**BBB Booting**

How Booting takes place in Beagle Bone Black rev C.

1 ) Power On BBB & Power On self Test

2) First ROM boot loader(RBL) start executing.

 2.1) It is hard coded into ROM at the time of chip manufacturing. We are not able to modify it.

2.2) It manly initilise stack, Watchdog timer,

**Answer I found on Internet :**

**1 )**

Boot sequence (in order)

* Boot ROM
* X-loader --->>> 1st stage boot loader or (MLO)
* U-boot --->>> 2nd stage boot loader
* Linux

At power-up an OMAP3 device begins booting from internal Boot ROM.  This code is fixed during the manufacturing process and cannot be altered.  The Boot ROM reads boot configuration pins (SW4 on the OMAP3 EVM) which tell the Boot ROM where to look for the first external boot

loader.  The choices include NAND, UART, and SD/MMC Card.  Control is then passed to this first external boot loader called x-loader.  The x-loader application is included in the Linux PSP provided by TI and can be modified by the end user.  The x-loader application passes control to u-boot.  U-boot is also a boot loader and is considered the second external boot loader in this case.

U-boot is the application which passes control to the Linux system.  The main goal of u-boot is to retrieve the Linux kernel and provide the kernel with information about the location of the Linux file system.  U-boot can be configured to retrieve the kernel from NAND, SD/MMC Card, UART or Ethernet (via TFTP).  U-boot can also specify a root file system that is located in NAND (jffs2), SRAM (ram disk), SD/MMC card (ext3 partition) or mounted over IP (NFS).

U-boot then boots the Linux kernel.  The Linux kernel mounts the Linux root file system.

**2 )**

The first-stage bootloader runs directly on the board from power-up. I don't know the name of this bootloader(From TI official wiki, it called Boot Rom). This bootloader initializes a minimal amount of CPU and board hardware, then accesses the first partition of the SD card (which must be in FAT format), and loads a file called "MLO", and executes it. "MLO" is the second-stage bootloader.

The second-stage bootloader can apparently be one of either the X-loader or SPL. This bootloader apparently also just reads the first partition of the SD card, and loads a file called "u-boot.bin", and executes it. "u-boot.bin" is the third-stage bootloader.

The third-stage bootloader is U-boot, which is a popular bootloader for many different embedded boards and products. This bootloader has lots of different features, including an interactive shell, variables, ability to access the SD card and show its contents, etc. What happens next depends on the version of U-boot you have for the Panda board, and how it is configured. In a very simple configuration, U-Boot will look for the file "uImage" in the root of the first partition of the SD card (which, again, must be formatted as a FAT partition), and execute that. This is the Linux kernel. U-Boot passes the kernel a command line argument. Depending on how the kernel is configured it may accept the command line from U-Boot, or use one that was compiled into it when it was built.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**What is the difference between them?**

**Image**: the generic Linux kernel binary image file.

**zImage**: a compressed version of the Linux kernel image that is self-extracting.

**uImage**: an image file that has a U-Boot wrapper (installed by the **mkimage** utility) that includes the OS type and loader information.  
A very common practice (e.g. the typical Linux kernel Makefile) is to use a zImage file. Since a zImage file is self-extracting (i.e. needs no external decompressors), the wrapper would indicate that this kernel is "not compressed" even though it actually is

**Which type of kernel image do I have to use?**

You could choose whatever you want to program for.  
For economy of storage, you should probably chose a compressed image over the uncompressed one.  
Beware that executing the kernel (presumably the Linux kernel) involves more than just loading the kernel image into memory. Depending on the architecture (e.g. ARM) and the Linux kernel version (e.g. with or without DTB), there are registers and memory buffers that may have to be prepared for the kernel. In one instance there was also hardware initialization that U-Boot performed that had to be replicated.

**Why do we need all these multiple boot Loader. Why can't we Load whole Linux using ROM Loader or Single Boot Loader ?**

First I want to clearly that we can load whole OS from single Boot Loader. But This is not practical approach to do it.

Now we talk about why we are not doing it. The season is simple that the Internal SRAM size is very less and suitable for just starting minimum functionality. and It passes that **control to next stage** boot Loader. Which Further take care all other functionality. ROM boot loader is hard coded by manufactures. This Concept gives more multi boot loader gives more power to user. So that after boot up your SOC may work properly. and manufacture knows SOC better then you.

One reason can also be that your SOC does not know which kind of RAM are you going to use in your board development it can be DDR3, DDR4 or any other. So user can modify other parameters according to their RAM type.

 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Links :**

**1)** [Pentester Academy TV](https://www.youtube.com/channel/UChjC1q6Ami7W0E71TzPZELA) --->>>> very Important

https://www.youtube.com/watch?v=DV5S\_ZSdK0s

2 ) Very usefull

https://processors.wiki.ti.com/index.php/The\_Boot\_Process