yocto PROJECT is not only BitBake !

Pierre Ficheux (pierre.ficheux@smile.fr)

February 2020

Who am I

- CTO @ Smile ECS (Embedded & Connected Systems)
- Teacher and writer

Agenda

- Build system (for industrial use)
- Some Yocto reminders
- Building image for Pi
- Creating recipe
- Devshell
- Devtool (build patch for a recipe)
- Building kernel module inside Yocto
- Building and using eSDK (user, kernel)
- Using “ptest” and “testimage”
- Several demos !

Build system for industry

- Several tasks during an industrial project
  - Installing BSP (Yocto ?)
  - Creating SDK (for building apps)
  - Developing your apps (your job !)
  - System integration and maintenance
- A build system is NOT a development tool but creates one (SDK / cross toolchain)
- A build system is an integration tool, software should be “bug free” to be integrated
- A build system helps you for costly but necessary tasks :-)
- Building apps needs additional steps such a CI

Some famous tools

- Yocto/OpenEmbedded
  - Written in Python (BitBake)
  - Very powerful but needs training
  - Mostly text mode (poor GUI)
- Buildroot
  - Based on standard GNU-Make
  - Was a tool for uClibc developers
  - Independent project since 2009
  - GUI for configuration but no packages
- OpenWrt
  - Close to Buildroot
  - Handle binary packages
- PTXdist

OpenEmbedded

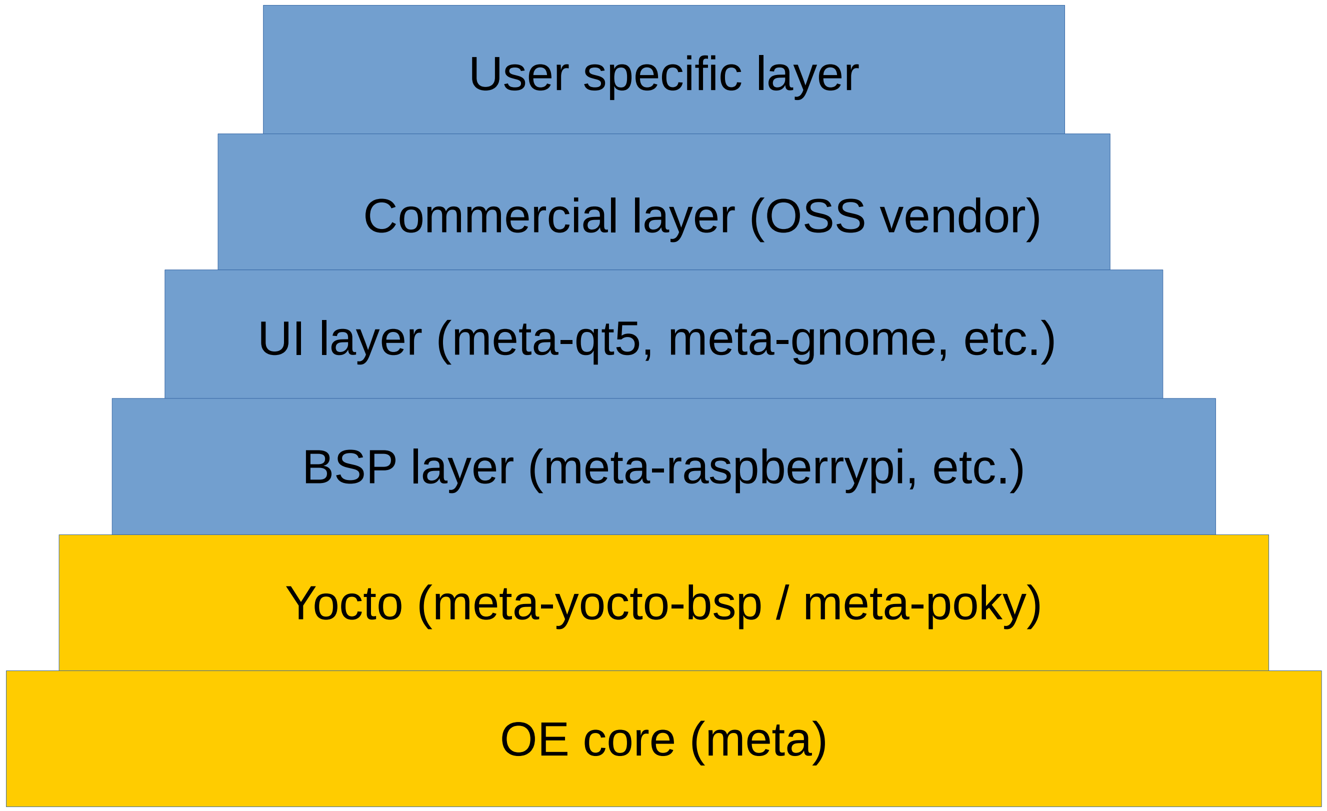
- OE is a “cross-compilation framework”
- Started in 2003 by Chris Larson, Michael Lauer and Holger Schuring for OpenZaurus to replace Buildroot
- The Zaurus (SHARP) is the “first” PDA running Linux OS in 2001

Yocto/OE main concepts

- Metadata
  - Recipes (`.bb`)
  - Extended recipes (`.bbappend`)
  - Configuration (`.conf`)
  - Classes (`.class`)
  - Includes (`.inc`)
- Layer is Metadata container
- Recipe creates one (or several) binary package(s)

```
$ bitbake my-recipe
```

Yocto layers principle


User specific layer
Commercial layer (OSS vendor)
UI layer (meta-qt5, meta-gnome, etc.)
BSP layer (meta-raspberrypi, etc.)
Yocto (meta-yocto-bsp / meta-poky)
OE core (meta)

🟦 External project

🟨 Yocto project

Yocto for RPi “in a nutshell”

- Install Poky (Yocto reference distro)

```
$ git clone -b <branch> git://git.yoctoproject.org/poky
```

- Get the HW layer (meta-raspberrypi)

```
$ cd poky
$ git clone -b <branch> git://git.yoctoproject.org/meta-raspberrypi
```

- Create the build directory

```
$ source oe-init-build-env rpi-build
```

- Add RPI layer to `conf/bblayers.conf`

```
$ bitbake-layers add-layer ../meta-raspberrypi
```

- Specify the target device in `conf/local.conf`

```
echo “MACHINE = \"raspberrypi\"" >> conf/local.conf
```

- Create the image

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

- Copy the image to SD card

```
$ sudo dd if=<path>/core-image-minimal-raspberrypi.rpi-sdimg of=/dev/mmcblk0
```

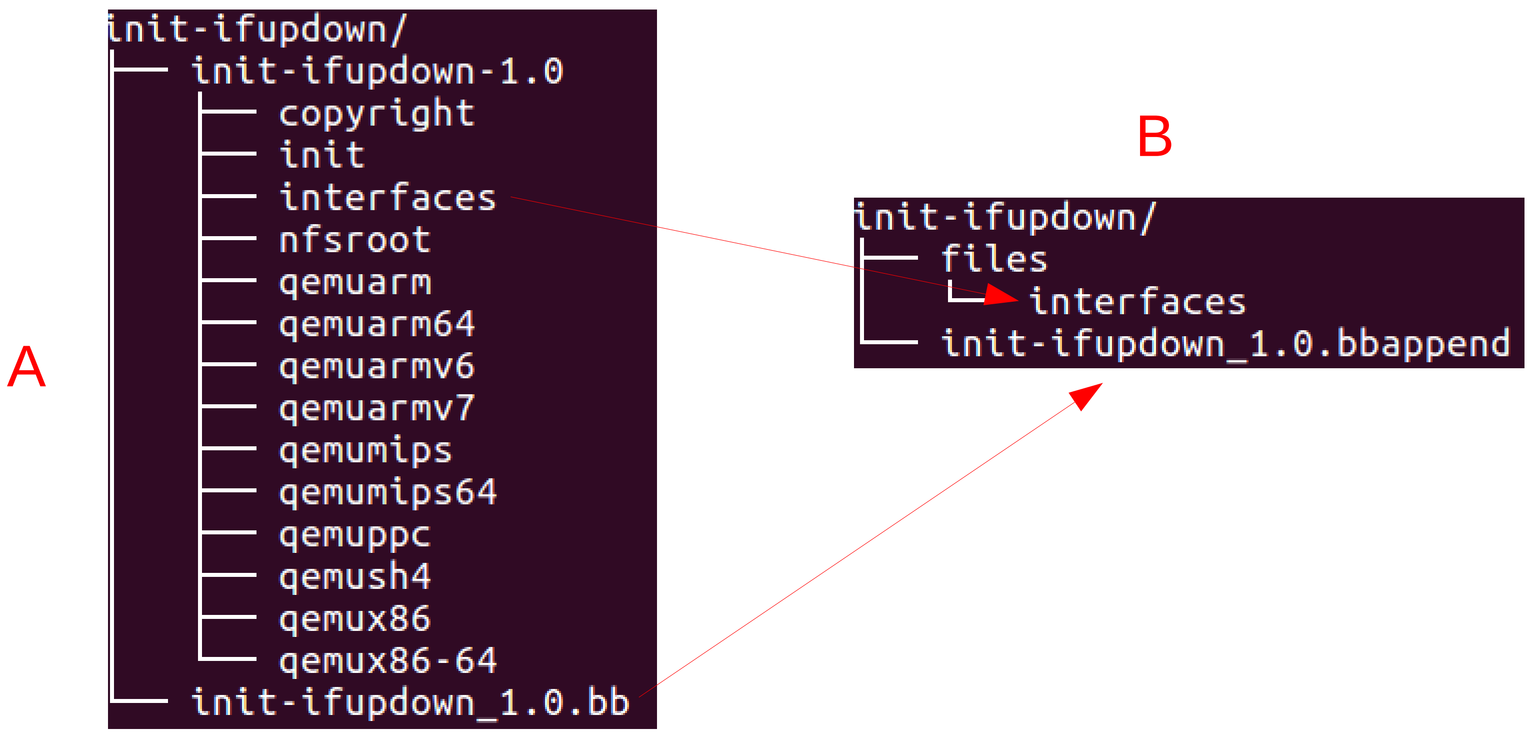
Using recipes

Recipe for CMake based example

```
DESCRIPTION = "Helloworld software (CMake)"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM =
"file://COPYING;md5=8ca43cbc842c2336e835926c2166c28b"

SRC_URI = "http://pficheux.free.fr/pub/tmp/mypack-cmake-1.0.tar.gz"

inherit cmake

SRC_URI[md5sum] = "70e89c6e3bff196b4634aeb5870ddb61"
```

Using bbappend

- Recipe (.bb) defined in layer “A”
- Recipe is “appended” with .bbappend in layer “B”



```
init-ifupdown/
├── init-ifupdown-1.0
│   ├── copyright
│   ├── init
│   ├── interfaces
│   ├── nfsroot
│   ├── qemuarm
│   ├── qemuarm64
│   ├── qemuarmv6
│   ├── qemuarmv7
│   ├── qemumips
│   ├── qemumips64
│   ├── qemuppc
│   ├── qemush4
│   ├── qemux86
│   └── qemux86-64
└── init-ifupdown_1.0.bb
```

```
init-ifupdown/
├── files
│   └── interfaces
└── init-ifupdown_1.0.bbappend
```

- Adding / updating data files
- Adding BitBake functions (prepend/append)
- Patching sources
- Auto-loading kernel modules

Devshell

- One can modify sources with Devshell

```
$ bitbake -c devshell <recipe>
```

- Open an new terminal where you can use standard development tools (`cmake`, `make`), instead of `bitbake`
- Very useful to test a quick modification !
- Use `devtool` for a better/simpler approach

Devtool

- Devtool is dedicated to add / modify / upgrade recipes and associated source code
- Three main functionalities :
  - Creating a recipe from source code (add)
  - Modifying an existing recipe (modify)
  - Upgrading version for an existing recipe (upgrade)
- Typical syntax

  ```
  $ devtool <command> <parameters>
  ```
- Very useful to create a patch in a `.bbappend`
- Does NOT replace Yocto experience for complex recipes !

Devtool

- Devtool uses a temporary “workspace”
- Default is `workspace` in the “build” directory
- You can create an external one with `create-workspace` command
- Workspace directory added to `bblayer.conf`
- Source in `workspace/sources` managed by Git
- Created/modified recipe should be copied to a real layer (with `finish` command)

Basic Devtool session

- Start modifying existing recipe

```
$ devtool modify <recipe>
```

- Update source code

```
$ cd <workspace>/sources/<recipe>
$ vi file.c
$ git commit -a -m “updated code”
```

- Build recipe package (optional)

```
$ devtool build <recipe>
```

- Finally copy patched recipe (.bbappend) to a layer

```
$ devtool finish <recipe> <layer-path>
```

eSDK

eSDK

- Extensible SDK generated by Yocto
- “Internal” Yocto compiler is not usable without BitBake
- A set of application development tools
  - Cross compiler
  - Cross debugger (`gdb` / `gdbserver`)
  - Eclipse plugin
  - QEMU emulator (x86, ARM)
  - etc.
- Yocto knowledge is not necessary
- Documented in the “Application Development and the Extensible Software Development Kit (eSDK)” manual

SDK installation and testing

- Most of time, toolchain is produced by Yocto (except using `EXTERNAL_TOOLCHAIN` variable)
- The following command creates a basic toolchain as an installation script

```
$ bitbake meta-toolchain
```

- One can install the toolchain by the following :

```
$ sudo tmp/deploy/sdk/poky-glibc-x86_64-meta-toolchain-arm1176jzfshf-vfp-<machine>-toolchain-<version>.sh
```

- Use of the toolchain as follows :

```
$ . /opt/poky/<version>/environment-setup-arm1176jzfshf-vfp-poky-linux-gnueabi

$ $CC -o hello hello.c
```

Extended toolchain

- Some images include components required at build time (added libraries, tools, etc.)
- The "populate_sdk" task creates a toolchain including all specific stuff

```
$ bitbake -c populate_sdk my-image
```

- Add the following line to `local.conf` to include kernel headers (if you want to compile modules)

```
TOOLCHAIN_TARGET_TASK_append = " kernel-devsrc"
```

- Kernel part should be configured (as root !) before use

```
# . /opt/poky/<version>/environment-setup-<arch>
# cd /opt/poky/<version>/sysroots/<arch>/usr/src/kernel
# make oldconfig
# make scripts
```

- Compiling a module

```
$ make KERNEL_SRC=/opt/poky/<version>/sysroots/<arch>/usr/src/kernel
```

Building kernel module inside Yocto

- Source tree is located at `tmp/work-shared/raspberrypi`

```
$ ls -1 tmp/work-shared/raspberrypi/
kernel-build-artifacts ◄—— to be used in KERNEL_SRC variable in Makefile
kernel-source
```

- Build a module with

```
$ make KERNEL_SRC=<path>/kernel-build-artifacts
```

- Better way with Yocto eSDK !

CI

What is CI ?

- Modification (upgrade) should not add “regression”
  - Standard components (OS)
  - Added applications (developed with SDK)
- Methods and tools
  - SCM (Source Control Management)
  - Unit / functional test (per package)
  - Global test (Yocto image)
  - Emulation + test automation
  - Jenkins, LAVA, SonarQube, QEMU, etc.
- Yocto provides “ptest” (package test) and “testimage” (image test)

Testing package (ptest)

- Recipe should include inherit “ptest” class
- Recipe should include a `run-ptest` script
- Options to add to image (`local.conf`)

```
DISTRO_FEATURES_append = " ptest"
EXTRA_IMAGE_FEATURES += "ptest-pkgs"
```

- Image should now include `/usr/lib/*/ptest`
- List available tests

```
# ptest-runner -l
```

- Test is started with

```
# ptest-runner <pkg-name>
```

- All tests started if no parameter
- Use SSH for automatic testing

```
$ ssh root@<target-IP> ptest-runner
```

- Several recipes use ptest (BusyBox, BlueZ, etc.)

Global testing (testimage)

- Image configuration (`local.conf`)

```
INHERIT += "testimage"
TEST_SUITES = " ping ssh"
# For real board (not QEMU)
TEST_TARGET = "simpleremote"
TEST_SERVER_IP = "192.168.3.1"
TEST_TARGET_IP = "192.168.3.141"
```

- Build, install & boot the new image for the target
- Test from PC

```
$ bitbake -c testimage core-image-minimal
RESULTS:
RESULTS - ping.PingTest.test_ping - Testcase 964: PASSED
SUMMARY:
core-image-minimal () - Ran 1 test in 0.032s
core-image-minimal - OK - All required tests passed
```

Customizing testimage

- Tests in `meta/lib/oeqa/runtime/cases`
- Add new tests to `<meta-layer>/lib/oeqa/runtime/cases`

Question ?