## Contents



# Compiling the BeagleBone Black Kernel

The following contains instructions for building the BeagleBone Black kernel on Ubuntu 15.04.

# Prerequisites

## ARM Cross Compiler

To compile the linux kernel for the BeagleBone Black, you must first have an ARM cross compiler installed. I use gcc version 4.7.4 that ships with Ubuntu 15.04. To install the compiler run:

```
sudo apt-get install gcc-arm-linux-gnueabi
```

## GIT

The Beaglebone patches and build scripts are stored in a git repository. Install git:

```
sudo apt-get install git
```

And configure with your identity.

```
git config --global user.email "your.email@here.com"
```

## lzop Compression

The Linux Kernel is compressed using lzo. Install the lzop parallel file compressor:

```
sudo apt-get install lzop
```

## uBoot mkimage

The bootloader used on the BeagleBone black is U-Boot. U-Boot has a special image format called uImage. It includes parameters such as descriptions, the machine/architecture type, compression type, load address, checksums etc. To make these images, you need to have a mkimage tool that comes part of the U-Boot distribution.

Building U-Boot requires libssl-dev to be installed:

```
sudo apt-get install libssl-dev
```

Download U-Boot, make and install the U-Boot tools:

```
wget ftp://ftp.denx.de/pub/u-boot/u-boot-latest.tar.bz2
tar -xjf u-boot-latest.tar.bz2
cd into u-boot directory
make sandbox_defconfig tools-only
sudo install tools/mkimage /usr/local/bin
```

# Compiling the BeagleBone Black Kernel

Here we compile the BeagleBone Black Kernel, and generate an uImage file with a DTB blob:

```
git clone git://github.com/beagleboard/linux.git
cd linux
git checkout 4.1
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- bb.org_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j4
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage dtbs LOADADDR=0x80008000 -j4
```

Now we build any kernel modules:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules -j4
```

And if you have your rootfs ready, you can install them:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=/home/cpeacock/export/rootfs modules_install
```

# Testing

## TFTP Server

Rather than flashing your newly created kernel to find out it doesn't work or it is not quite configured correctly, a better way is to load the kernel into RAM and boot it from there. U-Boot allows kernel images to be loaded via TFTP.

To speed up development, I create an 'export' directory. A TFTP server is then configured to use this directory as the TFTP root.

From the export directory, add symbolic links to the kernel images. This way, you can recompile the Kernel and the new image is instantly available without having to move it.

```
ln -s /path to linux/arch/arm/boot/uImage uImage-BBB
ln -s /path to linux/arch/arm/boot/dts/am335x-boneblack.dtb am335x-boneblack.dtb
```

## U-Boot tftpboot

To test your kernel, bring up a serial console to the BeagleBone Black. First we will need to configure the IP addresses. The ipaddr variable contains the IP address for the BeagleBone Black, while the serverip variable is the address of the TFTP server containing the kernel image.

```
setenv ipaddr 192.168.0.250
setenv serverip 192.168.0.251
```

These variables can be saved to non-volatile memory to speed up development.

Next load the Kernel image and device tree binary blob. The kernel image is prefixed with a 64 byte MKIMAGE header. To take advantage of eXecute in place (XIP), we load the image at 0x80007FC0 (0x800080000 - 64 bytes) so the kernel is loaded at 0x80008000:

```
tftpboot 0x80F80000 am335x-boneblack.dtb
tftpboot 0x80007FC0 uImage-BBB
```

Let's use the existing root filesystem, and send kernel messages to ttyO0:

```
setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait
```

And boot from the memory location:

```
bootm 0x80007FC0 - 0x80F80000
```

Your BeagleBone Black should now boot your new Kernel:

```
## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-4.1.3
   Created:      2015-07-25  13:16:03 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    8068232 Bytes = 7.7 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 80f80000
   Booting using the fdt blob at 0x80f80000
   XIP Kernel Image ... OK
   Loading Device Tree to 8ffee000, end 8ffff0ec ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.1.3 (cpeacock@ubuntu) (gcc version 4.7.4 (Ubuntu/Linaro 4.7.4-2ubuntu1) ) #3 SMP PREEMPT Sat Jul 25
[    0.000000] CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: TI AM335x BeagleBone Black
[    0.000000] cma: Reserved 24 MiB at 0x9e000000
[    0.000000] Memory policy: Data cache writeback
[    0.000000] CPU: All CPU(s) started in SVC mode.
[    0.000000] AM335X ES2.0 (sgx neon )
[    0.000000] PERCPU: Embedded 13 pages/cpu @df927000 s22336 r8192 d22720 u53248
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129408
[    0.000000] Kernel command line: console=ttyO0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait
```

The example shown is to use the filesystem already in place (i.e. ext4 on mmcblk0p2). The preferred option is to export a root NFS filesystem.

# Booting from NFS

Booting from NFS has the added advantage that you can compile userland binaries on your development box, install them to the NFS export and have instant access to them on your target system.

This requires a root filesystem to be present on your development box with binaries that have been cross compiled for ARM. Various distributions exist to assist you with this. It is assumed a NFS Server is installed and configured properly to export this directory.

Angstrom BeagleBone demo files can be downloaded from here

Download the raw filesystem and decompress:

```
mkdir rootfs
wget http://downloads.angstrom-distribution.org/demo/beaglebone/Angstrom-systemd-image-eglibc-ipk-v2012.12-beaglebone-2013.09.12.r
tar -xJf Angstrom-systemd-image-eglibc-ipk-v2012.12-beaglebone-2013.09.12.rootfs.tar.xz -C rootfs
```

At this point you will need to ensure your Kernel modules are present. If not, run the following from your Kernel folder to install the modules on your rootfs:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=/home/cpeacock/export/rootfs modules_install
```

In U-Boot, you can now issue the following commands to load your Linux Kernel from TFTP and your rootfs over NFS:

```
setenv ipaddr 192.168.0.250
setenv serverip 192.168.0.251
tftpboot 0x80F80000 am335x-boneblack.dtb
tftpboot 0x80007FC0 uImage-BBB
setenv bootargs console=ttyO0,115200n8 root=/dev/nfs rw nfsroot=192.168.0.251:/home/cpeacock/export/rootfs ip=192.168.0.250:::::eth
bootm 0x80007FC0 - 0x80F80000
```

## NFS boot failures

If your NFS root freezes during boot with a message similar to:

```
nfs: server 192.168.0.251 not responding, still trying
```

Then it is possible the network connection used by the NFS connection has been re-configured. On Angstrom, this connection is re-established by the Connection Manager (ConnMan.) To quickly overcome this problem, remove /lib/systemd/system /connman.service from your rootFS.