  – Especially for the Debian LTS project

With Debian stable release + LTS

With poky build system
Quick start

1. Download the build tools
2. Setup build directory
3. Build minimal Linux image
4. Run minimal Linux image on QEMU

• See also meta-debian/README
Download build tools

• Download poky

```bash
$ git clone git://git.yoctoproject.org/poky.git
$ cd poky
$ git checkout daisy
```

• Download meta-debian into the poky directory

```bash
$ cd poky
$ git clone https://github.com/meta-debian/meta-debian.git
$ cd meta-debian
$ git checkout daisy
```

← meta-debian specific step
Setup build directory

- Change the default configuration
  - Enable meta-debian layer
  - Enable "debian" distro (DISTRO = "debian")
  - The default target machine is "qemux86" (MACHINE = "qemux86")
  - TEMPLATECONF is used by oe-init-build-env script

  $ export TEMPLATECONF=meta-debian/conf

- Run startup script
  - This setup a build directory and environment variables automatically
  - (builddir): name of build directory (optional)

  $ source /path/to/poky/oe-init-build-env (builddir)
Build minimal Linux image

- Run bitbake

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

- Built images (case of qemux86)
  - Output directly
    - /path/to/builddir/tmp/deploy/images/qemux86
  - Kernel
    - bzImage-qemux86.bin
  - Root filesystem
    - core-image-minimal-qemux86.ext3
    - core-image-minimal-qemux86.tar.gz
Run minimal Linux image on QEMU

- Run built images on QEMU environment
  - `qemux86`
    ```
    $ runqemu qemux86 nographic bootparams="init=/init root=/dev/sda"
    ```
  - `qemux86-64`
    ```
    $ runqemu qemux86-64 nographic bootparams="init=/init root=/dev/sda"
    ```
  - `qemuarm`
    ```
    $ runqemu qemuarm nographic bootparams="init=/init console=ttyAMA0"
    ```
  - `qemuppc`
    ```
    $ runqemu qemuppc nographic bootparams="init=/init"
    ```
How should we create recipe files?

• We need to create new recipes for Debian sources
  – How?

• Possible solutions
  – Method 1: Modify OE-Core recipes directly
  – Method 2: Add recipes into a new layer
Method 1: Modify OE-Core recipes

- We already tried this way previously... but
- Not the ideal solution 😞
  - Original OE-Core recipes are no longer available (Modified)
  - Just a fork
    - It becomes hard to catch up with the newest poky versions
    - Difficult to convince other people to join our effort
Method 2: Add recipes into a new layer

- The best way to add new recipes for specific purposes
  - Original OE-Core recipes are still there
  - Can be developed independently of OE-Core
  - Enable / disable the layer easily like a module

![Diagram showing the poky build system with new layer added]
How should we create recipes in a layer?

• From scratch?
  – Often takes time!
  – Why?
    • Need to create patches for supporting cross-build in poky

• We should follow the existing OE-Core recipes
  – How?

poky build system

meta-debian

recipe A’ recipe B’ recipe C’ recipe X

meta (OpenEmbedded-Core)

recipe A recipe B recipe C recipe D

How to follow?
• Method 1: "Include" OE-Core recipes
• Method 2: Use a part of OE-Core recipes
Method 1: "Include" OE-Core recipes

• We used to use this method before…
• Unsuitable for our case 😞
  – Difficult to override some variables and functions
    • Ex: already appended (_append) or prepended (_prepend) data
  – Automatically follow "unneeded" OE-Core updates against our will
    • Ex: Shown in the next slides
Method 1: "Include" OE-Core recipes

- binutils
- openssl
Method 1: "Include" OE-Core recipes

- **binutils**
  - Security patches applied twice
- **openssl**
  - Target version was upgraded, and patches also upgraded
  - Some upgraded patches conflict with Debian source

Difficult to maintain 😞
Method 1: "Include" OE-Core recipes

- **binutils**
  - Security patches applied twice

- **openssl**
  - Target version was upgraded
  - Some upgraded patches conflict with Debian source

Each recipe should create for relative source code one by one.
Method 2: Use a part of OE-Core recipes

- Try to create recipes from scratch using Debian source packages
- Re-use the essential data from OE-Core
  - patches, variables, functions for supporting cross-build
How should we implement recipes?

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- How to installed files
- How to make the package file
- Package dependencies

- The following slide describes:
  - Method 1: just re-use OE-Core recipes
  - Method 2: Follow Debian’s packaging
Method 1: re-use OE-Core (not a good solution)

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- Installed files and paths
- How to package files
- Dependencies between others

Debian source

Recipe

files

Build

OE-Core based
Method 1: re-use OE-Core (not a good solution)

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- Installed files and paths
- How to package files
- Dependencies

WHO IS THIS?

Debian?
poky?
Method 1: re-use OE-Core (not a good solution)

• Bad results: conflicts of two distributions
  – Compile fails
    • Cause: missing configure options that Debian source requires
  – Some programs fail to call commands or load data file
    • Cause: installation paths differ from Debian’s

• Cannot be used like Debian

We should define some development "policy" for creating recipes
Policies for creating recipes

- **By default, follow Debian’s packaging**
  - i.e. debian/rules
  - For getting good affinity with Debian sources
- **Customize for embedded system if necessary**
  - Disable features
  - Remove dependencies
- **Re-use only essential data from OE-Core for supporting cross-compile**
  - See "Method 2: Re-use OE-Core recipes"
Method 2: Follow Debian’s packaging

- LICENSE information
- Required files
  - Source code
  - initscripts, configs
  - Patches
- Configure commands & options
- Compile commands & options
- Installed files and paths
- How to package files
- Dependencies between others

Debian source

Recipe

files

Customize for embedded

DEbian source

debian/rules

OE-Core based

Debian-based

Build

our solution
How to create recipes (Sample: zlib)

PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
    ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
    oe_runmake
}
do_install () {
    oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
    mkdir -p ${D}/${base_libdir}
    mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
    tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNME_${PN}-dbg       = "${PN}1g dbg"
DEBIANNME_${PN}-staticdev = "${PN}1g staticdev"
DEBIANNME_${PN}-dev       = "${PN}1g dev"
DEBIANNME_${PN}-doc       = "${PN}1g doc"
DEBIANNME_${PN}           = "${PN}1g"
Step 1: Add recipe revision

- Define recipe revision: \${PR}
- Increment every update

```
PR = "r0"
```

```
License = "Zlib"
LIC_FILES_CHKSUM = "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"
```

```
SRC_URI += "file://remove.ldconfig.call.patch"
```

```
do_configure () {
    ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
    oe_runmake
}
do_install () {
    oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
    mkdir -p ${D}/${base_libdir}
    mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
    tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}
```
Step 2: Inherit `debian-package.bbclass`

- **Setup Debian source package**
  - Define `SRC_URI`
  - Apply Debian’s patches (do_debian_patch)
Step 3: Add license information

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"
SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
    ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile() {
    oe_runmake
}
do_install() {
    oe_runmake
}
do_install_append_class-target() {
    mkdir -p ${D}/${base_libdir}
    mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
    tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg = "${PN}1g-dbgsym"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev = "${PN}1g-dev"
DEBIANNAME_${PN}-doc = "${PN}1g-doc"
DEBIANNAME_${PN} = "${PN}1g"
```

- **LICENSE**: License name
  - Common license names are found in `meta/files/common-licenses`
- **LIC_FILES_CHKSUM**: Checksum of the license text
  - Usually found in COPYING, LICENSE, or header of source files (.c, .h)
Step 4: Append patches

```
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = Y
  "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
    ./configure
    --shared
    --prefix=${prefix}
    --libdir=${libdir}
}

do_compile() {
    oe_runmake
}

do_install() {
    oe_runmake DESTDIR=${D} install
}

do_install_append_class-target() {
    mkdir -p ${D}/${base_libdir}
    mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
    tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sf ../../${base_libdir}/${tmp} ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g~dbg"
DEBIANNAME_${PN}-staticdev  = "${PN}1g~staticdev"
DEBIANNAME_${PN}-dev        = "${PN}1g~dev"
DEBIANNAME_${PN}-doc        = "${PN}1g~doc"
DEBIANNAME_${PN}            = "${PN}1g"
```

- Add patches into SRC_URI
  - Necessary for being built in cross-compile environment
  - Copied from OE-Core (or create it from scratch)
Step 5: Define configure options

- Define configure commands
  - The same options as debian/rules
  - Some features should be disabled for embedded

```bash
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
    ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile() {
    oe_runmake
}
do_install() {
    oe_runmake DESTDIR=${D} install
}
do_install_append_class() {
    mkdir -p ${D}/${base_libdir}
mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g dbg"
DEBIANNAME_${PN}-staticdev = "${PN}1g staticdev"
DEBIANNAME_${PN}-dev        = "${PN}1g dev"
DEBIANNAME_${PN}-doc        = "${PN}1g doc"
DEBIANNAME_${PN}            = "${PN}1g"
```
Step6: Define compile and install commands

```bash
PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
  ./configure --shared --prefix=${prefix} --libdir=${libdir}
}

do_compile () {
  oe_runmake
}
do_install() {
  oe_runmake DESTDIR=${D} install
}
do_install_append_class -target() {
  mkdir -p ${D}/${base_libdir}
  mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
  tmp=`readlink ${D}/${libdir}/libz.so`
  ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${ PN} -dbg = "${ PN}1g-dbg"
DEBIANNAME_${ PN} -staticdev = "${ PN}1g-staticdev"
DEBIANNAME_${ PN} -dev = "${ PN}1g-dev"
DEBIANNAME_${ PN} -doc = "${ PN}1g-doc"
DEBIANNAME_${ PN} = "${ PN}1g"
```

- Define compile & install commands
- `autotools.bbclass` often replaces them
Additional Steps: Change library paths

PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = "file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
    ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
    oe_runmake
}
do_install () {
    oe_runmake DESTDIR=${D} install
}
do_install_append_class-target() {
    mkdir -p ${D}/${base_libdir}
    mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
    tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sf ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbgsym"
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev        = "${PN}1g-dev"
DEBIANNAME_${PN}-doc        = "${PN}1g-doc"
DEBIANNAME_${PN}            = "${PN}1g"

Move run-time libraries to the same directory as Debian
Additional Steps: Change package name

- Change the default binary package name to Debian’s
- "libz" => "zlib1g"

PR = "r0"
inherit debian-package

LICENSE = "Zlib"
LIC_FILES_CHKSUM = 
"file://zlib.h;beginline=4;endline=23;md5=fde612df1e5933c428b73844a0c494fd"

SRC_URI += "file://remove.ldconfig.call.patch"

do_configure() {
    ./configure --shared --prefix=${prefix} --libdir=${libdir}
}
do_compile () {
    oe_runmake
}
do_install() {
    oe_runmake DESTDIR=${D} install
}
do_install_append_class -target() {
    mkdir -p ${D}/${base_libdir}
    mv ${D}/${libdir}/libz.so.* ${D}/${base_libdir}
    tmp=`readlink ${D}/${libdir}/libz.so`
    ln -sfn ../../${base_libdir}/$tmp ${D}/${libdir}/libz.so
}

DEBIANNAME_${PN}-dbg       = "${PN}1g-dbgn"  
DEBIANNAME_${PN}-staticdev = "${PN}1g-staticdev"
DEBIANNAME_${PN}-dev        = "${PN}1g-dev"
DEBIANNAME_${PN}-doc        = "${PN}1g-doc"
DEBIANNAME_${PN}            = "${PN}1g"
Build results (zlib packages)

Debian 8.0 jessie

zlib1g
- /lib/i386-linux-gnu/libz.so.1
- /lib/i386-linux-gnu/libz.so.1.2.8
- /usr/share/doc/zlib1g/

zlib1g-dev
- /usr/include/i386-linux-gnu/zconf.h
- /usr/include/zlib.h
- /usr/lib/i386-linux-gnu/libz.a
- /usr/include/i386-linux-gnu/pkgconfig/zlib.pc
- /usr/share/doc/
- /usr/share/man/
- /usr/lib/i386-linux-gnu/pkgconfig/zlib.pc

zlib1g-dbgsym
- /lib/libz.a

zlib1g-dbgsym
- /lib64z1*
- libn32z1*

Ignore non-essential files

meta-debian

zlib1g
- /lib/libz.so.1
- /lib/libz.so.1.2.8

zlib1g-dev
- /usr/include/zconf.h
- /usr/include/zlib.h
- /usr/lib/libz.a
- /usr/include/pkgconfig/zlib.pc

zlib1g-doc
- /usr/share/man/

zlib1g-staticdev
- /usr/lib/libz.a

zlib1g-dbgsym
Directory structure

- poky
  - meta
    - recipes-xxx
    - pkg
      - pkg_1.0.bb
        - files
    - classes
    - conf
  - meta-debian
    - recipes-xxx
    - pkg
      - pkg_debian.bb
        - files
      - debian-package.bbclass
    - classes
      - debian-package.bbclass
    - conf
      - layer.conf
      - distro
        - debian.conf
Core recipes
• Data for supporting cross-compile is partially copied from original recipes
Directory structure

- poky
  - meta
    - recipes-xxx
      - classes
      - conf
    - pkg
      - pkg_1.0.bb
        - files
  - meta-debian
    - recipes-xxx
      - pkg
        - pkg_debian.bb
          - files
      - classes
        - debian-package.bbclass
        - debian.conf

Provides debian specific functions and variables
- Fetch a source package automatically
- Apply Debian’s patches automatically
  - debian/patches/*
Defines configurations of distro
• Distro name = "debian"
• Common server URIs
• Features
• System managers
  • Ex: init manager = busybox
  • Ex: device manager = udev
Build flow

bitbake tasks

do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
......

Download directory: ${DL_DIR}

Working directory: ${WORKDIR}
Build flow

bitbake tasks

- do_fetch()
- do_unpack()
- do_debian_patch()
- do_patch()
- do_configure()
- do_compile()
- do_install()
- do_package()

......

Download directory: ${DL_DIR}

pkg.git

poky

localfiles

--

Check files

localfiles.done

quilt.git.done

Working directory: ${WORKDIR}
Build flow

bitbake tasks

do_fetch()
do_unpack()
do_debian_patch()
do_patch()
do_configure()
do_compile()
do_install()
do_package()
......

Download directory: ${DL_DIR}

Copy

git checkout

Working directory: ${WORKDIR}
Build flow

**bitbake tasks**
- do_fetch()
- do_unpack()
- **do_debian_patch()**
- do_patch()
- do_configure()
- do_compile()
- do_install()
- do_package()

......

Download directory: ${DL_DIR}

Working directory: ${WORKDIR}

Apply debian/patches/*
Build flow

bitbake tasks

- do_fetch()
- do_unpack()
- do_debian_patch()
- **do_patch()**
- do_configure()
- do_compile()
- do_install()
- do_package()

.....

Download directory: ${DL_DIR}

Working directory: ${WORKDIR}

Apply patches for supporting cross-build
Build flow

bitbake tasks

- do_fetch()
- do_unpack()
- do_debian_patch()
- do_patch()
- do_configure()
- do_compile()
- do_install()
- do_package()
......

Download directory: ${DL_DIR}

Configure & compile with same options as Debian

Install into the same paths as Debian

Working directory: ${WORKDIR}

Install directory: ${D}

Built files

git

pkg.git

poky

localfiles

localfiles.done

quilt.git.done

git2

pkg.git

localfiles

localfiles.done
Build flow

**bitbake tasks**

- do_fetch()
- do_unpack()
- do_debian_patch()
- do_patch()
- do_configure()
- do_compile()
- do_install()
- do_package()

......

Download directory: `${DL_DIR}`

Install directory: `${D}`

Working directory: `${WORKDIR}`

Built files

Package by the same way as Debian
Current status

• **Supported CPUs**
  - x86 32bit
  - x86 64bit
  - ARM
  - PowerPC

• **Kernel**
  - LTSI

• **User land**
  - busybox-based minimal system
  - Number of available packages: around 200
    • Recipe implementation is still ongoing
    • implementing more recipes to support other packages
<table>
<thead>
<tr>
<th><strong>Current development status</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Meta-debian</strong></th>
<th><strong>Current development status</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Poky version</td>
<td>Daisy</td>
</tr>
<tr>
<td>Debian version for source code</td>
<td>Debian 8 (Jessie)</td>
</tr>
<tr>
<td>Kernel</td>
<td>LTSI 3.10 and 3.14 + RT patch</td>
</tr>
<tr>
<td>Distribution Support</td>
<td>Debian 8 (Jessie)</td>
</tr>
<tr>
<td>Status</td>
<td>Under development</td>
</tr>
<tr>
<td>Number of packages</td>
<td>Approx. 200</td>
</tr>
<tr>
<td>BSPs</td>
<td>QEMU(X86, X86_64, ARM, PowerPC)</td>
</tr>
<tr>
<td></td>
<td>RaspberryPi</td>
</tr>
<tr>
<td></td>
<td>MinnowBoard</td>
</tr>
<tr>
<td></td>
<td>..etc.</td>
</tr>
</tbody>
</table>

- Recipe development is ongoing to increase packages
Conclusions

• **What is Shared Embedded Linux distribution**
  – Share the work of maintaining long-term support for an embedded distribution, by leveraging the work of the Debian project
  • Metadata for building embedded Linux systems using Debian source packages
  • Implemented as an independent layer of OpenEmbedded-Core

• **meta-debian is intended to provide**
  – Wide embedded CPU support
  – Stability
  – Long-term support
  – Fully customizable Linux
Conclusions

- **Policies for creating recipes**
  - Debian based configs & packaging + customization for embedded
    - For getting affinity with Debian sources
    - Re-use OE-Core data for supporting cross-build

- **Examples**
  - How to build & run a tiny Linux image
  - How to create recipes
Future work

- Support more features and packages (over 200)
  - LTSI + RT kernel, Wayland, Qt, etc.
- Support more embedded boards
- Testing: improve the quality of the system and packages
  - LTP, LTP-DDT, POSIX test suite, ptest, etc.
  - Contribute to the LTSI test project (JTA) \[3\]
- Keep following updates of poky and Debian
Please give us feedback

• **E-mail**
  – yoshitake.kobayashi@toshiba.co.jp
  – kazuhiro3.hayashi@toshiba.co.jp

• **Repository**
  – https://github.com/meta-debian/meta-debian.git
Questions?
References

1. Poky meets Debian: Understanding How to Make an Embedded Linux by Using an Existing Distribution's Source Code

2. LTSI Test Project
   – http://ltsi.linuxfoundation.org/ltsi-test-project

3. Yocto Project Manuals
Thank you
remove.ldconfig.call.patch

When /etc/ld.so.cache is writeable by user running bitbake then it creates invalid cache (in my case libstdc++.so cannot be found after building zlib(-native) and I have to call touch */libstdc++.so && /sbin/ldconfig to fix it.

So remove ldconfig call from make install-libs

Upstream-Status: Inappropriate [disable feature]

diff -uNr zlib-1.2.6.orig/Makefile.in zlib-1.2.6/Makefile.in
--- zlib-1.2.6.orig/Makefile.in 2012-01-28 23:48:50.000000000 +0100
+++ zlib-1.2.6/Makefile.in 2012-02-13 15:38:20.577700723 +0100
@@ -199,7 +199,6 @@
     rm -f $(DESTDIR)$(sharedlibdir)/$(SHAREDLIB)
 $(DESTDIR)$(sharedlibdir)/$(SHAREDLIBM); ¥
    ln -s $(SHAREDLIBV) $(DESTDIR)$(sharedlibdir)/$(SHAREDLIB); ¥
-   ln -s $(SHAREDLIBV) $(DESTDIR)$(sharedlibdir)/$(SHAREDLIBM); ¥
     $(LDCONFIG) || true) >/dev/null 2>&1; ¥
 fi
 cp zlib.3 $(DESTDIR)$(man3dir)
 chmod 644 $(DESTDIR)$(man3dir)/zlib.3
recipes-extended/newt/files/cross_ar.patch

... Makefile.in | 3 ++-
   configure.ac | 4 ++++
 2 files changed, 6 insertions(+), 1 deletion(-)

--- a/Makefile.in
+++ b/Makefile.in
@@ -7,6 +7,7 @@ CFLAGS = @CFLAGS@
   LDFLAGS = @LDFLAGS@
   CPPFLAGS = -D_GNU_SOURCE @CPPFLAGS@
   GNU_LD = @GNU_LD@
+AR = @AR@

   VERSION = @VERSION@
   TAG = r$(subst .,-,${VERSION})
@@ -95,7 +96,7 @@ whiptcl.so: $(WHIPTCLOBJS) $(LIBNEWTSH)
       $(CC) -shared $(SHCFLAGS) $(LDFLAGS) -o whiptcl.so $(WHIPTCLOBJS) -L. -lnewt $(LIBTCL) -lpopt $(LIBS)

   $(LIBNEWT): $(LIBOBJJS)
       ar rv $@ $^ 
+      $(AR) rv $@ $^ 

   newt.o $(SHARDDIR)/newt.o: newt.c Makefile
...
require qt4-libs.inc

PR = "r0"

QT_CONFIG_FLAGS = " ¥
  -v ¥
  -embedded ${QT_ARCH} ¥
  -release ¥
  -opensource ¥
  -make libs ¥
  -nomake tools ¥
  -nomake examples ¥
  -nomake demos ¥
  ...