Compiling and Deploying BeagleBone Black Kernel

LinuxBeagle Bone

It’s been a while since I wrote my first article and though I need to come up with something a little more advanced this time, something which will be perfect for the die-hard Linux users and those who await to get their hands dirty with some kernel grease.

For the sake of this discussion,it is assumed that you have some working knowledge of the Linux operating systems (at least as a user). Needless to say you should have a working bone to follow this tutorial.

BeagleBone Black

BeagleBone Black (BBB), is a popular Single Board Computer (SBC) which was released as a successor to the BeagleBone or BeagleBone White. If you don’t have a BBB, just order one to dive into the world of Embedded Linux. I’m sure that BBB will occupy a special place in your electronics hardware inventory :-).

Why custom kernel deploying?

Well, I know that this question will be itching your mind. Instead of using the pre-built image, why should we use this method of building our own image and RFS? The answer for this question is, you have to do this in order to get some fun out of BeagleBone. Electronics is fun when you start doing things of your own and also you will learn a lot of things while doing this.

For starters, I would strongly recommend to use the pre-built image for working with BeagleBone. But, as I already stated, this post is for the intermediate level users of BBB. This will be cool when you do this and I’m sure this will guide you into the real world of Embedded Linux.

Tools needed to get started

For building linux kernel you will need several tools other than BeagleBone. The tools which are required is listed below:

- A PC/Laptop with a flavor of Linux (Preferably Ubuntu 14.04)
- Linux kernel source for BeagleBone Black
- ARM cross compiler
- U-Boot(optional)
- mkimage

1. Installing ARM cross compiler

The first and foremost thing in compiling kernel is installing ARM gcc cross compiler. Since, BeagleBone Black is based on AM335x Sitara Processor, we need to compile the kernel for that environment. There are numerous compilers available online for free but it is important to install a stable one for proper compilation. For instance gcc-arm-linux-gnueabihf compiler available in standard Ubuntu package is an unstable one. So, download a stable compiler. The preferred one is Linaro cross compiler.

You can download the compiler here

After downloading, extract the compiler using the following command.

```
$ sudo tar xvf gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux.tar.xz -C /opt/
```

The compiler will be extracted to /opt/ directory. opt is nothing but the optional directory. Next, step is to add the compiler to the PATH variable, in order to direct the shell to find our compiler.

Go to /opt/ directory and change the directory name for adaptivity. Then, add the compiler to PATH variable.

```
$ cd /opt/
$ sudo mv gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/ gcc-arm-linux
$ export PATH=$PATH:/opt/gcc-arm-linux/bin
```

After installing the compiler you can verify it using the following command,

```
$ arm-linux-gnueabihf-gcc --version
arm-linux-gnueabihf-gcc (crosstool-NG linaro-1.13.1-4.8-2014.04 - Linaro GCC 4.8-2014.04) 4.8.3 20140401
(prerelease)
```

2. Cloning the Kernel

After installing the compiler, clone the kernel source for BeagleBone Black from GitHub using

```
$ git clone https://github.com/beagleboard/linux.git
```

Go to the linux directory and ensure that you have cloned the correct repo by executing the following command

```
$ cd linux
$ git remote -v
origin https://github.com/beagleboard/linux.git (fetch)
origin https://github.com/beagleboard/linux.git (push)
```

3. Cloning and Compiling U-boot(Optional)

U-boot is an open source universal bootloader for Linux systems. It supports features like TFTP, DHCP, NFS etc… In order to boot the kernel, a valid kernel image (uImage) is required. It is not possible to explain u-boot here, as it is beyond the scope of this post. So, we will see how to produce a bootable image using U-boot.

Clone u-boot using the following command

```
$ git clone git://git.denx.de/u-boot.git u-boot/
```

Before, compiling U-Boot we need to configure it. Thanks to the availability of configuration files in the configs/ directory under u-boot. We can configure using the am332x_boneblack_defconfig file. All the configuration will be written to .config file located in u-boot/ directory. By default you will not be able to view the .config file. To view give ls -a command.

```
$ cd u-boot
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am335x_boneblack_defconfig
```

After configuring, u-boot can be cross compiled using the following command.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

It will take around 10 to 15 minutes depending on the system configuration. Mine is Pentium dual core processor and it took 10 minutes for compilation. After successful compilation, several files will be produced in u-boot/ directory. Our prime concern is MLO and u-boot.img files.

For now, we will not use the above mentioned files for booting. But, during later stages those will be needed.

4. Formatting SD card

For deploying kernel from sd card, we need to format it and place the files accordingly. For this process, “Gparted” tool is needed. Install Gparted by the following command.

```
$ sudo apt-get install gparted
```

Insert your sd card by means of card reader and open Gparted. Select your sd card from the top right corner. it will be something like this, “/dev/sdb”

Note: Always use sd card of size greater than 2 GB. Although, 500 MB is more than enough for our task, having large free space will come handy at times.

Right click on the rectangular area showing your sd card name and select unmount as we need to unmount the existing partitions. Then, delete the existing partitions by again right clicking and selecting “delete”. This will delete all your files in sd card, so make sure you backed up any important files. We need two partitions in order to boot the kernel, one is for storing the bootable images and another one is for storing the minimal RFS(Root File System). Select new option by right clicking the partitions and provide the following details:

- New size: 50MBembed journal
- File System: FAT32
- Label: BOOT

Click Add button. Then, create another partition for storing RFS by entering the options below

- New Size: 1000MB
- File System: EXT3
- Label: RFS

Finally, click the green tick mark at the menu bar. The partition will be created and you can see two partitions created as BOOT and RFS.

5. Compiling kernel

Before compiling the kernel we need to configure it. It will be hard for the newbies. Once again, thanks to the kernel developers for providing all configurations in a single file. Go to the kernel directory and issue the following command.

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bb.org_defconfig
```

This will write the configurations in file bb.org_defconfig to .config file.

Then compile the kernel.

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- uImage dtbs LOADADDR=0x80008000 -j4
```

The above command will compile the kernel using the arm cross compiler having the load address as 80008000. “j4” corresponds to the number of process to be run during the compilation. Give the value as twice that of your cpu core. Mine is dual core, so I it gave as 4.

After compiling you can find the image files in “arch/arm/boot/” directory. Copy “uImage” file from this directory and also “am335x-boneblack.dtb” file from “arch/arm/boot/dts/” directory to the BOOT partition.

Then, create a file named as “uEnv.txt” in BOOT partition and copy the following code to it.

```
console=ttyS0,115200n8
netargs=setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait debug earlyprintk
mem=512M
netboot=echo Booting from microSD ...; setenv autoload no ; load mmc 0:1 ${loadaddr} uImage ; load mmc 0:1 $
{fdtaddr} am335x-boneblack.dtb ; run netargs ; bootm ${loadaddr} - ${fdtaddr}
uenvcmd=run netboot
```

This will be the file in which uboot will look upon while booting. The instructions in this file will make the uboot to boot from our kernel.

After completing the above steps you can find the following files in BOOT partition of sd card.

- uImage
- am335x-boneblack.dtb
- uEnv.txt

Note: Make sure you have installed mkimage tool and mounted the sd card.

6. RFS

You can download RFS here.

Instead of downloading RFS, we can create our own custom RFS using BusyBox, which is an elaborate process and hence merits the need for a separate post. For the sake of simplicity, we can download and uncompress the RFS.

```
$ sudo tar -xvf rootfs.tar.xz -C /media/mani/RFS/
$ cd /media/mani/RFS/rootfs/
$ sudo mv ./* ../
$ cd ../
$ sudo rmdir rootfs
```

The above command will uncompress the tar file and will place it in the RFS partition of SD card. Just replace “mani” with your username in the above command.

7. Install Kernel Modules

We need modules for proper working of the kernel. So, install the kernel modules by the following command.

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4 modules
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=/media/mani/RFS/
modules_install
```

That’s it. After completing the above steps, remove the SD card and place it in your BeagleBone. Connect the Bone to your PC via USB to serial converter and open the serial console using minicom in PC. (Set baud rate as 115200). After ensuring all things are correct, power on your BBB while holding the Boot switch (SW2). It will boot from your own custom kernel. Now you can cherish that you have created your own kernel image and deployed it in BeagleBone Black!!!

In my next post I will show you how to create custom RFS using BusyBox. As always, if you encountered any troubles on the way, just throw it in the comments, we will try to figure it out.