**Beagle bone board bring up process:**

**Setup cross compiler environment:**

1. Install Cross Compile toolchain

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

2. Install Device Tree Compiler

$ sudo apt-get install device-tree-compiler

$sudo apt-get install ncurses-dev

**General booting of beaglebone:**

<http://www.slideshare.net/pradeep_tewani/beaglebone-black-booting-process>

**U-boot source code:**

[http://www.embeddedforu.com/embeaglebone/how-toblack/](http://www.embeddedforu.com/embedded-linux/beaglebone/how-to-cross-compile-kernel-for-beaglebone-black/)

[http://www.embeddedforu.com/embeaglebone/beagleblack-embedded-linux-deve](http://www.embeddedforu.com/embedded-linux/beaglebone/beaglebone-black-embedded-linux-development-with-buildroot/)

[https://](https://w/)www.twam.info/hardware/beaglebone-black/u-boot-on-beaglebone-black

https://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-Bootloader:U-Boot

https://www.twam.info/hardware/beaglebone-black/u-boot-on-beaglebone-black

**SD card partition procedure:**

http://www.armhf.com/boards/beaglebone-black/bbb-sd-install/

**Kernel source code:**

http://wiki.beyondlogic.org/index.php?title=BeagleBoneBlack\_Building\_Kernel

https://github.com/beagleboard/linux/tree/3.14

**Root file system:**

<http://www.armhf.com/download/>

Download ubuntu-trusty-14.04-rootfs-3.14.4.1-bone-armhf.com.tar.xz

**How to create SDCard/eMMC Partition:**

-----------------------------------

Insert the microSD Card into your computer and observe which device it registers as by typing

**ls /dev/sd\***. (\* means a/b/c )

If you are uncertain, remove the microSD Card and the entry should go away. Once you know which device your microSD Card is, follow the instructions below

**sudo fdisk -l** to check the list of memory devices.check with **dmesg** command.

replacing /dev/sdX with the name of the microSD Card in your system.

$**sudo fdisk /dev/sd\***  (\* means a/b/c disk)

1.Begin partitioning the microSD Card by typing ( )

2.Initialize a new partition table by selecting ( o ), then verify the partition table is empty by selecting ( p ).

3.Create a boot partition by selecting ( n ) for ‘new’, then ( p ) for ‘primary’, and ( 1 ) to specify the first partition. Press ( enter ) to accept the default first sector and specify ( eg: 4095 ) for the last sector.

4.Change the partition type to FAT32 by selecting ( t ) for ‘type’ and ( b ) for ‘ FAT32 ’.

5.Set the partition bootable by selecting ( a ) then ( 1 ).

6.Next, create the data partition for the root filesystem by selecting ( n ) for ‘new’, then ( p ) for ‘primary’, and ( 2 ) to specify the second partition. Accept the default values for the first and last sectors by pressing ( enter twice ).

7.Press ( p ) to ‘print’ the partition table. It should look similar to the one below.

Disk /dev/sdb: 7948 MB, 7948206080 bytes

255 heads, 63 sectors/track, 966 cylinders, total 15523840 sectors

Units = sectors of 1 \* 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0xafb3f87b

Device Boot Start End Blocks Id System

/dev/sdb1 \* 2048 4095 1024 e W95 FAT16 (LBA)

/dev/sdb2 4096 15523839 7759872 83 Linux

8.Finally, commit the changes by selecting ( w ) to ‘write’ the partition table and exit fdisk.

9.Format the Partitions

Format partition 1 as FAT by typing ( sudo mkfs.vfat /dev/sdX1 )

Format partition 2 as ext4 by typing ( sudo mkfs.ext4 /dev/sdX2 )

**Rename the sd card partitions :**

**boot:**

sudo mkfs.vfat -F 32 "BOOT" /dev/sdb1

**rootfs:**

sudo mke2fs -j -L "rootfs" /dev/sdb2

**Create your own built Images:**

----------------------------

U-boot Configuration & Compilation:

-----------------------------------

1.Download u-boot source code from git server.

2. U-boot Configuration

$ cd ~/bbb/uboot-2016

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- am335x\_evm\_config

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- menuconfig

3. U-boot Compilation

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- -j4

Linux Kernel Configuration & Compilation:

----------------------------------------------

1. Download kernel source code from the git server.

1.Kernel Configuration

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- beaglebone\_defconfig

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- menuconfig

2. kernel Compilation

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- uImage LOADADDR=0x80008000 -j4

4. dtbs compilation

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabi- dtbs

**Ubuntu Minimal FS Porting on beaglebone black using Ownbuilt Binaries:**

Using SDCard:

------------

1. Prepare SD Card with 2 partitions: one for the BOOT and one for the rootfs.

2. Copy files over to the boot partition:

$ cp MLO u-boot.img uEnv.txt zImage .config & am335x-boneblack.dtb /media/$USER/boot

**memory loca,sequential booting,compressed kernel image.,board specs**

3. copy root file system.

$ mkdir rootfs

$ sudo mount /dev/sdbx2 rootfs

$ sudo tar xJvf ubuntu-trusty-14.04-rootfs-3.14.4.1-bone-armhf.com.tar.xz -C rootfs

$ sudo umount rootfs

4. After insert the SD card on BBB, update software packages otherwise sudo permission problem encounter.

$ sudo apt-get update (user name:ubuntu; Password:ubuntu)

5. Insert SDcard on BBB target and press switch2 and power on board. Now BBB booting from SD card.

MMC vs eMMC vs NAND vs NOR

--------------------------

MMC - Multi Media Card is similar to an SD card.

eMMC -

The eMMC (embedded MMC) architecture puts the MMC components (flash memory plus controller) into a small ball grid array (BGA) IC package for use in circuit boards as an embedded non-volatile memory system. This is noticeably different from other versions of MMC as this is not a user-removable card, but rather a permanent attachment to the circuit board. eMMC also does not support the SPI-bus protocol.

eMMC is great example of "managed NAND." These mass storage devices contain NAND flash memory along with other related controller and interface circuits.

**On Begle Bone Black:**

* Power on the board using +5v adaptor.
* Connect ttyUSB to the system and do settings for the BBB to the system.
* $ sudo minicom -s
* settings: /dev/ttyUSB0, hardware control flow—no, 115200,8N1.
* Close minicom by ESC ,ctrl+a z
* Press sysboot switch to board bring up starts with sd card.
* sequence starts with u-boot, kernel and rootfile system.
* If u want to stop in u-boot prompt enter key before boot delay time completes.
* help will give you all u-boot commands.
* In U-boot using menuconfig u can change bootdelay time and command shell prompt name.
* In the U-boot level help will list out all the u-boot commands.
* U-boot commands: printenv,setenv,boot,mm,md,bdinfo,i2c,spl

**On board leds ON/OFF in u-boot:**

#u-boot=>gpio set 53

gpio 53 value is 1 //user led ON

#u-boot=>gpio clear 53

gpio 53 value is 0 //user led OFF

#u-boot=>gpio set 54

gpio 54 value is 1 //user led ON

#u-boot=>gpio clear 54

gpio 54 value is 0 //user led OFF

#u-boot=>gpio toggle 55

gpio 55 value is 1 //user led ON

#u-boot=>gpio toggle 55

gpio 55 value is 0 //user led OFF

#u-boot=>gpio toggle 56

gpio 56 value is 0 //user led OFF

#u-boot=>gpio toggle 56

gpio 56 value is 0 //user led OFF

debian:temppwd

How to add a driver

go to ~/linux/drivers

mkdir driver-name

cd driver-name

create 3 file

1) driver.c 2)Kconfig 3) Makefile

in driver.c write your code

in Kconfig

config drivername

tristate “my driver added” // this message will show in menuconfig

in Makefile write

outside one kconfig

add your driver name in that

source "drivers/drivername/Kconfig"

one Makefile is also there

obj-$(CONFIG\_my driver added) +=hellodriver/

compile it