The Embedded Linux Quick Start Guide

Workbook

Version 1.4

May 2016

0 Preparation

These notes are to help you get up and running with embedded Linux on the BeagleBone Black.

At this point, you should have

- A laptop or PC running Ubuntu 14.04 (*). Ideally, it should have a quad core processor and at least 4 GiB or RAM
- A BeagleBone Black board and the mini USB A/B to USB A cable supplied with the board
- An RS-232 serial to USB cable such as the FTDI TTL-232R-3V3 available from Digi-Key, Element 14 and others.
- A micro SD card of at least 2 GB and a card reader
- An Ethernet cross-over cable, available from many places

(*) The exercises were tested using Ubuntu 14.04, and some of the instructions are specific to 14.04, such as the list of packages to install, but apart from that you could use any recent Linux distribution with a little adaptation. You can use a virtual machine such as VmWare or Virtual Box so long as you provision it appropriately.

0.1 Get the sample files

There are some files you will need to complete the exercises. In the exercises I have assumed that they are installed into directory `~/embedded-linux`. You may, of course, place them somewhere else, but in that case you will need to adjust the paths later on. Open a terminal and enter these commands:

```
$ cd ~
$ git clone https://github.com/csimmonds/embedded-linux-quick-start-files.git embedded-linux
```

1 Buildroot

Objective

In this exercise you will get a copy of Buildroot and use it to generate the four elements of embedded Linux: toolchain, bootloader, kernel and root filesystem.

1.1 Install packages

Buildroot requires some packages to be installed. The Buildroot manual https://buildroot.org/downloads/manual/manual.html has full details. If you are using Ubuntu 14.04, you can install them from the command-line like so:

```
$ sudo apt-get install binutils build-essential bzip2 chrpath cpio g++ \
 git gzip libncurses5-dev libsdl1.2-dev patch perl python rsync sed \
 texinfo unzip util-linux wget
```

There are another couple of packages that you will need: the terminal emulator, gtkterm, and the util-linux package which contains the lsblk command that you will use in the next exercise

```
$ sudo apt-get install gtkterm util-linux
```

1.2 Get Buildroot

Download buildroot-2016-02.tar.bz2 from buildroot.org and extract it like so:

```
$ cd ~
$ tar xf ~/Downloads/buildroot-2016.02.tar.bz2
$ cd buildroot-2016.02
```

Alternatively, you can clone the Buildroot git archive:

```
$ cd ~
$ git clone git://git.buildroot.net/buildroot
$ cd buildroot
$ git checkout 2016.02
```

1.3 Configure for BeagleBone

Select the default configuration for the target board (look in directory "`configs/`" to see all the configurations files):

```
$ make beaglebone_defconfig
```

Next, change the C library to glibc, add an ssh server called Dropbear and set the root filesystem format to a tar file which you can extract onto an SD card later on:

```
$ make menuconfig
```

- In **Toolchain -> C library**, select **glibc**
- In **Target packages->Networking applications** enable **dropbear**
- In **Filesystem images** select **tar**, leaving the tar options with default settings

Exit and save the changes

1.4 Build

On a reasonable spec PC/laptop (quad core, with 4 GiB RAM) the build this takes less than an hour. It will download 250 MiB of source code and will consume 5 GiB of disk space:

```
$ make
```

When complete, you will see that there is a cross toolchain in `output/host/usr/bin/`, and the files for the target are in `output/images/`.

2 Boot

Objective

Boot the BeagleBone Black from a microSD card using the images created by Buildroot

2.1 Format the uSD card

There are two possibilities: you may be using an external USB uSD card reader, or you may be using an internal SD card reader that is present on many laptops. The command `lsblk` is a useful tool to find out which device name the SD card has been allocated. In these two examples I am logged on as user *training* and the uSD card is a nominal 8 GB part.

External card reader

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   477G  0 disk
├─sda1   8:1    0   500M  0 part /boot/efi
├─sda2   8:4    0 457.6G  0 part /
└─sda5   8:5    0  15.8G  0 part [SWAP]
sdb      8:16   1   7.2G  0 disk
└─sdb1   8:17   1   7.2G  0 part /media/training/2537-B714
```

In this case, it is sdb, and it has one partition, sdb1

Internal card reader

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   477G  0 disk
├─sda1   8:1    0   500M  0 part /boot/efi
├─sda2   8:4    0 457.6G  0 part /
└─sda5   8:5    0  15.8G  0 part [SWAP]
mmcblk0     179:0    0   7.2G  0 disk
└─mmcblk0p1 179:1    0   7.2G  0 part /media/training/2537-B714
```

In this case, it appears as mmcblk0 and the partition is mmcblk0p1

Note that the SD card may have been formatted already, in which case you may see a different number of partitions with different names.

Use the script `format-sdcard.sh` to format it, with the device name as a parameter, for example

```
$ ~/embedded-linux/format-sdcard.sh sdb
```

This creates two partitions: the first is 67 MiB, formatted as FAT32, and will contain the bootloader and kernel, the second is 89 MiB, formatted as ext4, and will contain the root filesystem.

Remove and re-insert the SD card and check that the partitions are mounted, and they are (almost) empty:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G  4.0K  3.9G   1% /dev
tmpfs           790M  1.6M  788M   1% /run
/dev/sda2       451G  358G   70G  84% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
none            5.0M  4.0K  5.0M   1% /run/lock
none            3.9G  178M  3.7G   5% /run/shm
none            100M   48K  100M   1% /run/user
/dev/sda1       496M   19M  478M   4% /boot/efi
/dev/sdb1        67M  468K   66M   1% /media/training/boot
/dev/sdb2        89M  1.6M   81M   2% /media/training/rootfs
```

2.2 Copy files to the uSD card

The bootloader files

```
$ cd ~/buildroot-2016.02/output/images
$ cp MLO u-boot.img uEnv.txt /media/training/boot
```

The kernel and device tree files

```
$ cp zImage am335x-boneblack.dtb /media/training/boot
```

The root file system

```
$ sudo tar xf rootfs.tar -C /media/training/rootfs
```

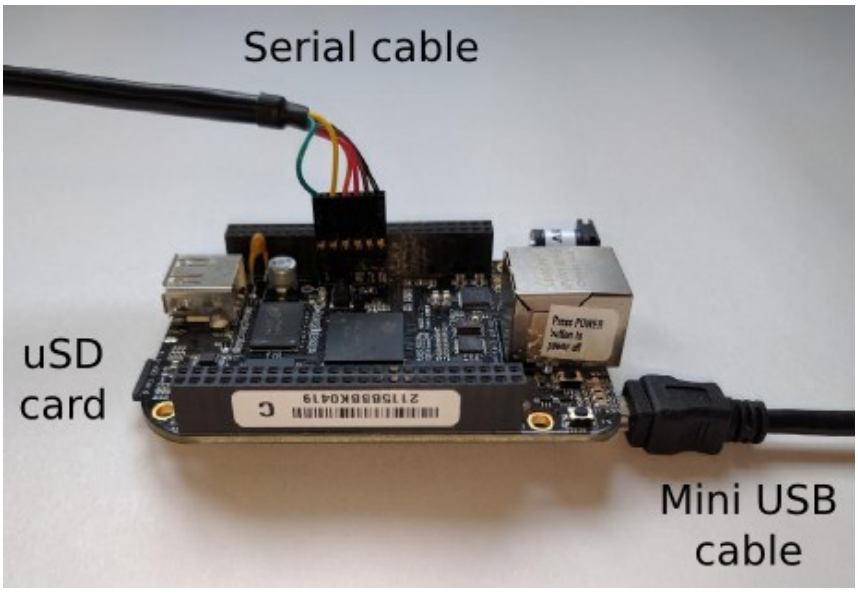
Unmount the partitions

```
$ sudo umount /media/training/rootfs
$ sudo umount /media/training/boot
```

Take out the uSD card and put it into the target BeagleBone Black

2.3 Set up the terminal emulator for the serial console

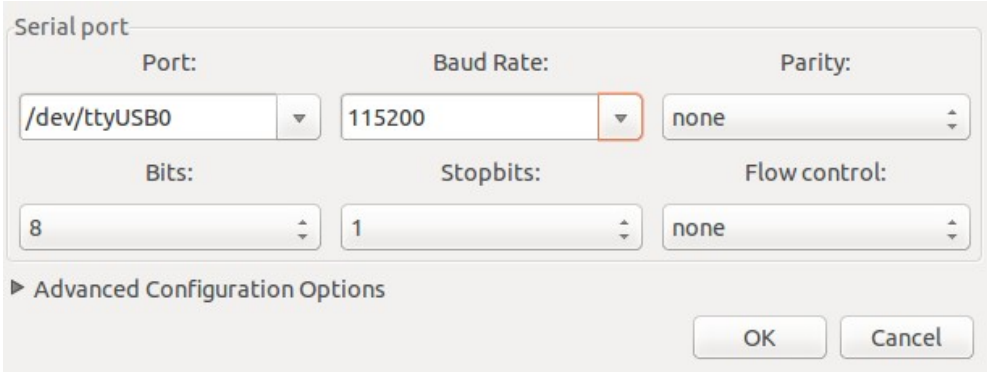
Plug the serial-to-USB cable (the long thick one) into the 4-pin header on the BeagelBone Black and into your laptop/PC but **do not power on the BBB yet**.



Launch gtkterm:

```
$ gtkterm
```

Go to **Configuration->Port** and change the settings as follows:



Save it as the default configuration: Configuration->Save configuration

2.4 Power up the BeagleBone Black

Make sure that the uSD card is plugged in to the BBB.

Press and hold the boot button to force boot from the uSD card and not the internal flash. You may release the button once you see text appearing on the serial terminal.

Plug the micro USB cable into the BBB and the other end to your laptop/PC. This will supply power to the BeaglelBone Black. You should see something like this:

```
U-Boot SPL 2016.03 (Apr 14 2016 - 10:00:18)
Trying to boot from MMC
reading args
spl_load_image_fat_os: error reading image args, err - -1
reading u-boot.img
reading u-boot.img


U-Boot 2016.03 (Apr 14 2016 - 10:00:18 +0100)
```

Finally, you should see:

```
Starting logging: OK
Initializing random number generator... done.
Starting network...
Starting dropbear sshd: OK
[    2.426763] PM: CM3 Firmware Version = 0x185

Welcome to Buildroot
beaglebone login:
```

2.5 Try it out

Log on as root, no password.

Have a look round, for example:

- What is the kernel command line? (`cat /proc/cmdline`)
- What type of CPU? (`cat /proc/cpuinfo`)
- What processes are running? (`ps`)
- How much free memory? (`free`)
- How much disk storage is used? (`df -h`)
- What type of file systems are mounted? (`cat /proc/mounts`)
- What kernel version (`uname -a`)

Note: the kernel that is built by Buildroot is version 3.14. It does not include the cape manager and so does not support device tree overlays. Which means that some capes will not work with this setup unless you spend some time editing the device tree and adjusting the kernel configuration. If these things are important to you, then you would be better served by using the stock Debian Linux images for the BeagleBone Black. The aim of these exercise is to help you understand the process of building an embedded Linux distribution rather than building something that is specifically tailored to one platform.

3 User and network configuration

Objective

To create a user account and set up the target with a static network address.

3.1 Make the root filesystem writeable

You will find that the root filesystem has been mounted read-only, and so you cannot make any changes. Therefore, the first thing to do is to make root writeable by remounting it. Enter this command on the target:

```
# mount -o remount,rw /
```

3.2 Set a password for the root user

It would be nice if there was a password for the super-user (root) account. The `passwd` command allows you to change a user's password:

```
# passwd
Changing password for root
New password:
Retype password:
Password for root changed by root
```

Log out, by typing **exit** or pressing Ctrl-D, and log in again using the new password.

3.3 Create a user account

Use the **`id`** command to find the user and group id of the root user.

Now, create a new user:

```
# mkdir /home
# adduser chris
Changing password for chris
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
```

You can see that the user name has been added to `/etc/passwd` and the group to `/etc/group`.

Log out and log in again as the new user, and use **id** again to show the new user and group ids.

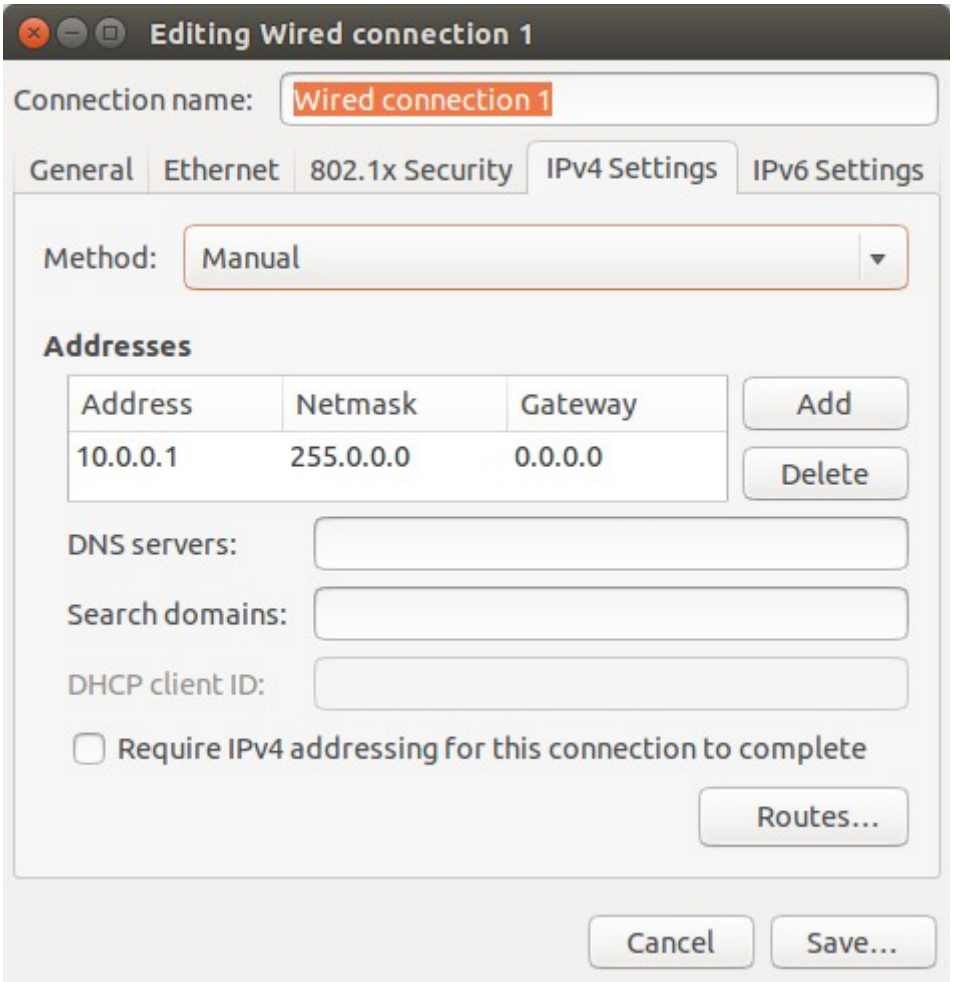
3.4 Network configuration on the laptop

In this exercise you will create a simple two node network between your laptop/PC and the target board.

Plug in the Ethernet loopback cable between the BeagleBone Black and your laptop/PC.

Set the IP address of the Ethernet port on your PC to 10.0.0.1 as follows (instructions are for Ubuntu 14.04):

- Click on the network icon in the menu bar, top right
- Select **Edit connections...**
- On the Wired tab, select **Wired connection1** and click Edit...
- On the **Ipv4 Settings** tab, change Method from Automatic (DHCP) to Manual
- Click the **Add** button to add a static address
- Enter Address 10.0.0.1, Netmask 255.0.0.0, then click on Gateway *but do not enter an address*
- Click the **Save...** button and then close the Network Connection dialog box



3.5 Network configuration on the target

Create a static IP address for the Ethernet port by using vi on the target to edit `/etc/network/interfaces` If you are not familiar with vi, there is a short beginners guide at http://2net.co.uk/downloads/vi-for-beginners.pdf.

Add these lines at the bottom of the file:

```
auto eth0
iface eth0 inet static
      address 10.0.0.2
      netmask 255.255.255.0
```

Now bring up the network:

```
# ifup -a
```

Check that the Ethernet port is now configured using command **ifconfig** (on the target):

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 1C:BA:8C:E9:11:DA
          inet addr:10.0.0.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:77 errors:0 dropped:0 overruns:0 frame:0
          TX packets:47 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16144 (15.7 KiB)  TX bytes:6390 (6.2 KiB)
          Interrupt:56
```

Check that you can ping the laptop from the target and visa versa

Check that you can use ssh (on the host) to log on to the target:

```
$ ssh root@10.0.0.2
```

4 Buildroot

Objective

To experiment with Buildroot.

- Make the root filesystem writeable
- Make the network configuration permanent
- Add a package – a web server – to the root file system

4.1 Writeable root filesystem

Begin by launching the Buildroot configuration menu:

```
$ cd ~/buildroot-2016.02
$ make menuconfig
```

In **System configuration**, enable **remount root filesystem read-write during boot**

Exit and save the configuration

4.2 Network configuration

A simple way to customise the root filesystem image is to create an overlay directory. The files in the overlay are copied into the final image as the last part of the build process. The configuration for the BeagelBone is in `board/beagelbone`, so this is a natural place to put the overlay directory:

```
$ cd ~/buildroot-2016.02/board/beaglebone
$ mkdir -p overlay/etc/network
```

Copy the interfaces file into the overlay:

```
$ scp root@10.0.0.2:/etc/network/interfaces overlay/etc/network
```

Now, configure Buildroot to use the overlay, by running `make menuconfig`.

In **System configuration**, select **Root filesystem overlay directories** and enter

board/beaglebone/overlay

Exit and save the configuration

4.3 Adding a web server

Run `make menuconfig`

In **Target packages->Networking applications** enable **tinyhttpd**

Exit and save the new configuration.

The web pages for tinyhttps are stored in `/var/www`. You can add one to the overlay:

```
$ cd ~/buildroot-2016.02/board/beaglebone
$ mkdir -p overlay/var/www
$ cp ~/embedded-linux/index.html overlay/var/www
```

4.4 Build new system images

```
$ cd ~/buildroot-2016.02
$ make
```

The build should only take a minute or two. The end result is in `output/images`.

4.5 Re-flashing the new root filesystem

Power off the BeagelBone Black and plug the uSD card into the card reader.

Re flash the the uSD card as you did in exercises 2.1 and 2.2.

Boot up the BeagelBone and check that

1. The root filesystem is writeable
2. The Ethernet device eth0 is configured (by running **ifconfig**)
3. The web server is running

    `ps ax` shows tinyhttpd

    `netstat -ltnp` shows a server listening on port 80

You should be able to load the page in a web browser using FireFox on the host with URL 10.0.0.2/

Note that the password for root has been reset to the default, empty password. Run the **passwd** command again to change it (this is necessary because you will need to use ssh later on)

5 Cross compiling

Objective

To cross-compile a program and run it on the target using the toolchain created by Buildroot

5.1 Finding out about the cross toolchain

First, it is convenient to add the toolchain binaries to the path. Enter this line in a shell on the host::

```
PATH=/home/training/buildroot-2016.02/output/host/usr/bin:$PATH
```

Note that this changes the PATH in the current shell only; if you want to make the change permanent, add that line to the bottom of ~/.bashrc.

The toolchain prefix is `arm-linux-`. Find the version of GCC you are using:

```
$ arm-linux-gcc -version
```

When compiling, the toolchain searches for header and library files relative to the **sysroot**. You can find the sysroot like this:

```
$ arm-linux-gcc -print-sysroot
```

Let''s keep a copy of that path in a shell variable to make the next few lines easier to read, and to type

```
$ export SYSROOT=$(arm-linux-gcc -print-sysroot)
```

Now you can find the location of the header files like this:

```
$ ls $SYSROOT/usr/include
```

And you can find the library files like this:

```
$ ls $SYSROOT/lib
$ ls $SYSROOT/usr/lib
```

5.2 Hello world!

Write a hello world program and compile it

```
$ arm-linux-gcc helloworld.c -o helloworld
```

You can confirm that it is compiled for an ARM target using the file command:

```
$ file helloworld
helloworld: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32, not stripped
```

5.3 Run the program on the target

To test your program you will have to copy it to the target. There are two ways to do that. You could power off the target, take out the uSD card, plug it into the reader and write the compiled binary to the /usr/bin directory on the second partition, and then boot up the target again.

A faster method is to copy it using scp:

```
$ scp helloworld root@10.0.0.2:/usr/bin
```

6 Device tree: stop those LEDs flashing!

The four user LEDs on the target are configured to trigger on certain events. By changing the device tree, you can change the triggers.

The kernel source code is in:

`~/buildroot-2016.02/board/beaglebone/output/build/linux-7f280334068b7c875ade51f8f3921ab311f0c824/`

In that directory, edit `arch/arm/boot/dts/am335x-bone-common.dtsi`

Look for this:

```
led@2 {
                        label = "beaglebone:green:heartbeat";
                        gpios = <&gpio1 21 GPIO_ACTIVE_HIGH>;
                        linux,default-trigger = "heartbeat";
                        default-state = "off";
                };
```

Delete the **default-trigger** line completely. Repeat for led@3, led@4, and led@5.

Re-build the device tree binary (.dtb):

```
$ cd ~/buildroot-2016.02
$ make linux-rebuild
```

Power off the BeagleBone, take out the uSD card and copy `output/images/am335x-boneblack.dtb` to the boot partition.

Boot up the target, and check that the user LEDs no longer flash.

7 Input and output

Objective

To write a program that will interface with the outside world: in this case flashing the user LEDs

7.1 Make those LEDs flash again

The LEDs are represented by these four directories

```
/sys/class/leds/beaglebone:green:heartbeat
/sys/class/leds/beaglebone:green:mmc0
/sys/class/leds/beaglebone:green:usr2
/sys/class/leds/beaglebone:green:usr3
```

Each directory contains a file named **brightness**: writing the string “255” tunes it on and the string “0” turns it off.

Write a program with this loop:

- Write the string “0” to the brightness file
- Sleep for 500 ms (usleep(500000))
- Write the string “255” to the brightness file
- Sleep for 500 ms

Cross compile it and load it into the target as you did in lab 5

There is a worked solution in ~/embedded-linux/led-flash

8 Flash file systems

Objective

To use the eMMC flash memory on the target board to store the root file system.

8.1 The easy way

Copy the disk image from the uSD card to eMMC:

```
# dd if=/dev/mmcblk0 of=/dev/mmcblk1 bs=1M count=196
```

Reboot, pressing the boot button to that it boots from uSD card still.

```
# mount /dev/mmcblk1p1 /mnt
```

Edit `/mnt/uEnv.txt`, and change

```
bootpart=1:1
bootdir=
uenvcmd=run loadimage;run loadramdisk;run findfdt;run loadfdt;run ramboot
```

Change bootpart to 1:1 as shown above.

```
# unmount /mnt
```

Power off, remove the uSD card

Power on, and check that it boots from eMMC now

8.2 The correct way

It is inherently risky to copy a live filesystem from one device to another, as described in the last exercise. The main issue is that any recent file writes will be missed, which is especially a problem for files containing structured data such as databases. In this simple case, there is very little file writing going on and so it works fine.

It would be more robust to install file system tools into the target image and use them to create a partition table, format the partitions, and then untar rootfs.tar to eMMC, handling any errors that may occur.

However, that is more complex than we can handle in a short workshop such as this.