Getting started with Buildroot - Lab

Trevor Woerner, Togán Labs

March 13, 2019

These lab instructions are written for the *Getting started with Buildroot* tutorial of the *Embedded Apprentice Linux Engineer* track. They are designed to work for the *PocketBeagle* hardware platform.

This lab is broken out into two separate paths:

- Basic Lab
- In-Depth Lab

Both labs start with the same resources, and both end up creating the same artifacts, but they both take different routes. If you would like to get an image up-and-running on your board without worrying too much about the details, take a look at the **Basic Lab**. If you'd like to know a little more about what's going on "under the hood", try the **In-Depth Lab**.

Both labs start with the same **Initial Setup**.

All the work for these labs occur on the **host** computer, not the target. A reasonably recent Linux machine/VM is required for this work.

Initial Setup

Our first step is to obtain the buildroot meta-data. Normally this would be done by simply cloning from buildroot's git repository. But I've created a simple fork of upstream that contains little tweaks to make this lab easier.

Therefore start by grabbing a tarball of this lab's buildroot and unpacking it. Then, move into the top-level directory of the unpacked tarball.

```
$ wget https://cm.e-ale.org/2019/SCaLE17x/buildroot/buildroot-e-ale.tar.xz
$ xz -d < buildroot-e-ale.tar.xz | tar xf -
$ cd buildroot-e-ale
```

If downloading `buildroot-e-ale.tar.xz` is taking too long, you can also run these exercises with the `buildroot-e-ale_SM.tar.xz` tarball. But then your build will be slower.
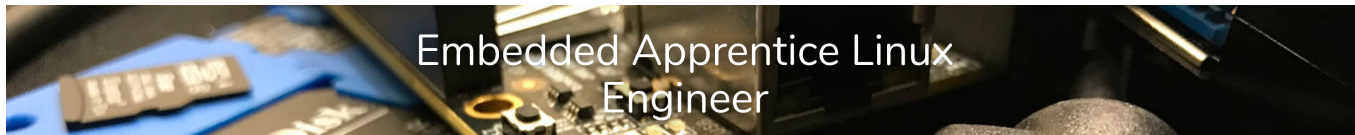
Basic Lab

A great place to start with any project is from a **known location**. In this case our buildroot already knows what a **pocketbeagle** is, so we simply tell buildroot we want to build a basic image for this board, then go ahead and build it.

```
$ make pocketbeagle_defconfig
$ make
```

The build will take a while (15 - 30 minutes, perhaps more depending on the speed of your Internet connection or on the capabilities of your host machine).

Now jump ahead all the way to the **Testing The Build** section.

In-Depth Lab

What if buildroot didn't know what a **pocketbeagle** is? In this lab we're going to configure our pocketbeagle build from scratch.

Creating a minimal configuration

We start by configuring our build:

```
$ make menuconfig
```

In the configuration, we'll have to customize a number of options, as detailed below. Of course, take this opportunity to navigate in all the options, and discover what Buildroot can do.

- In *Target options*
  - Change *Target architecture* to *ARM (little endian)*
  - Change *Target architecture variant* to *Cortex-A8*
- In *Build options*
  - set *global patch directories* to `board/pocketbeagle/patches/`. This will allow us to put patches for Linux, U-Boot other packages in subdirectories of `board/pocketbeagle/patches/`.
- In *Toolchain*
  - Change *Toolchain type* to *External toolchain*. By default, Buildroot builds its own toolchain, but it can also use pre-built external toolchain. We'll use the latter, in order to save build time.
- In *System configuration*
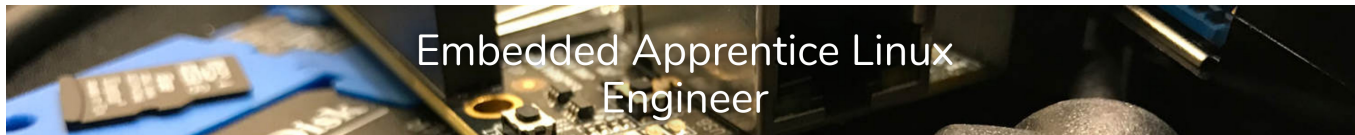
- you can customize the *System host name* and *System banner* if you wish. Keep the default values for the rest.

- In *Kernel*

  - Enable the *Linux kernel*, obviously!

  - Patches will already be applied to the kernel, thanks to us having defined a *global patch directory* above.

  - Choose `omap2plus` as the *Defconfig name*

  - We'll need the Device Tree of the PocketBeagle, so enable *Build a Device Tree Blob (DTB)*

  - And use `am335x-pocketbeagle` as the *Device Tree Source file names*

- In *Target packages*

  - we'll keep just Busybox enabled for now. In the next sections, we'll enable more packages.

- In *Filesystem images*

  - enable *ext2/3/4 root filesystem*

  - select the *ext4* variant

  - you can also disable the *tar* filesystem image, which we won't need.

- In *Bootloaders*

  - enable *U-Boot*, and in *U-Boot*:

    * Switch the *Build system* option to *Kconfig*: we are going to use a modern U-Boot, so let's take advantage of its modern build system!

    * Keep version *2018.01*

    * set *Custom U-Boot patches* to `board/pocketbeagle/patches/u-boot`

    * Use `am335x_pocketbeagle` as the *Board defconfig*

    * The *U-Boot binary format* should be changed from `u-boot.bin` to `u-boot.img`. Indeed, this second stage bootloader will be loaded by a first stage bootloader, and needs to have the proper header to be loaded by the first stage.

    * Enable *Install U-Boot SPL binary image* to also install the first stage bootloader. Its name in *U-Boot SPL/TPL binary image name(s)* should be changed to `MLO` since that's how U-Boot names it, and how the AM335x expects it to be named.

Running the build

To start the build, you can run just `make`. But it's often convenient to keep the build output in a log file, so you can do:
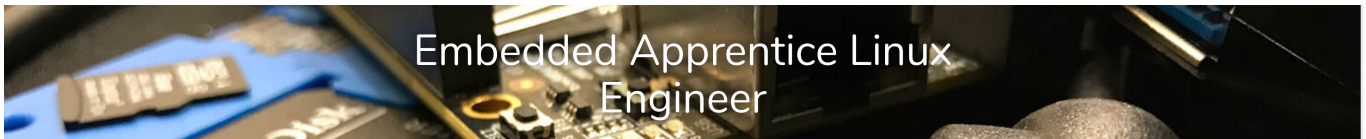
```
$ make 2>&1 | tee build.log
```

or alternatively use a wrapper script provided by Buildroot:

```
$ ./utils/brmake
```

The build will take a while.

The overall build takes quite some time, because the Linux kernel configuration `omap2plus_defconfig`, which supports all OMAP2, OMAP3, OMAP4 and AM335x platforms has a *lot* of drivers and options enabled. It would definitely be possible to make a smaller kernel configuration for the *Pocket Beagle*, reducing the kernel size and boot time.

At the end of the build, the output is located in `output/images`. We have:

- `MLO`, the first stage bootloader

- `u-boot.img`, the second stage bootloader

- `zImage`, the Linux kernel image

- `am335x-pocketbeagle.dtb`, the Linux kernel Device Tree Blob

- `rootfs.ext4`, the root filesystem image

However, that doesn't immediately give us a bootable SD card image. We could create it manually, but that wouldn't be really nice. So move on to the next section to see how Buildroot can create the SD card image for you.

Creating a SD card image

To create a SD card image, we'll use a tool called `genimage`, which provided a configuration file, will output the image of a block device, with multiple partitions, each containing a filesystem. See `https://git.pengutronix.de/cgit/genimage/tree/README.rst` for some documentation about *genimage* and its configuration file format.

`genimage` needs to be called at the very end of the build. To achieve this, Buildroot provides a mechanism called *post-image scripts*, which are arbitrary scripts called at the end of the build. We will use it to create a SD card image with:

- A FAT partition containing the bootloader images, the kernel image and Device Tree

- An ext4 partition containing the root filesystem

In addition, the U-Boot bootloader for the *PocketBeagle* is configured by default to load a file called `uEnv.txt` to indicate what should be done at boot time. This file should also be stored in the first partition of the SD card.

So, go back to `make menuconfig`, and adjust the following options:

- In *System configuration*
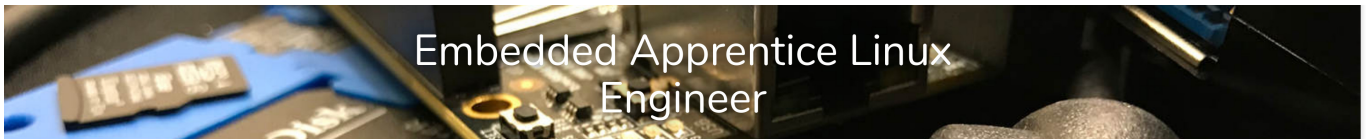
- – Set *Custom scripts to run after creating filesystem images* to `board/pocketbeagle/`
  `post-image.sh`

- In *Host utilities*

  - – enable

    * `host dosfstools`

    * `host genimage`

    * `host mtools`

    mtools and *dosfstools* are needed because our *genimage* configuration includes the creation of a FAT partition.

Restart the build again. Once the build is finished, you should now have a `sdcard.img` file in `output/images/`.

Storing our Buildroot configuration

Our Buildroot configuration is currently stored as `.config`, which is not under version control and would be removed by a `make distclean`. So, let's store it as a *defconfig* file:

`$ make BR2_DEFCONFIG=configs/eale_pocketbeagle_defconfig savedefconfig`

And then look at `configs/eale_pocketbeagle_defconfig` to see what your configuration looks like.

Testing The Build

Working through either lab, if successful, you should find your `*.img` file waiting for you at `output/sdcard.img`.

```
/home/trevor/Conference/SCaLE17x

File  Edit  View  Search  Terminal  Help
[trevor]$ ls -lh
total 114M
-rw-r--r-- 1 trevor trevor   33K Mar  9 10:49 am335x-pocketbeagle.dtb
-rw-r--r-- 1 trevor trevor   16M Mar  9 10:50 boot.vfat
-rw-r--r-- 1 trevor trevor   77K Mar  9 10:43 MLO
-rw-r--r-- 1 trevor trevor   60M Mar  9 10:50 rootfs.ext2
lrwxrwxrwx 1 trevor trevor    11 Mar  9 10:50 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 trevor trevor   13M Mar  9 10:50 rootfs.tar
-rw-r--r-- 1 trevor trevor   77M Mar  9 10:50 sdcard.img
-rw-r--r-- 1 trevor trevor  367K Mar  9 10:43 u-boot.img
-rwxr-xr-x 1 trevor trevor   76K Mar  9 10:43 u-boot-spl.bin
-rw-r--r-- 1 trevor trevor   218 Mar  9 10:50 uEnv.txt
-rw-r--r-- 1 trevor trevor  5.5M Mar  9 10:49 zImage
[trevor]$
```

Using **Etcher** or **dd**, flash this image to your SDcard. Insert the SDcard into your PocketBeagle and apply power. Ideally you'll have a serial console setup (i.e. **minicom** or **screen**) so you can watch the progress.

```
/home/trevor

File  Edit  View  Search  Terminal  Help
[    1.668836] mmc0: new high speed SDHC card at address 0007
[    1.677395] mmcblk0: mmc0:0007 SD16G 14.6 GiB
[    1.688578]  mmcblk0: p1 p2
[    1.706150] tps65217 0-0024: TPS65217 ID 0xe version 1.2
[    1.712616] omap_i2c 44e0b000.i2c: bus 0 rev0.11 at 400 kHz
[    1.747003] omap_i2c 4819c000.i2c: bus 2 rev0.11 at 400 kHz
[    1.754286] hctosys: unable to open rtc device (rtc0)
[    1.759744] sr_init: No PMIC hook to init smartreflex
[    1.765276] sr_init: platform driver register failed for SR
[    1.941988] EXT4-fs (mmcblk0p2): recovery complete
[    1.957898] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. )
[    1.966682] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    1.975783] devtmpfs: mounted
[    1.981117] Freeing unused kernel memory: 1024K
[    2.097564] EXT4-fs (mmcblk0p2): re-mounted. Opts: data=ordered
Starting logging: OK
Initializing random number generator... done.
Starting network: OK

Welcome to Buildroot
buildroot login: root
# uname -a
Linux buildroot 4.14.18 #1 SMP Sat Mar 9 10:47:28 PST 2019 armv7l GNU/Linux
#
```