**YOCTO PROJECT**

* Yocto Project isnt an embedded linux distro, it creates one for you
* Why to use yocto

1. To configure the linux according to our specifications

2. The image will contain what we need

3. No extra packages

4. Small image size

* Why not use ubuntu instead

1. Lightweight and Tailored System

Embedded Systems: They often have limited resources (e.g., CPU, RAM, storage). Ubuntu and similar distros are general-purpose operating systems with lots of pre-installed packages and services that aren't needed in embedded systems.

Yocto: Allows you to build a custom Linux distribution tailored to your hardware and application requirements. You can strip out unnecessary components and include only what you need, making it highly optimized and lightweight.

2. Customization and Control

Ubuntu: Comes with pre-built binaries and a fixed set of features. Customizing it to match specific hardware or to optimize performance is difficult.

Yocto: Provides fine-grained control over every aspect of the system (kernel, libraries, toolchains, etc.). You can modify or create your own recipes for custom features, ensuring the system matches your exact requirements.

3. Hardware Support

Embedded Systems: Usually involve custom or unique hardware, like microcontrollers, ARM processors, or custom boards. These devices often require:

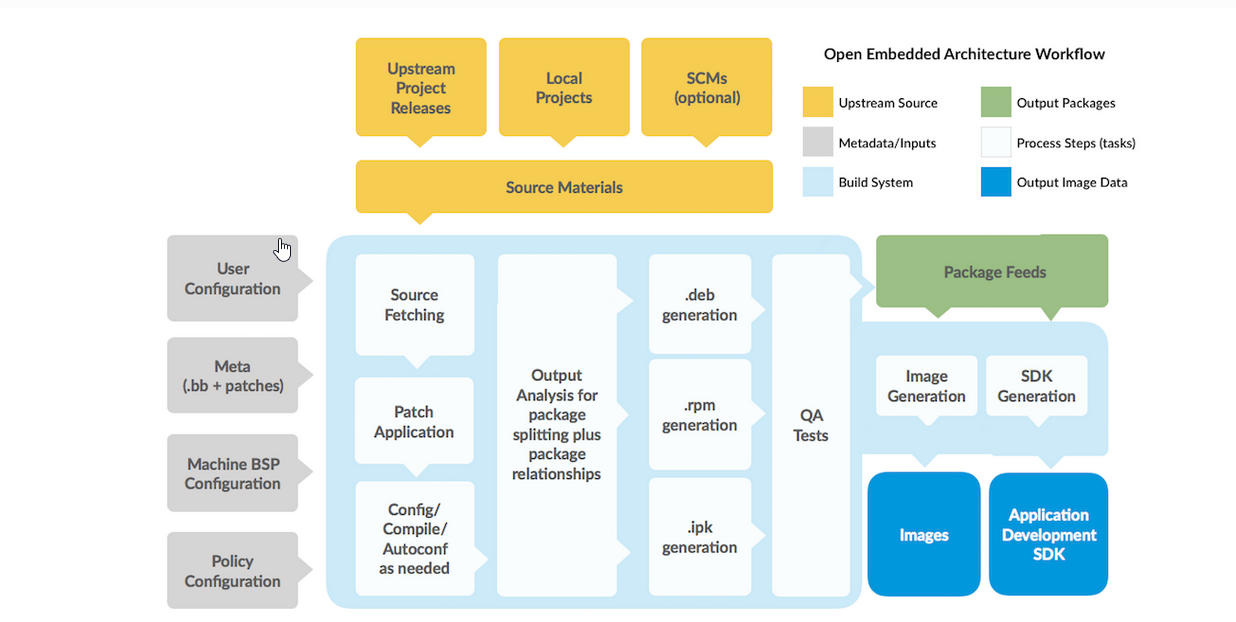
Custom drivers

Board Support Packages (BSPs)

Minimal kernel configurations

Yocto: Supports custom BSP layers and allows you to configure the Linux kernel, drivers, and firmware specifically for your hardware. This level of customization is essential for embedded systems.

* General Workflow of how yocto works



Here's a simple explanation of the Yocto workflow based on the diagram:

Source Materials (Yellow)

Yocto gets its sources from Upstream Project Releases, Local Projects, or Source Code Management (SCM) systems like Git.

These provide the raw materials for building the embedded Linux system.

Metadata & Configuration (Gray)

User Configuration: Defines what should be built (e.g., target hardware, features).

Meta (.bb + patches): Recipes (.bb files) that describe how to fetch, build, and package software.

Machine BSP Configuration: Specifies board-specific details like kernel, bootloader, and drivers.

Policy Configuration: Defines packaging and licensing rules.

Build Process (Blue - Yocto Build System)

Source Fetching: Downloads required sources from repositories.

Patch Application: Applies patches if needed.

Config/Compile/Autoconf: Configures, compiles, and builds the packages.

Output Analysis: Splits packages and determines dependencies.

Package Generation: Creates .deb, .rpm, or .ipk package formats.

QA Tests: Ensures the system is built correctly.

Output Generation (Blue & Green)

Package Feeds: Stores generated packages for installation or updates.

Image Generation: Creates the final bootable embedded Linux image.

SDK Generation: Builds an SDK for developing applications.

Final Output (Blue)

Images: The actual embedded Linux system that runs on hardware.

Application Development SDK: A software development kit for building applications.

Supported Linux Distributions for Yocto

Yocto primarily requires a Linux-based build system, and while it can be made to work on various distributions, some are officially supported and tested.

✅ Recommended and Officially Supported Distros

Yocto officially supports these distributions for building embedded Linux images:

Ubuntu (LTS versions preferred)

Ubuntu 20.04 LTS

Ubuntu 22.04 LTS

Debian (Stable releases)

Debian 10 (Buster)

Debian 11 (Bullseye)

Fedora

Fedora 36+

CentOS / Rocky Linux / AlmaLinux

CentOS 8+ (deprecated but works)

Rocky Linux / AlmaLinux 8+

OpenSUSE

OpenSUSE Leap 15+

**Compatible Linux Distribution**[**ℑ**](https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html#compatible-linux-distribution)

Make sure your [Build Host](https://docs.yoctoproject.org/ref-manual/terms.html#term-Build-Host) meets the following requirements:

* At least 90 Gbytes of free disk space, though much more will help to run multiple builds and increase performance by reusing build artifacts.
* At least 8 Gbytes of RAM, though a modern modern build host with as much RAM and as many CPU cores as possible is strongly recommended to maximize build performance.
* Runs a supported Linux distribution (i.e. recent releases of Fedora, openSUSE, CentOS, Debian, or Ubuntu). For a list of Linux distributions that support the Yocto Project, see the [Supported Linux Distributions](https://docs.yoctoproject.org/ref-manual/system-requirements.html#supported-linux-distributions) section in the Yocto Project Reference Manual. For detailed information on preparing your build host, see the [Preparing the Build Host](https://docs.yoctoproject.org/dev-manual/start.html#preparing-the-build-host) section in the Yocto Project Development Tasks Manual.
  + Git 1.8.3.1 or greater
  + tar 1.28 or greater
  + Python 3.8.0 or greater.
  + gcc 8.0 or greater.
  + GNU make 4.0 or greater

If your build host does not meet any of these three listed version requirements, you can take steps to prepare the system so that you can still use the Yocto Project. See the [Required Git, tar, Python, make and gcc Versions](https://docs.yoctoproject.org/ref-manual/system-requirements.html#required-git-tar-python-make-and-gcc-versions) section in the Yocto Project Reference Manual for information.

What is Poky in Yocto?

Poky is the reference distribution of the Yocto Project. It serves as the default build system and provides the foundation for building custom embedded Linux distributions.

Key Features of Poky:

Build System (BitBake + Metadata)

Poky contains BitBake, the tool responsible for processing recipes and building packages.

It also includes Yocto metadata, which defines how to fetch, compile, and package software.

Reference Distribution

Poky is not an actual Linux distribution like Ubuntu. Instead, it provides a template for building custom embedded Linux systems.

Layers & Recipes

Poky includes meta-yocto (Yocto's core metadata layer) and meta-yocto-bsp (Board Support Package for common hardware platforms).

Developers can add custom layers (meta-<yourproject>) to extend functionality.

Toolchain & SDK

Poky provides a cross-compilation toolchain for building software for embedded targets.

It includes application development SDKs for developing applications.

Minimal & Customizable

Unlike Ubuntu or Debian, Poky does not come with unnecessary software.

It allows developers to customize the OS specifically for their embedded system.

Poky Structure

When you clone Poky, you get:

poky/

├── bitbake/ # BitBake build tool

├── meta/ # Core Yocto metadata (recipes, classes, etc.)

├── meta-yocto/ # Yocto Project metadata

├── meta-yocto-bsp/ # BSPs for standard hardware (QEMU, etc.)

├── scripts/ # Helper scripts for Yocto

└── oe-init-build-env # Initialization script

How Poky Fits into Yocto

Poky = BitBake + Metadata

It is the default environment used to build an embedded Linux image in Yocto.

Developers often start with Poky and then add custom layers to modify the build.

How to Use Poky

Clone Poky

git clone https://git.yoctoproject.org/poky

Initialize the Build Environment

cd poky

source oe-init-build-env

Build a Basic Image

bitbake core-image-minimal

Summary

Poky is the reference build system for Yocto.

It contains BitBake, metadata, and tools to build embedded Linux distributions.

It is highly customizable and serves as the foundation for custom embedded OS builds.

Start working on Yocto Project  
  
sudo apt install build-essential chrpath cpio debianutils diffstat file gawk gcc git iputils-ping libacl1 liblz4-tool locales python3 python3-git python3-jinja2 python3-pexpect python3-pip python3-subunit socat texinfo unzip wget xz-utils zstd

git clone https://github.com/yoctoproject/poky.git

logs

hza1kor@KOR-V-001FY:~/Yocto\_Project\_BeagleBone$ git clone https://github.com/yoctoproject/poky.git

Cloning into 'poky'...

remote: Enumerating objects: 673109, done.

remote: Counting objects: 100% (1602/1602), done.

remote: Compressing objects: 100% (368/368), done.

remote: Total 673109 (delta 1346), reused 1287 (delta 1234), pack-reused 671507 (from 4)

Receiving objects: 100% (673109/673109), 228.46 MiB | 8.41 MiB/s, done.

Resolving deltas: 100% (499134/499134), done.

Updating files: 100% (5730/5730), done.

hza1kor@KOR-V-001FY:~/Yocto\_Project\_BeagleBone$ cd poky/

hza1kor@KOR-V-001FY:~/Yocto\_Project\_BeagleBone/poky$ git checkout kirkstone

hza1kor@KOR-V-001FY:~/Yocto\_Project\_BeagleBone/poky$ source oe-init-build-env

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:

core-image-minimal

core-image-full-cmdline

core-image-sato

core-image-weston

meta-toolchain

meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

Other commonly useful commands are:

- 'devtool' and 'recipetool' handle common recipe tasks

- 'bitbake-layers' handles common layer tasks

- 'oe-pkgdata-util' handles common target package tasks

Understanding the Command:

source oe-init-build-env

This command initializes the Yocto build environment when working with Poky. Let's break it down:

1. What is source?

source is a built-in shell command in Linux.

It runs a script in the current shell session instead of launching a new shell.

This means any environment variables set by the script remain active in your terminal session.

Example:

source myscript.sh

This runs myscript.sh in the same shell, so any environment changes persist.

2. What is oe-init-build-env?

This is a script inside the Poky directory (poky/oe-init-build-env).

It sets up the environment variables needed to build an embedded Linux image.

It creates a build directory (build/ by default) where Yocto stores configurations and output files.

3. What Happens When You Run It?

source oe-init-build-env

✅ 1. Creates the build/ directory (if it doesn’t exist)

By default, it creates build/ inside your Poky folder.

✅ 2. Sets Environment Variables

Updates PATH, BBPATH, and other variables needed by BitBake.

Defines the location of bitbake, meta layers, and configuration files.

✅ 3. Moves You into the build/ Directory

Your terminal automatically changes to the build/ directory.

4. Example Output

When you run:

source oe-init-build-env

You’ll see something like this:

### Shell environment set up for building with BitBake ###

You can now run 'bitbake <target>'

Common targets are:

core-image-minimal

core-image-sato

meta-toolchain

meta-ide-support

This means you are ready to run BitBake and start building images!

5. Customizing the Build Directory

By default, it creates poky/build/, but you can specify a custom directory:

source oe-init-build-env my-custom-build

This creates poky/my-custom-build/ instead of poky/build/.

6. What to Do After Running This Command?

Once the environment is set up, you can start building:

bitbake core-image-minimal

This will compile the embedded Linux image.

Summary

🔹 source oe-init-build-env sets up the Yocto build environment

🔹 Creates a build directory (default: build/)

🔹 Sets important environment variables for Yocto and BitBake

🔹 Moves you into the build directory

🔹 Allows you to run BitBake to build your embedded Linux system

hza1kor@KOR-V-001FY:~/Yocto\_Project\_BeagleBone/poky/build$ bitbake core-image-minimal

Understanding the Command:

bitbake core-image-minimal

This command builds a minimal Linux image using BitBake in the Yocto build system.

Breakdown of the Command

1. bitbake

BitBake is the build tool used by Yocto.

It processes recipes (.bb files) to fetch, compile, and package software.

Similar to how make compiles software from source, BitBake builds entire Linux distributions.

2. core-image-minimal

This is a target recipe that defines a basic embedded Linux image.

It contains:

A minimal root filesystem (no GUI).

A basic Linux kernel.

Essential system utilities and libraries.

What Happens When You Run It?

BitBake Parses the Recipe (core-image-minimal.bb)

It checks the dependencies and required packages.

Fetches Source Code

Downloads required sources (kernel, utilities, libraries) from the internet or local mirrors.

Applies Patches

If there are patches in Yocto metadata layers, they are applied.

Compiles the Software

Uses the cross-compilation toolchain to build the kernel, bootloader, and user-space applications.

Generates the Root Filesystem (rootfs)

Creates a minimal Linux filesystem with essential tools.

Builds the Final Image

Creates a .wic, .ext4, or other bootable image file.

Where Does the Image Go?

After a successful build, you can find the output image in:

build/tmp/deploy/images/<machine>/core-image-minimal-<machine>.wic

For example, if you are building for Raspberry Pi 4, the final image might be:

build/tmp/deploy/images/raspberrypi4/core-image-minimal-raspberrypi4.wic

How Long Does It Take?

First Build: Several hours (downloads + compilation).

Subsequent Builds: Much faster (cached components).

How to Use the Image?

Flash it to an SD card or eMMC:

sudo dd if=core-image-minimal-raspberrypi4.wic of=/dev/sdX bs=4M status=progress

Boot it on your embedded board.

Summary

bitbake core-image-minimal builds a minimal embedded Linux OS.

It fetches, compiles, and packages the OS.

Produces a bootable image in deploy/images/.

Ideal for embedded devices with limited resources.