Embedded Linux

LXE22109

Practical Labs

bootlin
https://bootlin.com

August 24, 2022

About this document

Updates to this document can be found on https://bootlin.com/doc/training/lxe22109-02.

This document was generated from LaTeX sources found on https://github.com/bootlin/training-materials.

More details about our training sessions can be found on https://bootlin.com/training.

Copying this document

Corrections, suggestions, contributions and translations are welcome!

Basic Buildroot usage

Objectives:

- *Get Buildroot*
- *Configure a minimal system with Buildroot for the BeagleBone Black*
- *Do the build*
- *Prepare the BeagleBone Black for usage*
- *Flash and test the generated system*

Setup

Go to the `$HOME/lxe22109-02-labs/` directory.

As specified in the Buildroot manual[1], Buildroot requires a few packages to be installed on your machine. Let's install them using Ubuntu's package manager:

```
sudo apt install sed make binutils gcc g++ bash patch \
  gzip bzip2 perl tar cpio python unzip rsync wget libncurses-dev
```

Download Buildroot

Since we're going to do Buildroot development, let's clone the Buildroot source code from its Git repository:

```
git clone https://git.buildroot.net/buildroot
```

Go into the newly created `buildroot` directory.

We're going to start a branch from the *2022.02* Buildroot release, with which this training has been tested.

```
git checkout -b lxe22109-02 2022.02
```

Configuring Buildroot

If you look under `configs/`, you will see that there is a file named `beaglebone_defconfig`, which is a ready-to-use Buildroot configuration file to build a system for the BeagleBone Black platform. However, since we want to learn about Buildroot, we'll start our own configuration from scratch!

Start the Buildroot configuration utility:

[1] https://buildroot.org/downloads/manual/manual.html#requirement-mandatory


```
make menuconfig
```

Of course, you're free to try out the other configuration utilities `nconfig`, `xconfig` or `gconfig`.

Now, let's do the configuration:

- `Target Options` menu
  - It is quite well known that the BeagleBone Black is an ARM based platform, so select `ARM (little endian)` as the target architecture.
  - According to the BeagleBone Black website at https://beagleboard.org/BLACK, it uses a Texas Instruments AM335x, which is based on the ARM Cortex-A8 core. So select `cortex-A8` as the `Target Architecture Variant`.
  - On ARM two *Application Binary Interfaces* are available: `EABI` and `EABIhf`. Unless you have backward compatibility concerns with pre-built binaries, `EABIhf` is more efficient, so make this choice as the `Target ABI` (which should already be the default anyway).
  - The other parameters can be left to their default value: `ELF` is the only available `Target Binary Format`, `VFPv3-D16` is a sane default for the *Floating Point Unit*, and using the `ARM` instruction set is also a good default (we could use the `Thumb-2` instruction set for slightly more compact code).
- We don't have anything special to change in the `Build options` menu, but take nonetheless this opportunity to visit this menu, and look at the available options. Each option has a help text that tells you more about the option.
- `Toolchain` menu
  - By default, Buildroot builds its own toolchain. This takes quite a bit of time, and for `ARMv7` platforms, there is a pre-built toolchain provided by ARM. We'll use it through the *external toolchain* mechanism of Buildroot. Select `External toolchain` as the `Toolchain type`. Do not hesitate however to look at the available options when you select `Buildroot toolchain` as the `Toolchain type`.
  - Select `Arm ARM 2021.07` as the `Toolchain`. Buildroot can either use pre-defined toolchains such as the ones provided by ARM, or custom toolchains (either downloaded from a given location, or pre-installed on your machine).
- `System configuration` menu
  - For our basic system, we don't need a lot of custom *system configuration* for the moment. So take some time to look at the available options, and put some custom values for the `System hostname`, `System banner` and `Root password`.
- `Kernel` menu
  - We obviously need a Linux kernel to run on our platform, so enable the `Linux kernel` option.
  - By default, the most recent Linux kernel version available at the time of the Buildroot release is used. In our case, we want to use a specific version: `5.15.35`. So select `Custom version` as the `Kernel version`, and enter `5.15.35` in the `Kernel version` text field that appears.
  - Now, we need to define which kernel configuration to use. We'll start by using a default configuration provided within the kernel sources themselves, called a *defconfig*. To identify which *defconfig* to use, you can look in the kernel sources directly, at

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/configs/?id=v5.15. In practice, for this platform, it is not trivial to find which one to use: the AM335x processor is supported in the Linux kernel as part of the support for many other Texas Instruments processors: OMAP2, OMAP3, OMAP4, etc. So the appropriate *defconfig* is named `omap2plus_defconfig`. You can open up this file in the Linux kernel Git repository viewer, and see it contains the line `CONFIG_SOC_AM33XX=y`, which is a good indication that it has the support for the processor used in the BeagleBone Black. Now that we have identified the *defconfig* name, enter `omap2plus` in the `Defconfig name` option.

  - The `Kernel binary format` is the next option. Since we are going to use a recent U-Boot bootloader, we'll keep the default of the `zImage` format.
  - On ARM, all modern platforms now use the *Device Tree* to describe the hardware. The BeagleBone Black is in this situation, so you'll have to enable the `Build a Device Tree Blob` option. At https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/boot/dts/?id=v5.15, you can see the list of all Device Tree files available in the 5.10 Linux kernel (note: the Device Tree files for boards use the `.dts` extension). The one for the BeagleBone Black is `am335x-boneblack.dts`. Even if talking about Device Tree is beyond the scope of this training, feel free to have a look at this file to see what it contains. Back in Buildroot, enable `Build a Device Tree Blob (DTB)` and type `am335x-boneblack` as the `In-tree Device Tree Source file names`.
  - The kernel configuration for this platform requires having OpenSSL available on the host machine. To avoid depending on the OpenSSL development files installed by your host machine Linux distribution, Buildroot can build its own version: just enable the `Needs host OpenSSL` option.

- `Target packages` menu. This is probably the most important menu, as this is the one where you can select amongst the 2800+ available Buildroot packages which ones should be built and installed in your system. For our basic system, enabling `BusyBox` is sufficient and is already enabled by default, but feel free to explore the available packages. We'll have the opportunity to enable some more packages in the next labs.
- `Filesystem images` menu. For now, keep only the `tar the root filesystem` option enabled. We'll take care separately of flashing the root filesystem on the SD card.
- `Bootloaders` menu.
  - We'll use the most popular ARM bootloader, *U-Boot*, so enable it in the configuration.
  - Select `Kconfig` as the `Build system`. U-Boot is transitioning from a situation where all the hardware platforms were described in C header files to a system where U-Boot re-uses the Linux kernel configuration logic. Since we are going to use a recent enough U-Boot version, we are going to use the latter, called *Kconfig*.
  - Use the custom version of U-Boot `2022.04`.
  - Look at https://gitlab.denx.de/u-boot/u-boot/-/tree/master/configs to identify the available U-Boot configurations. For many AM335x platforms, U-Boot has a single configuration called `am335x_evm_defconfig`, which can then be given the exact hardware platform to support using a Device Tree. So we need to use `am335x_evm` as `Board defconfig` and `DEVICE_TREE=am335x-boneblack` as `Custom make options`
  - U-Boot on AM335x is split in two parts: the first stage bootloader called `MLO` and the second stage bootloader called `u-boot.img`. So, select `u-boot.img` as the `U-`

Boot binary format, enable Install U-Boot SPL binary image and use MLO as the U-Boot SPL binary image name.

– The specific version of U-Boot that we selected requires having OpenSSL available on the host machine. To avoid depending on the OpenSSL development files installed by your host machine Linux distribution, Buildroot can build its own version: just enable the U-Boot needs OpenSSL option.

You're now done with the configuration!

Building

You could simply run make, but since we would like to keep a log of the build, we'll redirect both the standard and error outputs to a file, as well as the terminal by using the tee command:

```
make 2>&1 | tee build.log
```

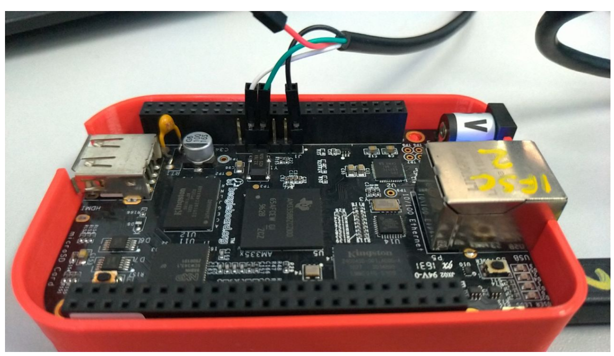
While the build is on-going, please go through the following sections to prepare what will be needed to test the build results.

Prepare the BeagleBone Black

The BeagleBone Black is powered via the USB-A to mini-USB cable, connected to the mini-USB connector labeled P4 on the back of the board.

The Beaglebone serial connector is exported on the 6 male pins close to one of the 48 pins headers. Using your special USB to Serial adapter provided by your instructor, connect the ground wire (black) to the pin closest to the power supply connector (let's call it pin 1), and the TX (green) and RX (white) wires to the pins 4 (board RX) and 5 (board TX)

You always should make sure that you connect the TX pin of the cable to the RX pin of the board, and vice-versa, whatever the board and cables that you use.



Once the USB to Serial connector is plugged in, a new serial port should appear: /dev/ttyUSB0. You can also see this device appear by looking at the output of dmesg.

To communicate with the board through the serial port, install a serial communication program, such as picocom:

```
sudo apt install picocom
```

If you run ls -l /dev/ttyUSB0, you can also see that only root and users belonging to the dialout group have read and write access to this file. Therefore, you need to add your user to the dialout group:


```
sudo adduser $USER dialout
```

Important: for the group change to be effective, in Ubuntu 18.04, you have to *completely reboot* the system [2]. A workaround is to run `newgrp dialout`, but it is not global. You have to run it in each terminal.

Now, you can run `picocom -b 115200 /dev/ttyUSB0`, to start serial communication on `/dev/ttyUSB0`, with a baudrate of `115200`. If you wish to exit `picocom`, press `[Ctrl][a]` followed by `[Ctrl][x]`.

There should be nothing on the serial line so far, as the board is not powered up yet.

Prepare the SD card

Our SD card needs to be split in two partitions:

- A first partition for the bootloader. It needs to comply with the requirements of the AM335x SoC so that it can find the bootloader in this partition. It should be a FAT32 partition. We will store the bootloader (`MLO` and `u-boot.img`), the kernel image (`zImage`)and the Device Tree (`am335x-boneblack.dtb`).
- A second partition for the root filesystem. It can use whichever filesystem type you want, but for our system, we'll use *ext4*.

First, let's identify under what name your SD card is identified in your system: look at the output of `cat /proc/partitions` and find your SD card. In general, if you use the internal SD card reader of a laptop, it will be `mmcblk0`, while if you use an external USB SD card reader, it will be `sdX` (i.e `sdb`, `sdc`, etc.). **Be careful: `/dev/sda` is generally the hard drive of your machine!**

If your SD card is `/dev/mmcblk0`, then the partitions inside the SD card are named `/dev/mmcblk0p1`, `/dev/mmcblk0p2`, etc. If your SD card is `/dev/sdc`, then the partitions inside are named `/dev/sdc1`, `/dev/sdc2`, etc.

To format your SD card, do the following steps:

1. Unmount all partitions of your SD card (they are generally automatically mounted by Ubuntu)
2. Erase the beginning of the SD card to ensure that the existing partitions are not going to be mistakenly detected:
   `sudo dd if=/dev/zero of=/dev/mmcblk0 bs=1M count=16`. Use `sdc` or `sdb` instead of `mmcblk0` if needed.
3. Create the two partitions.
   - Start the `cfdisk` tool for that:
     `sudo cfdisk /dev/mmcblk0`
   - Chose the *dos* partition table type
   - Create a first small partition (128 MB), primary, with type `e` (*W95 FAT16*) and mark it bootable
   - Create a second partition, also primary, with the rest of the available space, with type `83` (*Linux*).
   - Exit `cfdisk`

[2] As explained on https://askubuntu.com/questions/1045993/after-adding-a-group-logoutlogin-is-not-enough-in-18-04/.

4. Format the first partition as a *FAT32* filesystem:
   `sudo mkfs.vfat -F 32 -n boot /dev/mmcblk0p1`. Use `sdc1` or `sdb1` instead of `mmcblk0p1` if needed.
5. Format the second partition as an *ext4* filesystem:
   `sudo mkfs.ext4 -L rootfs -E nodiscard /dev/mmcblk0p2`. Use `sdc2` or `sdb2` instead of `mmcblk0p2` if needed.
   - `-L` assigns a volume name to the partition
   - `-E nodiscard` disables bad block discarding. While this should be a useful option for cards with bad blocks, skipping this step saves long minutes in SD cards.

Remove the SD card and insert it again, the two partitions should be mounted automatically, in `/media/$USER/boot` and `/media/$USER/rootfs`.

Now everything should be ready. Hopefully by that time the Buildroot build should have completed. If not, wait a little bit more.

Flash the system

Once Buildroot has finished building the system, it's time to put it on the SD card:

- Copy the `MLO`, `u-boot.img`, `zImage` and `am335x-boneblack.dtb` files from `output/images/` to the `boot` partition of the SD card.
- Extract the `rootfs.tar` file to the `rootfs` partition of the SD card, using:
  `sudo tar -C /media/$USER/rootfs/ -xf output/images/rootfs.tar`.
- Create a file named `extlinux/extlinux.conf` in the `boot` partition. This file should contain the following lines:

  ```
  label buildroot
    kernel /zImage
    devicetree /am335x-boneblack.dtb
    append console=ttyO0,115200 root=/dev/mmcblk0p2 rootwait
  ```

  These lines teach the U-Boot bootloader how to load the Linux kernel image and the Device Tree, before booting the kernel. It uses a standard U-Boot mechanism called *distro boot command*, see https://source.denx.de/u-boot/u-boot/-/raw/master/doc/README.distro for more details.

Cleanly unmount the two SD card partitions, and eject the SD card.

Boot the system

Insert the SD card in the BeagleBone Black. Push the S2 button (located near the USB host connector) and plug the USB power cable while holding S2. Pushing S2 forces the BeagleBone Black to boot from the SD card instead of from the internal eMMC.

You should see your system booting. Make sure that the U-Boot SPL and U-Boot version and build dates match with the current date. Do the same check for the Linux kernel.

Login as `root` on the BeagleBone Black, and explore the system. Run `ps` to see which processes are running, and look at what Buildroot has generated in `/bin`, `/lib`, `/usr` and `/etc`.

Note: if your system doesn't boot as expected, make sure to reset the U-Boot environment by running the following U-Boot commands:


```
env default -f -a
saveenv
```

and reset. This is needed because the U-Boot loaded from the SD card still loads the U-Boot environment from the eMMC. Ask your instructor for additional clarifications if needed.

Explore the build log

Back to your build machine, since we redirected the build output to a file called `build.log`, we can now have a look at it to see what happened. Since the Buildroot build is quite verbose, Buildroot prints before each important step a message prefixed by the `>>>` sign. So to get an overall idea of what the build did, you can run:

```
grep ">>>" build.log
```

You see the different packages between downloaded, extracted, patched, configured, built and installed.

Feel free to explore the `output/` directory as well.