Yocto

Yasir NL

About yocto

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. It was founded in 2010 as a collaboration among many hardware manufacturers, open-source operating systems vendors, and electronics companies to bring some order to the chaos of embedded Linux development.

Why use the Yocto Project? It's a complete embedded Linux development environment with tools, metadata, and documentation - everything you need. The free tools are easy to get started with, powerful to work with (including emulation environments, debuggers, an Application Toolkit Generator, etc.) and they allow projects to be carried forward over time without causing you to lose optimizations and investments made during the project's prototype phase. The Yocto Project fosters community adoption of this open source technology allowing its users to focus on their specific product features and development.

Yocto is Raspid, repeatable develop[ment of linux - based embedded systems

Platform Yocto supports

1) Raspberry pi
2) Beaglebone black
3) Digi Connect Core
4) Snickerdoodle(Zyng) etc.

Other Supported platforms

- All major intel chips(Atom , i7,etc)
- AMD
- Graphics Cards are hit and miss
- Most popular target hardware has all hardware of the board support

Board Support Packages

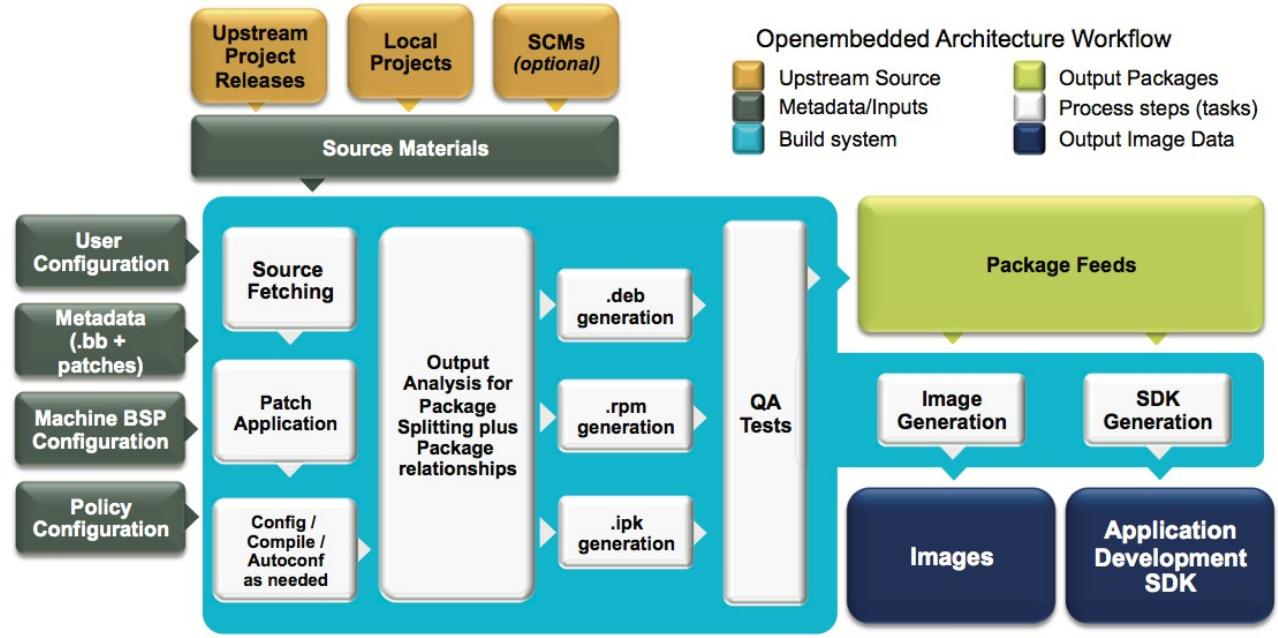
#MACHINE ?= "beaglebone"
#MACHINE ?= "genericx86"
#MACHINE ?= "genericx86-64"
#MACHINE ?= "mpc8315e-rdb"
#MACHINE ?= "edgerouter"
#MACHINE = "raspberrypi3"
#MACHINE = "raspberrypi2"
#MACHINE = "raspberrypi1"

Yocto workflow

Yocto workflow based on OpenEmbedded. User Configuration, Metadata, Hardware Configuration, and Policy Management determine what Upstream Projects, Local Projects (stored on the developer's system) and what sources from optional Source Change Management systems to include, as well as control the individual process steps of the workflow. A Yocto Project release already contains configuration information and metadata that will produce a working Linux system image only requiring very minimal adaption to the developer's local build environment.

While below figure shows a rather linear process consisting of the steps Source Fetching,

Patch Application, Configure/Compile, Output Analysis for Packaging, Package Creation and QA Tests, these steps are in fact repeated for each source package and therefore potentially many thousand times before all PackageFeeds have been created and can be combined into an image. Therefore Yocto supports multi-processor and multi-core build systems by automatically calculating dependencies and executing process steps in parallel, hence greatly accelerating the entire build process. Of course, Yocto also manages changes and only rebuilds Package Feeds whose input, source, metadata, dependencies, etc. have changed.



Let's look at what the components in the the diagram stand for:

User Configuration : This is metadata you can use to control the build process.

Metadata layer : These are various layers that provide sostware, machine, and distribution metadata.

Source Filres : These contain upstream release, local project, and source control manegment (Git, SVN, and so on).

Build syatem : These are processes under the control of Bitbake. This block expands on how Bitbake fetches source files, applies patches, complets, compilation, analyze output for package generation, creates and tests the packages, generates images, and generates cross-development tools.

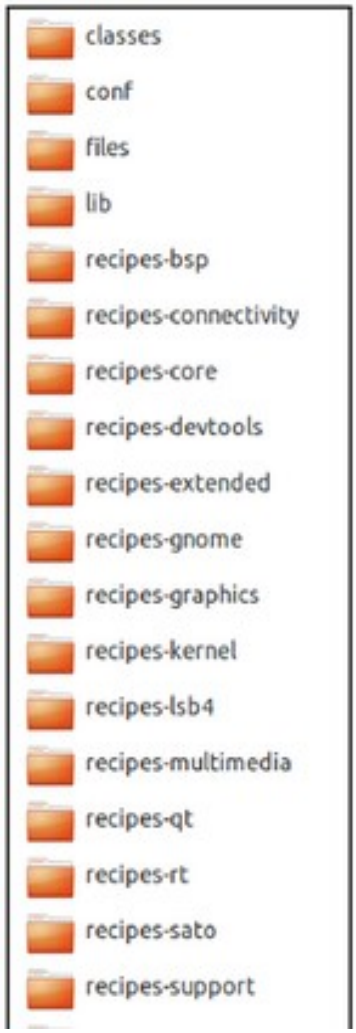
Package feeds : these are directories containing output packages(RPM, DEB, or IPK) which are subsequently used in the construction of an image or SDK produced by the build system. These feeds can also be copied and shared using a web server or other means to facilitate extending existing images on device at runtime package management is enebled.

Image: these are the images produced by development process(the pieces that compose the OS ,such as rootfs, uboot, kernel).

Application Development SDK : These are cross-development tolls that are produced along with an image or separately with bitbake.

BitBake : Bitbake is a make-like build tool with the special focus of distributions and packages for embedded Linux cross compilation although it is not limited to that. It is inspired by Portage, which is the package management system used by the Gentoo Linux distribution.

Yocto Meta-raspberrypi(This contains recipe)



The meta data contains the information about

1. Software reciepes
2. The versions of the kernel
3. Where to get software for a particular package and kernel
4. How to build kernel, Busy box(Root File System), BSP, etc.
5. How to build packages
6. How to boot the images
7. What machine hardware you are targeting

Poky

Poky is a *reference distribution* of the Yocto Project. It contains the OpenEmbedded Build System (BitBake and OpenEmbedded Core) as well as a set of metadata to get you started building your own distro. To use the Yocto Project tools, you can download Poky and use it to bootstrap your own distribution. Note that Poky does not contain binary files - it is a working example of how to build your own custom Linux distribution from source.

Bitbake

1. Is a task runner , task scheduler for Yocto
2. consumes meta-data(.bb files)
3. Outputs bootable media for the target hardware
4. Has analytics
5. Enforces licensing rules
6. Many sanity checks and buils stare checks
7. Multi-core
8. Can also produce SDK for atrget

Yocto Build steps for raspberry pi 3

1. Create directory structure to download source

```
mkdir -p ~/rpi/sources
```

2. cd into directory

```
cd ~/rpi/sources
```

3. Get the required layers

We will need bare minimum above 3 clones for building Linux for Raspberry Pi 3
– poky
– meta-openembedded
– meta-raspberrypi

git clone -b krogoth git://git.yoctoproject.org/poky

git clone -b krogoth git://git.openembedded.org/meta-openembedded

git clone -b krogoth git://git.yoctoproject.org/meta-raspberrypi

4.Preparing the build configuration

Following command will create a build environment using the setup script that comes with poky, and will create a “rpi-build” directory in “~/rpi/” directory

```
cd ~/rpi/

source sources/poky/oe-init-build-env rpi-build
```

Now you will be in “~/rpi/rpi-build” directory

5. Add some configuration lines to Config files

Add following lines to change the build process for us

a) Changes in local.conf

```
echo 'MACHINE = "raspberrypi3"' >> conf/local.conf #taget machine
```

```
echo 'PREFERRED_VERSION_linux-raspberrypi = "4.%"' >> conf/local.conf
#kernel version

echo 'DISTRO_FEATURES_remove = "x11 wayland"' >> conf/local.conf

echo 'DISTRO_FEATURES_append = " systemd"' >> conf/local.conf

echo 'VIRTUAL-RUNTIME_init_manager = "systemd"' >> conf/local.conf

# Default to setting automatically based on cpu count

echo PARALLEL_MAKE ?= " -j ${@oe.utils.cpu_count()}" >> conf/local.conf

# option determines how many tasks bitbake should run in parallel:

echo BB_NUMBER_THREADS  ?= " ${@oe.utils.cpu_count()}" >> conf/local.conf
```

b) Changes the bblayers.conf file to following

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BSPDIR := "/home/yasir/rpi/"

BBLAYERS ?= " \
 ${BSPDIR}/sources/poky/meta \
 ${BSPDIR}/sources/poky/meta-poky \
 ${BSPDIR}/sources/poky/meta-yocto-bsp \
 ${BSPDIR}/sources/meta-openembedded/meta-oe \
 ${BSPDIR}/sources/meta-openembedded/meta-multimedia \
 ${BSPDIR}/sources/meta-raspberrypi \
 "
BBLAYERS_NON_REMOVABLE ?= " \
 ${BSPDIR}/sources/poky/meta \
 ${BSPDIR}/sources/poky/meta-poky \
 "
```

6. Menuconfig changes for Kernel/Busybox

```
bitbake linux-raspberrypi -c menuconfig

bitbake -c menuconfig busybox
```

Note : These commands leads to UI configuration menuconfig, select and save the config required(same as the normal kernel configuration)

7. Only if you want to build specific receipe

Bitbake -c <receipe name>

Ex : Only kernel build

Bitbake -c linux-raspberrypi

Note : If you want add extra packages like alsa,gstreamer,apt etc

This can be configured in local.conf file

IMAGE_INSTALL_append

or

CORE_IMAGE_EXTRA_INSTALL

Specifies the list of packages to be added to the image. You should only set this variable in the local.conf configuration file found in the Build Directory.

Ex: CORE_IMAGE_EXTRA_INSTALL += "apt"

Even You can add tools in below given file

meta-raspberrypi/recipe-core/images/rpi-hwup-image.bb

ex:

*IMAGE_INSTALL += " *

*kernel-modules *

***git ***

“

Note : *You can add any number of tools based on your requirment*

8. complete os Finally Build Image

bitbake rpi-basic-image

(this minimal image with single core)

OR

bitbake rpi-hwup-image

(minimal image with multicore, for raspberry pi 3 4 cores, with some more feature than rpi-basic-image)

OR

bitbake rpi-test-image

This will take few hours depending upon the Host Machine configuration. If above command is successful and runs without errors you will find the image in the particular folder.

This image takes quite a while to build - expect over a day if you are building inside a virtual machine. It is also very large - not the image itself, but the build components take up almost 50GB, so make sure you have enough space on your computer. The actual image that will be booted onto your embedded device will be very small.

9. Flashing image to SD Card

sudo dd if=~/rpi/rpi-build/tmp/deploy/images/raspberrypi3/

rpi-basic-image-raspberrypi3.rpi-sdimg

of=/dev/sdX bs=4M

10. Boot Raspberry Pi 3

Insert the SD Card into the Raspberry Pi 3 and boot it up, **login with *root* and no password**

busybox configuration and building

01. Execute the command menuconfig on the package busybox. This allows customization of busybox applets

02. Run all the tasks for package busybox

03. Create the final image, in this case the frame buffer image, see tmp/deploy/images for outputs.

Cloning the Kernel fron local Repositary or other Repositary

Use below command to change kernel recipe

vi rpi/sourses/meta-raspberrypi/recipe-kernel/linux/x_x_kernel-version.bb

Change the SRCREV(with recent commit) and SRC_URI

Recent commit can be found by using below git command

```
git rev-parse HEAD or git rev-parse --verify HEAD
```

Change SRC_URL with your URL from where you want kernel source

Ex : SRC_URI = "git://gitolite_server@192.168.2.197:yocto.git;protocol=file;branch=master;bareclone=1 \ “

bareclone --> cloning localy

branch -->which branch you wnat to clone

Note : *If you have patchs those u can apply*

first copy pacths in this folder *meta-raspberrypi/recip-kernel/linux/linux-raspberrypi-4.1*

then write packs name is SRC_URI

EX :

SRC_URI =
"git://gitolite_server@192.168.2.197:yocto.git;protocol=file;branch=master;bareclone=1 \

*file://0001-arm64-enabled-sdhost-support-to-drive-sdcard-into-BC.patch; *

*file://0001-arm64-bcm2837-Alsa-HDMI-Audio-Enabled.patch *

*file://0001-BLUETOOTH_ENABLE.patch *

"

Note : make sure all pacths are in order

Note : This only for changing kernel recipe, If you want to change other recipe procedure is same only path will change.

Bitbake usefull commands

bitbake <image> (build an image)
bitbake <package> -c <task> (run a single task for a single recipe)
bitbake <package> -c devshell (enter a shell for a single receipe)
bitbake <package> -c listtasks (lists the tasks defined in a .bb recipe)
bitbake <kernel-recipe-name> -c menuconfig(kernel menuconfig launch)
bitbake-layers show-layers(lists layers found based on bblayers.conf file)
bitbake-layers show-recipes"*-image-*(lists recipes for a single image)
bitbake -e <package> (print environment vars for a recipe)
bitbake -c clean <package>(example of how to run a single build step)
bitbake -v <package> (verbose output)
bitbake <image> -g -u depexp (show a awesome dependency viewer)
bitbake <image> -c populate_sdk(compile the SDK installer)

Bitbake packages compilation process(These steps will happen automatically)

Compilation of Any package using bitbake goes through stages as mentioned in below image,

1. do_fetch – This downloads the source code of packages from remote location, use below command to fetch package source,

$ bitbake -c fetch package_name

2. do_unpack –This extracts the source code of tar/zip, use below command to unpack package source,

$ bitbake -c unpackpackage_name

3. do_patch–This patches the local changes from metadata to the source code, use below command to patch package source,

$ bitbake -c patch package_name

4. do_configure – This configures the source code for compilation, mostly it runs configure script to create a Makefile, use below command to configure package source,

$ bitbake -c configure package_name

5. do_compile – This compiles the source code to create libraries and executables, use below command to compile package source,

$ bitbake -c compile package_name

6. do_stage –

$ bitbake -c stage package_name

7. do_install – This installs the executable, libraries into temp source location, use below command to install package source,

$ bitbake -c install package_name

8. do_package – This creates a .rpm, .deb etc which can be used to later install as standalone package into already existing OS, use below command to package,

$ bitbake -c package package_name

References

1) https://github.com/Nuand/bladeRF/wiki/Creating-Linux-based-Embedded-System-Images-with-Yocto

2) Yocto for Raspberry Pi - Pierre-Jean Texier, Petter Mabacker - Google Books.html