Poky:

The Yocto project provides a reference build system for embedded Linux, called **Poky**, which has the **BitBake and OpenEmbedded-Core (OE-Core)** projects at its base.

The purpose of Poky is to build the components needed for an embedded Linux product, namely:

- A bootloader image
- A Linux kernel image
- A root filesystem image
- Toolchains and software development kits (SDKs) for application development

With these, the Yocto project covers the needs of both system and application When the Yocto project is used as an integration environment for bootloaders kernel, and user space applications, we refer to it as system development.

The Yocto project makes a new release every six months. The latest release at the time of this writing is Yocto 3.3 “HARDKNOTT”

We are using Yocto 3.0.4 “Zeus”

Linux Build System	**Yocto Project Build System**
make	bitbake
Makefile	recipe

Lab Experiment 1: Yocto Zeus 3.0.4 build for Beagle Bone Black (BBB)

Experiment 2: Choose u-boot 2019.07 version and add “beagleboneblack-uboot.git” repository in to yocto project.
ADD “beagleboneblack-uboot.git” repository in meta layer. Repository Details: Branch: master Commit ID: c3929a23a38d0aaa46e1fabd50c9486de41452e3 URL: https://github.com/kernelmasters/beagleboneblack-uboot.git
Implementation: meta/recipes-bsp/u-boot/ u-boot-common.inc u-boot_2019.07.bb
Expected Results: Switch, LED Mux configurations disabled in u-boot source code. “gpio input 11” & “gpio output 10” commands are not working in u-boot command prompt.
Modifications: Tag: Task14 Branch: Commit Message: To add beagleboneblack-uboot repo in to yocto project

Experiment 3: To apply patches in to "beagleboneblack-uboot.git" repository

Enable LED, Switch mux configuration in u-boot source code and create patches.
Apply this patches in to "beagleboneblack-uboot.git" repository in meta-bbb layer.

Implementation:

```
meta-bbb/recipes-bsp/u-boot/
                            u-boot_%.bbappend
                            files/
                                 0001-Setup-U-boot-Environment.patch
                                 0002-Select-LCD_DATA14-pin-functionality-to-GPIO0_10-to-c.patch
                                 0003-Select-LCD_DATA15-pin-functionality-to-GPIO0_11-to-r.patch
```

Expected Results:

"gpio input 11" & "gpio output 10" commands are working in u-boot command prompt.

Modifications:

Tag: Task15

Branch:

Commit Message: To apply beagleboneblack-uboot patches in to yocto project

Experiment 4: Choose Kernel 4.19.94 version and add "beagleboneblack-kernel.git" repository in to yocto project

ADD "beagleboneblack-kernel" repository in meta layer.

Repository Details:

Branch: master
Commit ID: 58ac7b864e15789f208b285202bbc31c91edd259
URL: https://github.com/kernelmasters/beagleboneblack-kernel.git

```
meta/ recipes-kernel/linux/

meta-bbb/conf/local.conf.sample
                    PREFERRED_VERSION_linux-stable = "4.19.94"
meta-bbb/recipes-kernel/linux/linux-stable_4.19.bb
meta-bbb/recipes-kernel/linux/linux-stable-4.19/beaglebone/defconfig
```

Expected Results:

Input event device file is not created in /dev/input folder.
"evtest" application unable to read switch events from Linux user command prompt.

Modifications:

Tag: Task16

Branch:

Commit Message: To add beagleboneblack-kernel repo in to yocto project

Experiment 5: To apply patches in to "beagleboneblack-kernel.git" repository

Apply patches in to "beagleboneblack-kernel" repository in meta-bbb layers.

```
meta-bbb/recipes-kernel/linux/
                            linux-stable_4.19.bb
meta-bbb/recipes-kernel/linux/linux-stable-4.19/
                              0001-Setup-Kernel-Environment.patch
                              0002-Mux-Config-in-DTS-LCD_DATA_13.GPIO0_9-LCD_DATA14.GPI.patch
                              0005-Enter-Switch-GPIO11-Raising-Edge-Interrupt-porting-o.patch
```

Expected Results:

Input event device file is created in /dev/input folder.

“evtest” application properly read switch events from Linux user command prompt.
Modifications: **Tag:** Task17 **Branch:** **Commit Message:** To apply patches in to beaglebone-kernel repo.

Experiment 6: To apply patches in to “beagleboneblack-kernel.git” repository
Apply patches in to “beagleboneblack-kernel” repository in meta-bbb layers.
meta-bbb/recipes-kernel/linux/ linux-stable_4.19.bb meta-bbb/recipes-kernel/linux/linux-stable-4.19/ 0001-Setup-Kernel-Environment.patch 0002-Mux-Config-in-DTS-LCD_DATA_13.GPIO0_9-LCD_DATA14.GPI.patch 0005-Enter-Switch-GPIO11-Raising-Edge-Interrupt-porting-o.patch
Expected Results: Input event device file is created in /dev/input folder. “evtest” application properly read switch events from Linux user command prompt.
Modifications: **Tag:** Task16 **Branch:** **Commit Message:** To apply patches in to beaglebone-kernel repo.

Lab Experiment 7: Create a Yocto layer

Add a custom layer to the Yocto project for your project needs

- Create a new Yocto layer
- Interface this custom layer to the existing Yocto project
- Use applications from custom layers

Commands:

```
$ bitbake-layers –h
$ bitbake-layers show-layers
```

Step 1: Create a new layer

This new layer was responsible for Kernel Masters Beagle Bone Black Expansion board support in Yocto.

```
$ bitbake-layers create-layer --help
$ cd ~/poky-zeus-bbb
$ bitbake-layers create-layer meta-km-bbb
```

With the above commands, create a new Yocto layer named “**meta-km-bbb”** with a priority of 7.
Before using the new layer, we need to configure its generated configuration files.

You can start with the README file which is not used in the build process but contains information related to layer maintenance.
You can then check, and adapt if needed, the global layer configuration file located in the conf directory of your custom layer

Step 2: Integrate a layer to the build

"build/conf/bblayers.conf" file which contains all the paths of the layers we use.
In this file add the full path to our newly created layer to the list of layers.

Validate the integration of the meta-km-bbb layer with:

```
$ bitbake-layers show-layers
```

and make sure you don't have any warning from bitbake.

Step 3: Add a recipe to the layer
You should instead always **use a custom layer** to add recipes or to customize the existing ones.

```
$ bitbake-layers show-recepies
$ cd ~/poky-zeus-bbb
$ mkdir -p meta-km-bbb/recipes-km-bbb-kernel/linux
```

Create "linux-stable_%.bbappend" file inside linux folde and write the below content.

```
FILESEXTRAPATHS_prepend := "${THISDIR}/patches:"
SRC_URI += "file://0001-volume-up-down-switches-ADDED.patch"
```

Create a patches folder inside linux folder and copy patches in this folder.

```
$ bitbake-layers show-recipes | grep km-bbb
```

You can also find detailed information on available packages, their current version, dependencies or the contact information of the maintainer by visiting
http://recipes.yoctoproject.org/

```
$ bitbake linux-stable –c cleanall
$ bitbake -v linux-stable
```

Expected Results:

```
NOTE: linux-stable-4.19.94-r0 do_patch: Applying patch '0001-Setup-Kernel-Environment.patch' (../meta-bbb/recipes-kernel/linux/linux-stable-4.19/0001-Setup-Kernel-Environment.patch)

NOTE: linux-stable-4.19.94-r0 do_patch: Applying patch '0002-Mux-Config-in-DTS-LCD_DATA_13.GPIO0_9-LCD_DATA14.GPI.patch'
 (../meta-bbb/recipes-kernel/linux/linux-stable-4.19/0002-Mux-Config-in-DTS-LCD_DATA_13.GPIO0_9-LCD_DATA14.GPI.patch)
```

NOTE: linux-stable-4.19.94-r0 do_patch: Applying patch '0005-Enter-Switch-GPIO11-Raising-Edge-Interrupt-porting-o.patch' (../meta-km-bbb/recipes-km-bbb-kernel/linux/patches/0005-Enter-Switch-GPIO11-Raising-Edge-Interrupt-porting-o.patch)

Commit ID: “Create meta-km-bbb layer and also create recipes-bsp, recipes-kernel inside the layer”

<u>The following packages build in console-image:</u>

```
$ bitbake -DDD console-image
```

```
DEBUG: sorted providers for console-image are: ['/home/km/poky-zeus-bbb/meta-bbb/images/console-image.bb']
DEBUG: adding /home/km/poky-zeus-bbb/meta-bbb/images/console-image.bb to satisfy console-image
DEBUG: Added dependencies ['qemuwrapper-cross', 'depmodwrapper-cross', 'cross-localedef-native'] for /home/km/poky-zeus-bbb/meta-bbb/images/console-image.bb
DEBUG: Added runtime dependencies ['openssh', 'openssh-keygen', 'openssh-sftp-server', 'packagegroup-core-boot', 'term-prompt', 'tzdata', 'binutils', 'binutils-symlinks', 'coreutils', 'cpp', 'cpp-symlinks', 'diffutils', 'elfutils', 'elfutils-binutils', 'file', 'gcc', 'gcc-symlinks', 'g++', 'g++-symlinks', 'gdb', 'gettext', 'git', 'ldd', 'libstdc++', 'libstdc++-dev', 'libtool', 'ltrace', 'make', 'perl-modules', 'pkgconfig', 'python3-modules', 'strace', 'bzip2', 'curl', 'dosfstools', 'e2fsprogs-mke2fs', 'ethtool', 'fbset', 'findutils', 'grep', 'i2c-tools', 'ifupdown', 'iperf3', 'iproute2', 'iptables', 'less', 'lsof', 'netcat-openbsd', 'nmap', 'ntp', 'ntp-tickadj', 'parted', 'procps', 'rndaddtoentcnt', 'sysfsutils', 'tcpdump', 'util-linux', 'util-linux-blkid', 'unzip', 'wget', 'zip', 'kernel-modules', 'emmc-upgrader', 'firewall', 'serialecho', 'spiloop', 'bbgw-wireless', 'crda', 'iw', 'linux-firmware-wl12xx', 'linux-firmware-wl18xx', 'wpa-supplicant', 'checksec', 'ncrack', 'nikto', 'python3-scapy', 'wireguard-init', 'wireguard-tools', 'run-postinsts', 'opkg', 'base-passwd', 'shadow', 'base-passwd', 'shadow', 'locale-base-en-us'] for /home/km/poky-zeus-bbb/meta-bbb/images/console-image.bb
```

Lab Experiment 8: Add an existing application

Example: evtest application

Step 1: Add an “evtest” package in to configuration file

Find out “evtest” package in yocto repo.

```
~/poky-zeus-bbb$ find meta* | grep evtest
meta-openembedded/meta-oe/recipes-test/evtest
meta-openembedded/meta-oe/recipes-test/evtest/evtest_1.34.bb
meta-openembedded/meta-oe/recipes-test/evtest/evtest
meta-openembedded/meta-oe/recipes-test/evtest/evtest/add_missing_limits_h_include.patch
```

The IMAGE_INSTALL variable controls the packages included into the output image.
To illustrate this, add the “evtest” to the list of enabled packages.
You can add packages (IMAGE_INSTALL += “evtest”) to be built by editing the below any one of the configuration file.

Local configuration file: (not recommended)
build/conf/local.conf.
image configuration file:
meta-bbb/images/console-image.bb

Step 2: Build “evtest” package

```
$ bitbake evtest
```

Starts a shell with the “evtest” environment set up for development/debugging. (Optional)

```
$ bitbake evtest -c devshell
```

Step 3: Add an “evtest” binary in to the rootfs image

```
$ bitbake console-image
```

Step 4: Verify “evtest” package

```
~/poky-zeus-bbb/build$ cat tmp/deploy/images/beaglebone/console-image-beaglebone-20210604183836.rootfs.manifest | grep evtest
evtest cortexa8hf-neon 1.34-r0
```

manifest file syntax:
<Package-name> <arch-name> <package-version>

Step 5: Expected Results

Copy rootfs image in to target board.

```
$ sudo ./km-bbb-yocto-install.sh --mmc /dev/sdb
```

Test “”evtest” package on target board, application runs properly.

```
root@beaglebone:~# evtest
No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:      gpio-keys
```

```
Select the device event number [0-0]: 0
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 28 (KEY_ENTER)
Key repeat handling:
  Repeat type 20 (EV_REP)
    Repeat code 0 (REP_DELAY)
      Value    250
    Repeat code 1 (REP_PERIOD)
      Value     33
Properties:
Testing ... (interrupt to exit)
Event: time 1622869843.388934, type 1 (EV_KEY), code 28 (KEY_ENTER), value 1
Event: time 1622869843.388934, -------------- SYN_REPORT ------------
Event: time 1622869843.600816, type 1 (EV_KEY), code 28 (KEY_ENTER), value 0
Event: time 1622869843.600816, -------------- SYN_REPORT ------------
Event: time 1622869843.889188, type 1 (EV_KEY), code 28 (KEY_ENTER), value 1
Event: time 1622869843.889188, -------------- SYN_REPORT ------------
Event: time 1622869844.051879, type 1 (EV_KEY), code 28 (KEY_ENTER), value 0
```

Tag:
Commit ID: “Add an “evtest” package in to rootfs”

Lab Experiment 9: Add a custom application

Add a new recipe to support a required custom application

Example: Hello World Application

Step 1: Setup and organization

In this lab we will add a recipe handling the “helloworld” application.
Create a “recipes-helloworld” directory under meta-km-bbb layer.
Create a “helloworld” sub directory for application under “recipes-helloworld” directory.

```
$ mkdir -p  meta-km-bbb/recipes-helloworld/helloworld
```

A recipe for an application is usually divided into a version specific bb file and a common one.
Try to follow this logic and separate the configuration variables accordingly.
Tip: it is possible to include a file into a recipe with the keyword require.

The IMAGE_INSTALL variable controls the packages included into the output image.
You can add packages (IMAGE_INSTALL += “helloworld”) to be built by editing the below any one of the configuration file.

Local configuration file: (not recommended)
build/conf/local.conf.
image configuration file:
meta-bbb/images/console-image.bb

Step 2: About helloworld application

The helloworld application is print “hello world” to the monochrome LCD.

Uses autotools build system.

First try to find the project homepage, download the sources and have a first look: license, Makefile, requirements…

Source Code URL: https://github.com/kernelmasters/yocto-helloworld.git

Step 3: Write the common recipe

Create an appropriate common file, ending in .inc (helloworld.inc)
In this file add the common configuration variables: source URI, description…

```
DESCRIPTION = "Example Hello, World application for Yocto build."
SRCREV = "fba6d4539310ced763e3a1c24ce9c4459b2fc8fa"
SRC_URI = "git://github.com/kernelmasters/yocto-helloworld.git"
```

Step 4: Write the version specific recipe
Create a file that respects the Yocto nomenclature: ${PN}_${PV}.bb (helloworld_1.0.bb)
Add the required common configuration variables: archive checksum, license file checksum, package revision…

Generate LICENSE
```
$ md5sum LICENSE
b2424aed282074c27f97c20dcbf25df3  LICENSE
```

```
require helloworld.inc
SECTION = "helloworld"
DEPENDS = ""
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://LICENSE;md5=b2424aed282074c27f97c20dcbf25df3"
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}-${PV}:"
S = "${WORKDIR}/git"
inherit autotools
```

Step 5: Testing and troubleshooting
You can check the whole packaging process is working fine by explicitly running the build task on the "helloworld" recipe:

```
$ bitbake helloworld
```

Try to make the recipe on your own. Also eliminate the warnings related to your recipe: some configuration variables are not mandatory but it is a very good practice to define them all.

If you hang on a problem, check the following points:
- The common recipe is included in the version specific one
- The checksum and the URI are valid
- The dependencies are explicitly defined
- The internal state has changed, clean the working directory:

```
$ bitbake -c cleanall helloworld
```

Tip: BitBake has command line flags to increase its verbosity and activate debug outputs. Also, remember that you need to cross-compile "helloworld" for ARM! Maybe, you will have to configure your recipe to resolve some mistakes done in the application's Makefile (which is often the case).

A bitbake variable permits to add some Makefile's options, you should look for it.

Step 6: Update the rootfs and test
Now that you've compiled the "helloworld" application, generate a new rootfs image.
```
$ bitbake console-image.
```

```
~/poky-zeus-bbb/build$ cat tmp/deploy/images/beaglebone/console-image-beaglebone-20210604200624.rootfs.manifest | grep helloworld
helloworld cortexa8hf-neon 1.0-r0
```

Access the board command line through SSH. You should be able to launch the "hellowrold" program. Now, it's time to play!

Tag:
Commit ID: "Add a 'helloworld' custom package in to rootfs"

Lab Experiment 10: Develop your application in the Poky SDK

Step 1: Build SDK

Build an SDK for the "console-image" image, with the populate_sdk task.
The below command to create SDK image

```
$ bitbake console-image -c populate_sdk
```

Once the SDK is generated, a script will be available at tmp/deploy/sdk

Step 2: Install the SDK

Open a new console to be sure that no extra environment variable is set. We mean to show you how the SDK sets up a fully working environment.
Install the SDK in ~/sdk by executing the script generated at the previous step.

```
km@KernelMasters:~/sdk$ ../poky-zeus-bbb/build/tmp/deploy/sdk/poky-glibc-x86_64-console-image-cortexa8hf-neon-beaglebone-toolchain-3.0.4.sh
Poky (Yocto Project Reference Distro) SDK installer version 3.0.4
=================================================================
Enter target directory for SDK (default: /opt/poky/3.0.4): /home/km/sdk
You are about to install the SDK to "/home/km/sdk". Proceed [Y/n]? Y
Extracting SDK.........................................................................................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
 $ . /home/km/sdk/environment-setup-cortexa8hf-neon-poky-linux-gnueabi
```

The below command to create SDK Extension image (optional)

```
$ bitbake –c populate_sdk_ext <console-image>
esdk image (dev tool -> recipes)
 o/p -> .sh
bitbake –DDD
```

Step 3: Set up the environment

Go into the directory where you installed the SDK (~/sdk). Source the environment script:
source environment-setup-cortexa8hf-vfp-neon-poky-linux-gnueabi
Have a look at the exported environment variables:

```
$ env
```

Step 4: Compile a "helloworld" application in the SDK

Download helloworld application:

```
$cd ~/sdk
$ git clone https://github.com/kernelmasters/yocto-helloworld.git
```

The below command to build helloworld package inside SDK

```
 $ source environment-setup-cortexa8hf-neon-poky-linux-gnueabi
 $ cd ~/sdk/yocto-helloworld/
 $ env | grep CONFIGURE_FLAGS
 $ aclocal
 $ autoheader
 $ libtoolize
 $ automake --add-missing
 $ autoconf
 $ ./configure ${CONFIGURE_FLAGS}
 $  make
```

Step 5: Test "helloworld" application in target board

Copy the binary to the board, by using the scp command. Then run it and play a bit to ensure it is working fine!

```
 # ./helloworld
```

Lab Experiment 11: Create a custom machine configuration

Let Poky know about your hardware!

During this lab, you will:

• Create a custom machine configuration
• Understand how the target architecture is dynamically chosen

Create a custom machine

The machine file configures various hardware related settings.
While it is not necessary to make our custom machine image here, we'll create a new one to demonstrate the process.
Add a new km-bbb machine to the previously created layer (meta-km-bbb), which will make the BeagleBone properly boot.
This machine describes a board using the cortexa8thf-neon tune and is a part of the ti33x SoC family.

Add the following lines to your machine configuration file:

```
require conf/machine/include/ti-soc.inc
DEFAULTTUNE = "armv7athf-neon"
require conf/machine/include/tune-cortexa8.inc
```

Populate the machine configuration

This km-bbb machine needs:

• To select "beagleboneblack-kernel" as the preferred provider for the kernel.
• To use km-bbb-am335x.dtb device tree.
• To select "beagleboneblack-uboot" as the preferred provider for the bootloader.

• To use arm as the U-Boot architecture.
• To use am335x_boneblack_config as the U-Boot configuration target.
• To use 0x80008000 as the U-Boot entry point and load address.
• To use a zImage kernel image type.
• To configure the serial console to 115200 ttyS0
• And to support some features:
– apm
– usbgadget
- usbhost
– vfat
– ext2
– alsa

Build an image with the new machine
You can now update the MACHINE variable value in the local configuration and start a fresh build.
MACHINE = km-bbb

Check generated files are here and correct
Once the generated images supporting the new km-bbb machine are generated, you can check all the needed images were generated correctly.
Have a look in the output directory, in $BUILDDIR/tmp/deploy/images/km-bbb/. Is there something missing?

Update the rootfs
You can now update your root filesystem, to use the newly generated image supporting our km-bbb machine!

Yocto is a set of tools for building a custom embedded Linux distribution. The systems are usually targeted at particular applications like commercial products.

Yocto Configuration

Yocto uses meta-layers to define the configuration for a system build. Within each meta-layer are recipes, classes and configuration files that support the primary python build tool, bitbake.

The meta-bbb layer generates some basic systems with packages that support C, C++, Qt5, Perl and Python development, the languages and tools we commonly use. Other languages are supported.

We use this layer as a template when starting new BeagleBone projects.

And we create a new layer "meta-km-bbb" for KM-BBB Expansion board.

Version Information

 Yocto Version: Zeus 3.0.3
 Bitbake Version: 1.44
 u-boot version: 2019
 Kernel Version: 4.19.94

Ubuntu Setup
The below packages installed in Ubuntu 18.04

```
$ sudo apt install build-essential chrpath diffstat gawk libncurses5-dev python3-distutils texinfo
```

For all versions of Ubuntu, you should change the default Ubuntu shell from dash to bash by running this command from a shell.

```
    $ sudo dpkg-reconfigure dash
```
Choose No to dash when prompted.

Download Yocto Source code Zeus Version 3.0.3

Enter Home Directory
```
 $ cd ~
```

Clone poky-zeus-bbb repo
```
~$ git clone git@github.com:kernelmasters/poky-zeus-bbb.git
```

Initialize the build directory

Enter poky repo folder
```
~$ cd poky-zeus-bbb
```

Initialize the build directory

```
~/poky-zeus-bbb$ source oe-init-build-env
```

Customize the configuration files:

```
  ~$ cp ~/poky-zeus-bbb/meta-bbb/conf/local.conf.sample ~/poky-zeus-bbb/build/conf/local.conf
  ~$ cp ~/poky-zeus-bbb/meta-bbb/conf/bblayers.conf.sample ~/poky-zeus-bbb/build/conf/bblayers.conf
```

Edit local.conf

The variables you may want to customize are the following:

TMPDIR
DL_DIR
SSTATE_DIR
The defaults for all of these work fine. Adjustments are optional.

TMPDIR

This is where temporary build files and the final build binaries will end up. Expect to use at least 35GB. You probably want at least 50GB available.
The default location is in the build directory, in this example ~/bbb/build/tmp.
If you specify an alternate location as I do in the example conf file make sure the directory is writable by the user running the build.

DL_DIR

This is where the downloaded source files will be stored. You can share this among configurations and build files so I created a general location for this outside the project directory. Make sure the build user has write permission to the directory you decide on.
The default location is in the build directory, ~/bbb/build/sources.

SSTATE_DIR

This is another Yocto build directory that can get pretty big, greater then 5GB. I often put this somewhere else other then my home directory as well.
The default location is in the build directory, ~/bbb/build/sstate-cache.

ROOT PASSWORD

There is only one login user by default, root.

The default password is set to km by these two lines in the local.conf file

```
INHERIT += "extrausers"
EXTRA_USERS_PARAMS = "usermod -P km root; "
```

These two lines force a password change on first login

```
INHERIT += "chageusers"
CHAGE_USERS_PARAMS = "chage -d0 root; "
```

You can comment them out if you do not want that behavior.

Run the build

You need to source the Yocto environment into your shell before you can use bitbake. The oe-init-build-env will not overwrite your customized conf files.

```
~$ source ~/poky-zeus-bbb/oe-init-build-env

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

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    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

~/bbb/build$
```

I don’t use any of the Common targets, but instead use my own custom image recipes.

There are a few custom images available in the meta-bbb layer. The recipes for the images can be found in meta-bbb/images/

console-image.bb
qt5-image.bb
installer-image.bb
You should add your own custom images to this same directory.

console-image

A basic console developer image. See the recipe meta-bbb/images/console-image.bb for specifics, but some of the installed programs are

gcc/g++ and associated build tools
git
ssh/scp server and client
python3 with a number of modules
The console-image has a line

```
inherit core-image
```
which is poky-dunfell/meta/classes/core-image.bbclass and pulls in some required base packages. This is useful to know if you create your own image recipe.

qt5-image

This image includes the console-image and adds Qt5 runtime libraries.

installer-image

This is a minimal image meant only to run from an SD card and whose only purpose is to perform an eMMC installation.

Build

To build the console-image run the following command

```
~/poky-zeus-bbb/build$ bitbake console-image
```
You may occasionally run into build errors related to packages that either failed to download or sometimes out of order builds. The easy solution is to clean the failed package and rerun the build again.

For instance if the build for zip failed for some reason, I would run this

```
~/bbb/build$ bitbake -c cleansstate zip
~/bbb/build$ bitbake zip
```

And then continue with the full build.

```
~/poky-zeus-bbb/build$ bitbake console-image
```

The cleansstate command (with two s's) works for image recipes as well.

The image files won't get deleted from the TMPDIR until the next time you build

Copying the binaries to an SD card

After the build completes, the bootloader, kernel and rootfs image files can be found in /deploy/images/beaglebone/.

The meta-bbb/scripts directory has some helper scripts to format and copy the files to a microSD card.
mk2parts.sh

This script will partition an SD card with the minimal 2 partitions required for the boards.

Insert the microSD into your workstation and note where it shows up.

lsblk is convenient for finding the microSD card.

For example

```
km@kernelmasters:~/bbb/meta-bbb$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda       8:0    0 931.5G  0 disk
|-sda1    8:1    0  93.1G  0 part /
|-sda2    8:2    0  93.1G  0 part /home
|-sda3    8:3    0  29.8G  0 part [SWAP]
|-sda4    8:4    0     1K  0 part
|-sda5    8:5    0   100G  0 part /oe5
|-sda6    8:6    0   100G  0 part /oe6
|-sda7    8:7    0   100G  0 part /oe7
|-sda8    8:8    0   100G  0 part /oe8
|-sda9    8:9    0   100G  0 part /oe9
`-sda10   8:10   0 215.5G  0 part /oe10
sdb       8:16   1   7.4G  0 disk
|-sdb1    8:17   1    64M  0 part
`-sdb2    8:18   1   7.3G  0 part
```

I would use sdb for the format and copy script parameters on this machine.

It doesn't matter if some partitions from the SD card are mounted. The mk2parts.sh script will unmount them.

BE CAREFUL with this script. It will format any disk on your workstation

```
~$ cd ~/poky-zeus-bbb/meta-bbb/scripts
~/poky-zeus-bbb/meta-bbb/scripts$ sudo ./mk2parts.sh sdb
```
You only have to format the SD card once.

/media/card

You will need to create a mount point on your workstation for the copy scripts to use.

```
~$ sudo mkdir /media/card
```
You only have to create this directory once.

copy_boot.sh

This script copies the bootloaders (MLO and u-boot) to the boot partition of the SD card.

The script also copies a uEnv.txt file to the boot partition if it finds one in either

<TMPDIR>/deploy/images/beaglebone/
or in the local directory where the script is run from.

If you are just starting out, you might just want to do this

```
~/poky-zeus-bbb/meta-bbb/scripts$ cp uEnv.txt-example uEnv.txt
```
This copy_boot.sh script needs to know the TMPDIR to find the binaries. It looks for an environment variable called OETMP.

For instance, if I had this in the local.conf

```
TMPDIR="../../build/tmp"
```
Then I would export this environment variable before running copy_boot.sh

```
~/poky-seus-bbb/meta-bbb/scripts$ export OETMP="../../build/tmp"
```
Then run the copy_boot.sh script passing the location of SD card

```
~$ sudo umount /dev/sdb1
~$ sudo umount /dev/sdb2
~/poky-zeus-bbb/meta-bbb/scripts$ ./copy_boot.sh sdb
```
This script should run very fast.

copy_rootfs.sh

This script copies the zImage Linux kernel, the device tree binaries and the rest of the operating system to the root file system partition of the SD card.

The script accepts an optional command line argument for the image type, for example console or qt5. The default is console if no argument is provided.

The script also accepts a hostname argument if you want the host name to be something other then the default beaglebone.

Here’s an example of how you’d run copy_rootfs.sh

```
~/poky-zeus-bbb/meta-bbb/scripts$ ./copy_rootfs.sh sdb console
```

or

```
~/poky-zeus-bbb/meta-bbb/scripts$ ./copy_rootfs.sh sdb qt5 bbb
```

The copy_rootfs.sh script will take longer to run and depends a lot on the quality of your SD card. With a good Class 10 card it should take less then 30 seconds.

The copy scripts will NOT unmount partitions automatically. If an SD card partition is already mounted, the script will complain and abort. This is for safety, mine mostly, since I run these scripts many times a day on different machines and the SD cards show up in different places.

Here’s a realistic example session where I want to copy already built images to a second SD card that I just inserted.

```
~$ sudo umount /dev/sdb1
~$ sudo umount /dev/sdb2
~$ export OETMP="../../build/tmp"
~$ cd ~/poky-zeus-bbb/meta-bbb/scripts
~/poky-zeus-bbb/meta-bbb/scripts$ ./copy_boot.sh sdb
~/poky-zeus-bbb/meta-bbb/scripts$ ./copy_rootfs.sh sdb console bbb2
```

Both copy_boot.sh and copy_rootfs.sh are simple scripts meant to be modified for custom use.

Booting from the SD card

The default behavior of the beaglebone is to boot from the eMMC first if it finds a bootloader there.

Holding the S2 switch down when the bootloader starts will cause the BBB to try booting from the SD card first. The S2 switch is above the SD card holder.

If you are using a cape, the S2 switch is usually inaccessible or at least awkward to reach. From the back of the board a temporary jump of P8.43 to ground when the bootloader starts will do the same thing as holding the S2 switch.