Poky-zeus-bbb repository

1️⃣ bitbake

What it is:

- BitBake is the core build engine of Yocto. Think of it like make for embedded Linux, but much smarter.
- It reads recipes (.bb files) and classes (.bbclass), resolves dependencies, and executes tasks to produce images or packages.

How it works:

1. Parse metadata: Reads all recipes, layer configurations, and machine configs.
2. Dependency resolution: Figures out which packages need to be built first.
3. Task execution: Executes tasks like do_fetch, do_configure, do_compile, do_install, do_package.
4. Sstate cache: Reuses previously built outputs to avoid recompiling.

Practical workflow:

source oe-init-build-env build

bitbake core-image-minimal

- BitBake will fetch sources, build kernel, libraries, and packages, and generate a bootable image.

2️⃣ meta

What it is:

- Core layer of Yocto. Provides base recipes, classes, and configuration.
- Essential building blocks for all images.

Contents:

- Recipes for basic packages: busybox, glibc, bash, openssl, etc.
- Classes: .bbclass files with reusable build logic.
- Configuration: defaults for compiler, sysroot, file system layout.

How it works:

- Other layers depend on meta for core functionality.
- Recipes here are minimal but essential for Linux to run.

Example:

- busybox_%.bb → builds busybox binary (shell, core commands)

- kernel.bbclass → provides generic kernel build logic

---

3 meta-poky

What it is:

- Reference layer built on top of meta.
- Contains default images, sample configurations, and reference distro settings.

Contents:

- core-image-minimal.bb
- core-image-sato.bb
- Reference distro.conf and machine.conf
- Provides a working starting point for your own layers

How it works:

- When you run bitbake core-image-minimal, recipes here define what packages go into the image.
- You can override these in your own layers if needed.

---

4 meta-openembedded

What it is:

- Community layer collection adding extra software and recipes.
- Includes several sublayers: meta-oe, meta-networking, meta-python, meta-perl, etc.

Contents:

- Thousands of recipes for apps, networking tools, Python modules, etc.

How it works:

- If your image requires software like curl, python3, or nginx, Yocto will look in meta-openembedded if not in meta or meta-poky.

---

5 meta-yocto-bsp

What it is:

- BSP (Board Support Package) layer in Yocto.
- Contains machine-specific configuration, kernel recipes, and bootloader support.

Contents:

- Device Tree sources (.dts)

- Machine configurations (*.conf)
- Kernel recipes

How it works:

- When building for MACHINE=beaglebone, BitBake uses configs in this layer to:
  - Select correct kernel
  - Apply patches
  - Generate root filesystem compatible with the board

---

6 meta-bbb

What it is:

- Custom layer for BeagleBone Black.
- Includes project-specific recipes, kernel patches, device tree overlays, and custom apps.

How it works:

- Overrides generic BSP settings from meta-yocto-bsp if needed.
- Adds new software for your BeagleBone Black image.

---

7 meta-qt5

What it is:

- Qt5 support layer for GUI apps.

Contents:

- Qt5 libraries, examples, and tools
- Recipes for building Qt5-based apps

How it works:

- If your image includes GUI apps, Yocto will build Qt5 from this layer.
- Ensures that the libraries are cross-compiled for the target architecture.

---

8 meta-security

What it is:

- Security-focused recipes and configurations.

Contents:

- SELinux / AppArmor support

- Crypto libraries, secure boot support

How it works:

- Adds security features to the target image.
- Integrates with meta and meta-poky recipes for secure builds.

9️⃣ meta-jumpnow

What it is:

- Custom/personal/company layer. Could include demo apps or prebuilt packages.

How it works:

- Adds project-specific software or configurations on top of core layers.
- Can override any recipe from other layers if needed.

1️⃣0️⃣ meta-skeleton

What it is:

- Template layer showing how to structure a Yocto layer.

Contents:

- Example recipes, classes, and configurations.

How it works:

- Helps you create your own layer properly with correct directory structure and metadata.

1️⃣1️⃣ meta-selftest

What it is:

- Self-test layer to verify build and recipes.

Contents:

- Automated tests for BitBake, recipes, classes

How it works:

- Ensures that the build system behaves correctly.
- Used mostly for Yocto development or testing new layers.

1️⃣2️⃣ build

What it is:

- Output directory created after source oe-init-build-env build.

Contents:

- conf/ → local.conf and bblayers.conf
- tmp/ → build output, logs, packages, and images

How it works:

- BitBake writes all intermediate and final files here.
- You can clean, rebuild, or configure builds using this directory.

---

1 3 scripts / contrib

- scripts: Helper scripts for building or testing images.
- contrib: Experimental or community utilities.

---

1 4 source

- Could contain project source code or patches for recipes.

How all layers work together

1. bitbake parses all meta layers (meta, meta-poky, meta-bbb, meta-openembedded, etc.)
2. Resolves dependencies for each recipe
3. Fetches source code (do_fetch)
4. Compiles (do_compile)
5. Installs into root filesystem (do_install)
6. Packages into .ipk or .deb
7. Generates image (core-image-minimal / core-image-sato)

Think of it like a stack:

```
+-----------------------+
| Custom apps (meta-bbb) |
+-----------------------+
| GUI apps (meta-qt5)    |
+-----------------------+
| Security (meta-security)|
+-----------------------+
| Community recipes      |
```

| (meta-openembedded) |

+----------------------+

| Reference distro |

| (meta-poky) |

+----------------------+

| Core recipes (meta) |

+----------------------+

| BitBake engine |

+----------------------+

Common Yocto targets, but they serve very different purposes in a build system.

1️⃣ core-image-minimal

- **Purpose:** A barebones Linux image for your target hardware.
- **Contents:** Only the absolute essentials:
  - Linux kernel
  - BusyBox (basic shell utilities)
  - Package management tools (optional)
- **Use case:**
  - For **testing your hardware**.
  - As a **starting point** for custom embedded images.
  - Minimal footprint → small SD card or flash memory.
- **Size:** Very small (~20–40 MB depending on layers).

2️⃣ core-image-sato

- **Purpose:** A “desktop-like” image with GUI and more user applications.
- **Contents:**
  - X11 server

    - GTK / Qt libraries
    - Window manager (lightweight)
    - Example apps (like puzzles, clocks, etc.)
- **Use case:**
    - For **demonstrating GUIs** on embedded boards like BeagleBone.
    - Testing touchscreen or display output.
- **Size:** Larger than core-image-minimal (usually 200–400 MB or more).

---

3 meta-toolchain

- **Purpose:** Builds a **cross-compilation toolchain**.
- **Contents:**
    - GCC for target architecture
    - Target-specific libraries (glibc, uClibc, or musl)
    - Header files
- **Use case:**
    - For **compiling applications on your host machine** to run on your target.
    - Saves you from compiling directly on the embedded board (very slow for ARM CPUs).
- **Example:**
- bitbake core-image-minimal -c populate_sdk

This generates a .sh SDK installer that you can run on your host.

---

4 meta-ide-support

- **Purpose:** Adds support for IDEs (Eclipse, VS Code, etc.) to work with Yocto projects.
- **Contents:**
    - Project files for IDE integration
    - Debugging helpers
    - Indexing support for code completion
- **Use case:**
    - If you want to **develop applications for your target in an IDE**.
    - Works well with the SDK generated by meta-toolchain.

✅ TL;DR Comparison Table

Target / Layer	Purpose	Contents	Use Case	Size
core-image-minimal	Minimal Linux image	Kernel + BusyBox + essentials	Hardware test, starting point for custom image	Small (~20–40 MB)
core-image-sato	GUI desktop image	X11, GTK/Qt, Window Manager, apps	GUI demos, touchscreen testing	Large (~200–400 MB)
meta-toolchain	Cross-compilation SDK for host	GCC, headers, libraries	Compile target apps on host machine	Varies
meta-ide-support	IDE integration for Yocto projects	Project files, debug helpers	Use IDE for target app development	Minimal

---

BeagleBone Black (BBB) setup.

1️⃣ What is meta-bbb?

meta-bbb is a **Yocto layer** specifically created for **BeagleBone Black** support. In the Yocto Project, a "layer" is a collection of recipes, configuration files, and scripts to add functionality or support for a specific hardware platform, software, or application.

So in simple terms:

meta-bbb tells Yocto **how to build images that can run on the BeagleBone Black.**

2️⃣ Structure of meta-bbb

From your ls, it contains:

- **conf/** → Configuration files. Usually includes layer.conf which tells BitBake that this layer exists and sets some variables.
- **docs/** → Documentation about the layer.
- **README.md** → Layer overview (usually mentions supported images, boards, etc.).
- **recipes-*** → These are the **actual build recipes**. Recipes are .bb files that tell Yocto **how to fetch, configure, compile, and install software**. Examples:
    - recipes-bsp → Board Support Package (BSP) related recipes, e.g., U-Boot, Device Tree, kernel.
    - recipes-core → Core system utilities.
    - recipes-kernel → Linux kernel recipes.

    - recipes-devtools → Tools for development (compilers, debugging, etc.).
    - recipes-qt → Qt5 framework.
    - recipes-connectivity → Networking related packages.
    - recipes-misc → Miscellaneous software.
    - recipes-support → Helper packages, libraries, utilities.
- **images/** → Predefined **images for BBB**, like the ones you listed:
- basic-image.bb
- bt-image.bb
- console-image.bb
- installer-image.bb
- qt5-image.bb

These .bb files define **what goes into each image**, like minimal console-only systems, GUI systems, or images with Bluetooth support.

- **scripts/** → Useful scripts to help with building, flashing, or debugging BBB.

3️⃣ Images in meta-bbb/images

Each .bb file is an **image recipe**:

Image	Purpose
basic-image.bb	Minimal image for BBB with essential packages. Good for testing or building custom apps.
console-image.bb	BBB runs only a console (CLI), no GUI. Useful for development/debugging.
qt5-image.bb	Image with **Qt5 libraries** for GUI apps.
installer-image.bb	Used to create SD card or eMMC installers for BBB.
bt-image.bb	Adds **Bluetooth support** along with basic image.

Tip: These images inherit core-image-* definitions from Yocto's **meta-poky** layer but are customized for BBB.

4️⃣ How is it related to BBB?

- It contains **BSP recipes**: U-Boot, kernel, Device Tree files for BBB.
- It contains **image recipes**: Preconfigured images you can build and flash onto BBB.

- It contains **meta-layer configurations**: Paths, machine settings (MACHINE = beaglebone) for BBB.

So basically:

meta-bbb is the layer that Yocto needs to **generate a working Linux image specifically for your BeagleBone Black**.

5️⃣ How to test these images on BBB

1. **Set up Yocto environment**:

cd ~/poky-zeus-bbb

source oe-init-build-env build

2. **Select machine**:

Edit conf/local.conf and set:

MACHINE = "beaglebone"

3. **Build an image**:

For example, build console-image:

bitbake console-image

- Yocto will fetch all recipes, build kernel, rootfs, U-Boot, and create .img or .wic files.

4. **Flash the image to SD card**:

sudo dd if=tmp/deploy/images/beaglebone/console-image-beaglebone.wic of=/dev/sdX bs=4M

sync

Replace /dev/sdX with your SD card device.

5. **Boot BBB**:

- Insert SD card into BBB.
- Power on BBB.
- Connect via UART (screen /dev/ttyUSB0 115200) or SSH (ssh root@192.168.x.x) depending on network setup.

6. **Verify functionality**:

- For console-image: You should see a Linux CLI login prompt.
- For qt5-image: GUI applications can run if you connect HDMI or VNC.

✅ Summary

- meta-bbb = Yocto support layer for BBB.

- Provides **recipes for kernel, U-Boot, libraries, tools**.
- Provides **predefined images** like console, basic, Qt5 GUI, installer.
- You can **build these images with BitBake**, flash them to SD/eMMC, and run them on BBB.

Yocto build system for BBB.

1️⃣ Yocto Layered Structure for BBB

Yocto builds images by combining **multiple layers**:

Layer	Purpose
meta-bbb	BBB-specific recipes: kernel, U-Boot, device tree, custom images.
meta-poky	Core Yocto recipes, base images like core-image-minimal.
meta-openembedded	Extra software packages (networking, Qt, tools).
meta-qt5	Qt5 libraries and GUI applications (optional, only if building GUI).
meta-skeleton / meta-jumpnow	Example apps, extra BSPs, or project-specific stuff.

Important: Layers are stacked. Yocto merges recipes and configurations from **all layers** to build a final image.

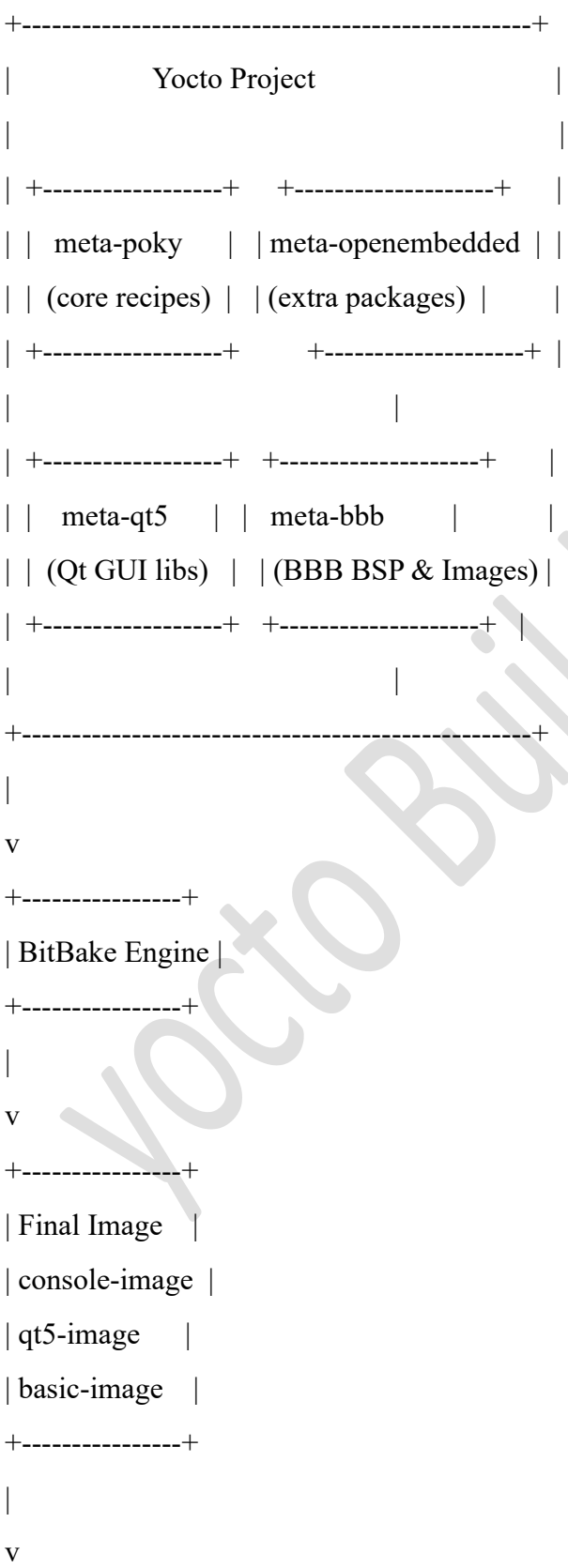
2️⃣ Image Build Flow

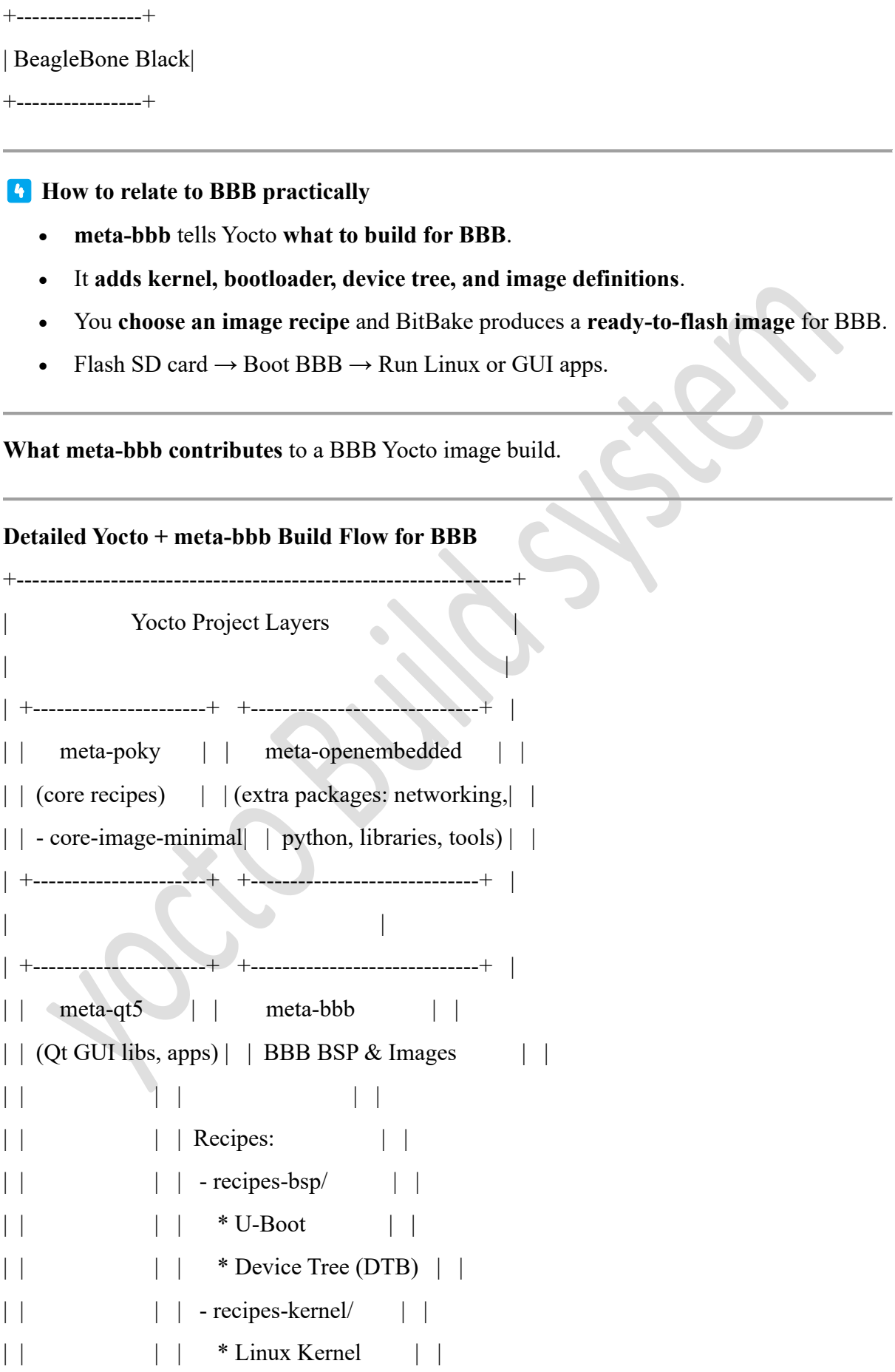
1. **BitBake parses image recipe** (console-image.bb, qt5-image.bb).
2. **Includes layer recipes**:
3. meta-bbb: custom BBB kernel, U-Boot, device tree
4. meta-poky: core utilities, base packages
5. meta-openembedded: networking, libraries, tools
6. meta-qt5: Qt5 libraries (if GUI image)
7. **BitBake builds each component**:
    - Fetches source code (kernel, packages)
    - Compiles and installs
    - Packages into **root filesystem**
    - Combines with **bootloader, kernel, dtb** → final image (.wic or .sdimg)

8. **Deploy to BBB** → boots with the selected image.

---

3 Diagram



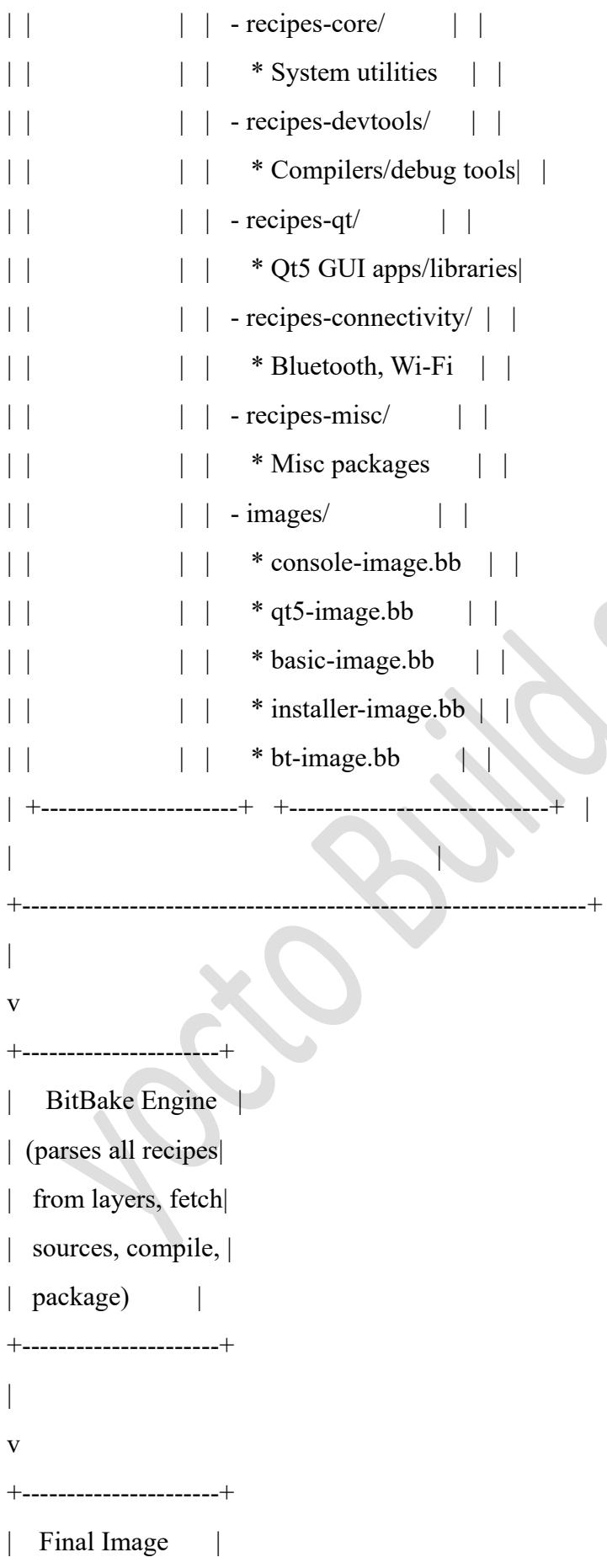


🔄 How to relate to BBB practically

- **meta-bbb** tells Yocto **what to build for BBB**.
- It **adds kernel, bootloader, device tree, and image definitions**.
- You **choose an image recipe** and BitBake produces a **ready-to-flash image** for BBB.
- Flash SD card → Boot BBB → Run Linux or GUI apps.

What meta-bbb contributes to a BBB Yocto image build.

Detailed Yocto + meta-bbb Build Flow for BBB



|----------------------|

| console-image |

| qt5-image |

| basic-image |

| installer-image |

| bt-image |

+----------------------+

|

v

+----------------------+

| BeagleBone Black |

| (bootable SD/eMMC) |

+----------------------+

✅ **Key Takeaways**

1. meta-bbb is **BBB-specific**, contains BSP (kernel, U-Boot, DTBs) and **predefined image recipes**.
2. Each **image recipe** combines different subsets of packages:
   - console-image.bb → minimal CLI system.
   - qt5-image.bb → GUI with Qt5.
   - basic-image.bb → essential apps.
   - installer-image.bb → installer for SD/eMMC.
   - bt-image.bb → Bluetooth + basic image.
3. **BitBake engine** merges all layers, builds, and generates **flashable images** for BBB.
4. You can **test an image** by building it with BitBake and flashing the .wic or .sdimg to BBB.

---

meta-bbb Recipe → Image Contribution Flow

meta-bbb/

│

├── recipes-bsp/

│ ├── u-boot_%.bb → Bootloader (MLO + u-boot.img)

│ └── device-tree_%.bb → Device Tree Blobs (.dtb) for BBB hardware

```
│
├── recipes-kernel/
│   └── linux-beaglebone_%.bb → Compiles BBB Linux kernel (zImage, modules)
│
├── recipes-core/
│   ├── initramfs-tools.bb   → Basic root filesystem setup
│   ├── busybox.bb           → CLI utilities (ls, cp, mkdir…)
│   └── systemd.bb           → Init system to boot services
│
├── recipes-devtools/
│   ├── gcc.bb               → C/C++ compiler for cross-build
│   ├── gdb.bb               → Debugging tools
│   └── strace.bb            → System call tracing tools
│
├── recipes-qt/
│   ├── qtbase.bb            → Qt5 core libraries
│   └── qt5-image.bb         → GUI framework for applications
│
├── recipes-connectivity/
│   ├── bluez.bb             → Bluetooth stack
│   └── wpa-supplicant.bb    → Wi-Fi management
│
├── recipes-misc/
│   └── htop.bb              → Monitoring tool
│
├── images/
│   ├── console-image.bb     → Minimal CLI image
│   ├── basic-image.bb       → Essential apps, minimal GUI optional
│   ├── qt5-image.bb         → Qt5 GUI + basic utilities
│   ├── bt-image.bb          → Bluetooth + basic-image
│   └── installer-image.bb   → SD/eMMC installer image
│
```

└── conf/

└── layer.conf → Registers layer with BitBake, sets paths

How Each Recipe Ends Up in the Final Image

Recipe/Layer	Contributes To	Notes
u-boot (recipes-bsp)	Bootloader (MLO, u-boot.img)	Initializes BBB hardware and loads kernel
device-tree (recipes-bsp)	.dtb files	Hardware description (GPIO, UART, I2C, etc.)
linux-beaglebone (recipes-kernel)	zImage, kernel modules	Main Linux kernel for BBB
busybox (recipes-core)	CLI utilities	Provides ls, cp, mkdir, etc.
systemd (recipes-core)	Init system	Boots services on startup
qtbase (recipes-qt)	Qt5 libraries	Needed for GUI applications
bluez / wpa-supplicant (recipes-connectivity)	Bluetooth/Wi-Fi	Only included in bt-image or custom images
htop (recipes-misc)	Monitoring tool	Optional, usually for debugging
console-image.bb	Minimal CLI root filesystem	Base for other images
basic-image.bb	CLI + essential apps	Can extend console-image
qt5-image.bb	GUI root filesystem with Qt5	GUI-ready BBB image
bt-image.bb	GUI or CLI + Bluetooth	For IoT / wireless projects
installer-image.bb	Complete SD/eMMC installer	Used to flash BBB storage

---

Image Build → Flash → Run on BBB

1. **BitBake reads the image recipe** (e.g., qt5-image.bb).
2. **Resolves dependencies**:
3. qt5-image → qtbase → busybox → systemd → linux-beaglebone → u-boot → device-tree
4. **Builds each component**:
    - Compiles kernel

- Builds root filesystem
- Packages Qt libraries, Bluetooth tools, etc.
- Combines bootloader + dtb + rootfs into .wic image

5. **Flash image to SD card / eMMC** → Insert into BBB → Boot

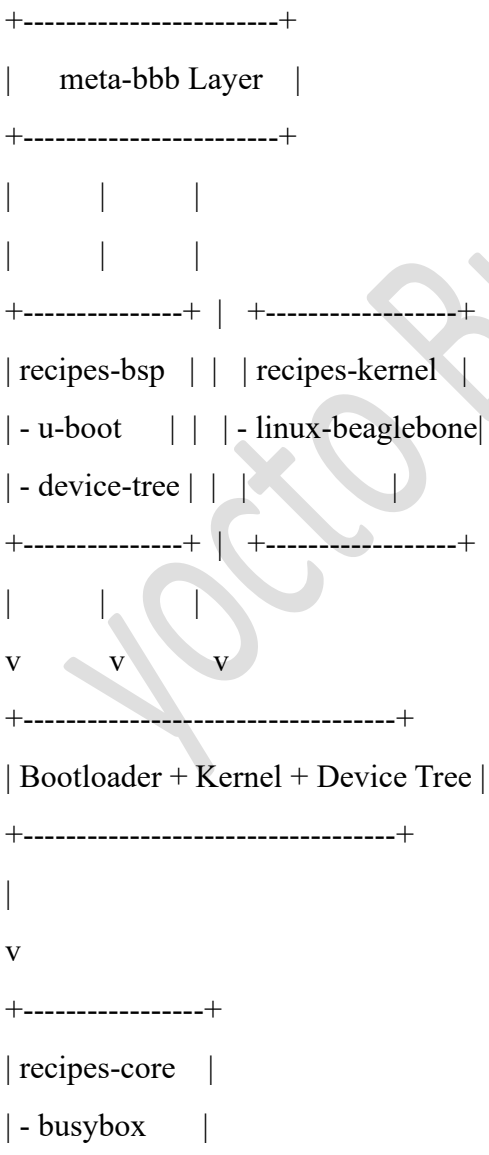
- CLI available for console-image
- Full GUI ready for qt5-image
- Bluetooth works for bt-image
- Installer ready for mass deployment

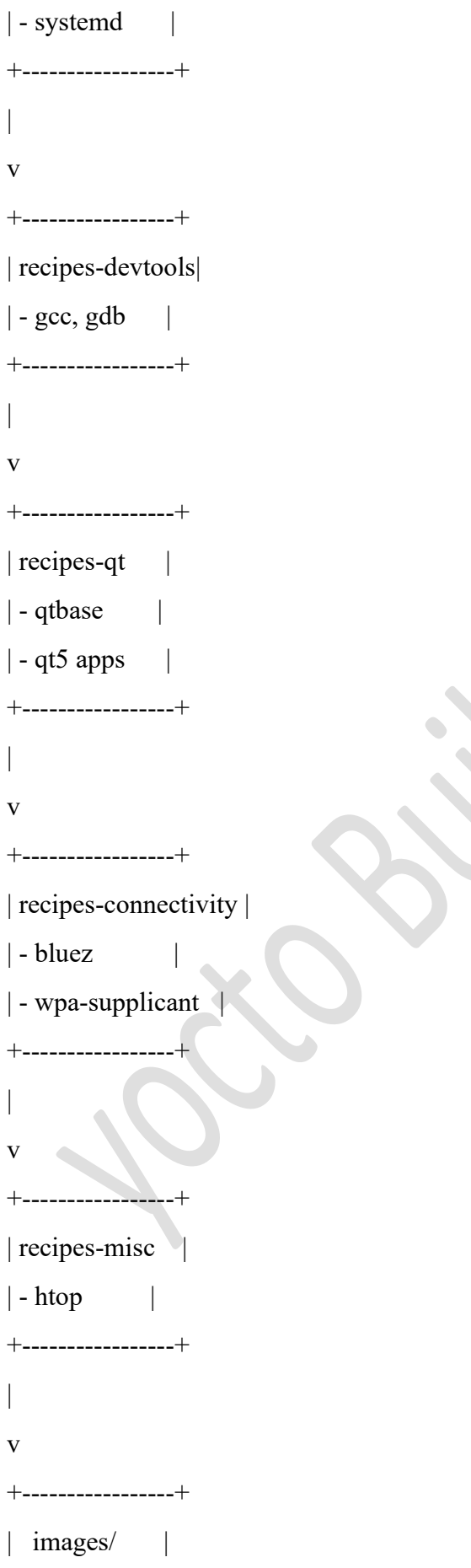
---

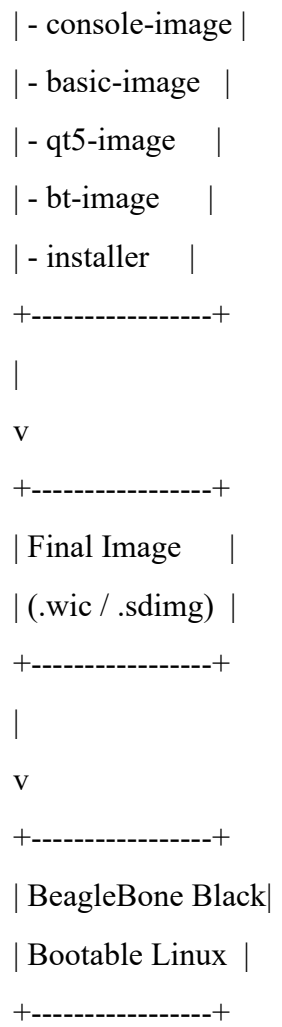
meta-bbb recipe contributes to the final image for BBB

---

meta-bbb → BBB Image Build Flowchart







Flow Explanation

1. **BSP (u-boot + device tree) → Kernel** → Initializes hardware.
2. **Core packages (busybox, systemd) → Root filesystem** → Minimal system functionality.
3. **Dev tools (gcc, gdb) → Optional developer support**.
4. **Qt / Connectivity / Misc packages** → Extra features (GUI, Bluetooth, Wi-Fi, monitoring).
5. **Image recipes** → Select which combination of packages goes into final .wic image.
6. **Flash to BBB** → Boot and run Linux.
