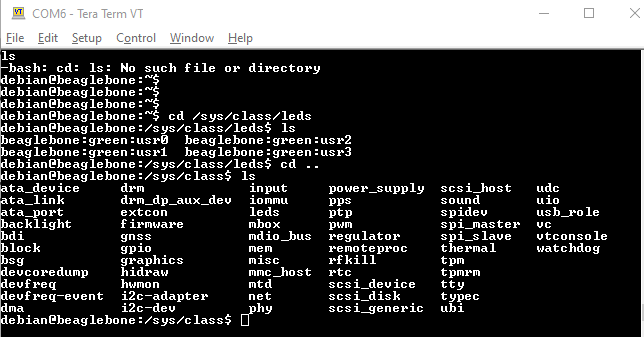
# BeagleBone Black RevC

-V.Balaji

## SYSFS

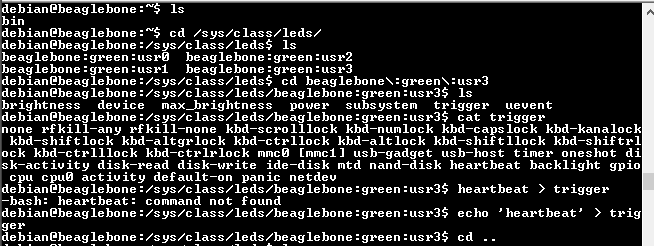
* SYSFS is a fly file system that exists on the RAM
* It is a windows that is used to peek into the various subsystem of the Linux Kernel like Networking subsystem, Memory subsystem, Bus, Device Drivers
* There are lots of examples in this repo balaji303/BeagleboneBlackC
* For SYSFS

## example



HeartBeat

* debian@beaglebone:/sys/class/leds/beaglebone:green:usr3$ echo 'heartbeat' > trigger





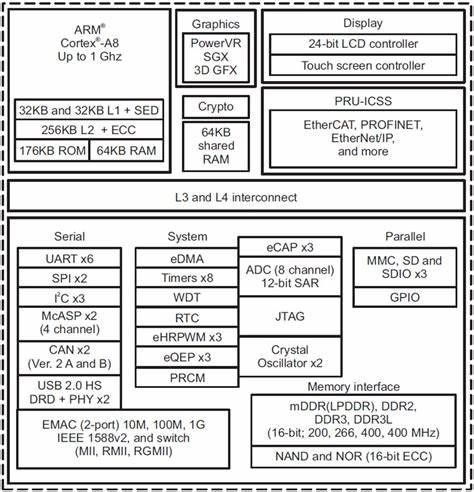






AM3358









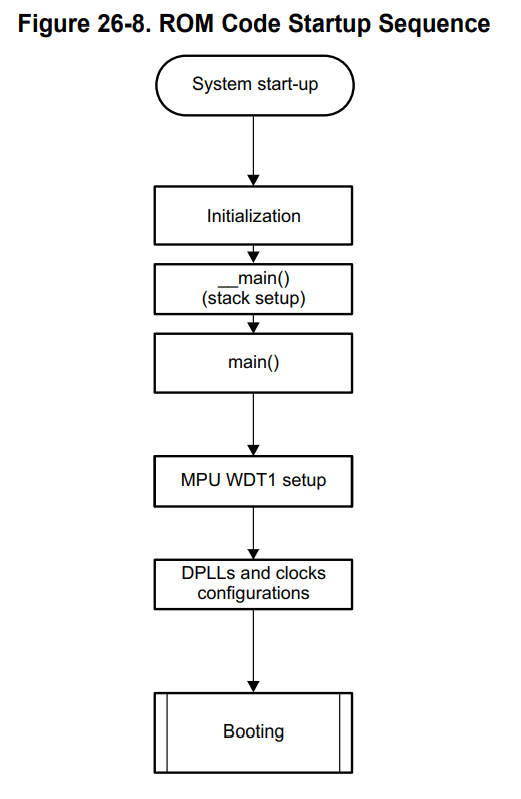




PLL Engine of SoC



@balaji303

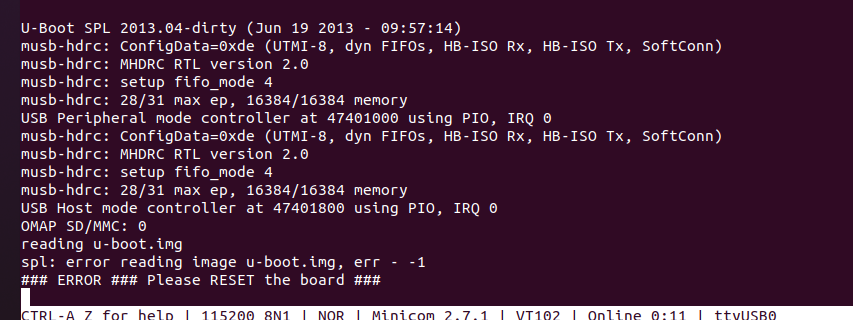


21





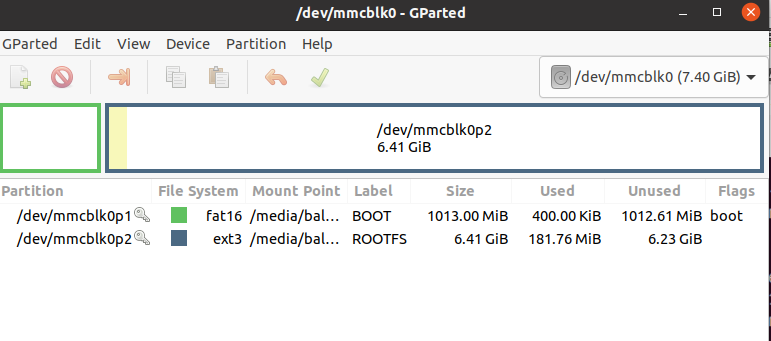




If only MLO is present in the SD card’s BOOT partition

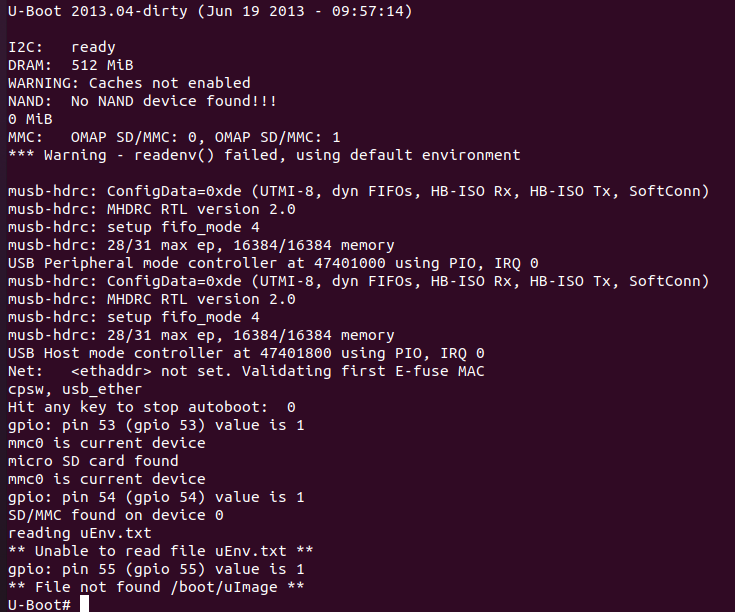
The error SPL searched for U-Boot but cant find

(Divided the SD card into two partition one is BOOT which is fat and another is ROOTFS which is ext3)



Moved MLO alone from OS.zip to Boot section and got the above error

Then moved U-BOOT also to Boot section



Now the U-Boot is loaded successfully but as there is no image the boot sequence failed

**Job Of U-BOOT**

* U-Boot initializes some peripherals like UART, I2C, NAND Flash, Ethernet, UART, USB because it supports loading kernel from all peripherals
* Loads the Linux Kernel image from various image
* You can change the boot behaviour of u-boot using uEnv.txt
* This uEnv.txt is used to set the environment variables
* U-image is just z-image with the uimage header

Boot Sequence

1. Power on Boot

2. RBL runs out of ROM

3. RBL copies the MLO to the internal ROM

4. MLO executes from the internal RAM

5. Once completed the MLO will load the U-Boot to the DDR

6. u-Boot executes in the DDR Ram

uEnv.txt

1. This is the place where commands and environmental variables are set.

2. bootm is the place where u-boot handes over the control to the Linux Kernel

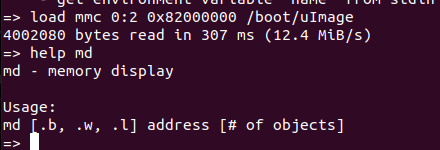
Enter U-Boot prompt by pressing space key after the device is powered ON

1. MMC0 interface ==> SD card

2. MMC1 interface ==> eMMC

3. To load a FAT based address we need to use the command ==> fatload

4. To load any other file system command we need to use ==> load



5. Load mmc 0:2 0x82000000 *boot/*Image

Load=> loads the image

mmc=> The Memory choose here is SD

0:2 => the SD card is in 0th position and the ROOTFS is in 2ed partition of the SD Card

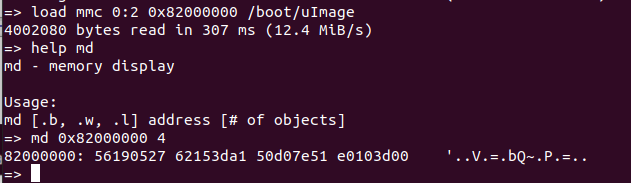
0x82000000 => is the load address for DDR memory

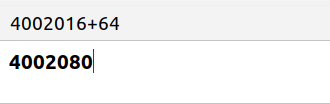
*boot*/uImage => uImages location in 2ed partition



Prints the 4 values from the uImage header file

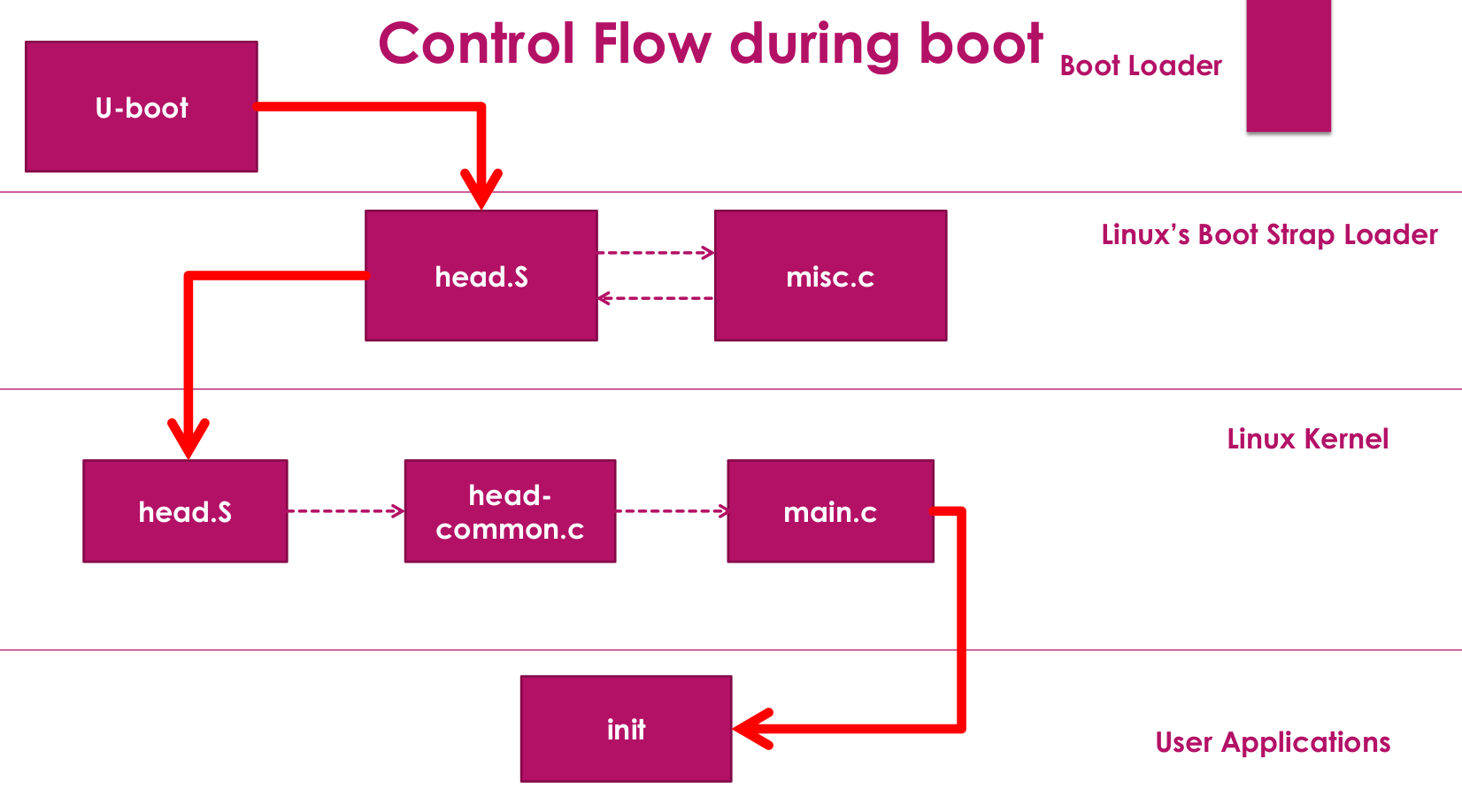
The below values are in little indian, I.e e0103d00 should be considered as 003d10e0

if we convert this last value to decimal from hex 003d10e0 => 4002016



Adding 64 bits to the Decimal of the last value gives the exact value of the mmc’s size

64 bits is the size of the uImage header



1. Kernel\_entry is the function that does this first thing to run the Kernel

3. decompression of the compressed kernel image is not the Job of U-Boot

4. misc.c file is the place where the decompression of the compressed kernel image happens.

5. head.S is actually arch specific code which does not depend on any SOC

6. arch means architecture

7. Register R10 holds the Processor register structure detected by lookup\_processor\_type

head-common.S

Init/main.c

Arch Dependant code

Entry level of Arch. Independant code of the Kernel

start\_kernel()

Function that starts the Kernel called in head-common.S

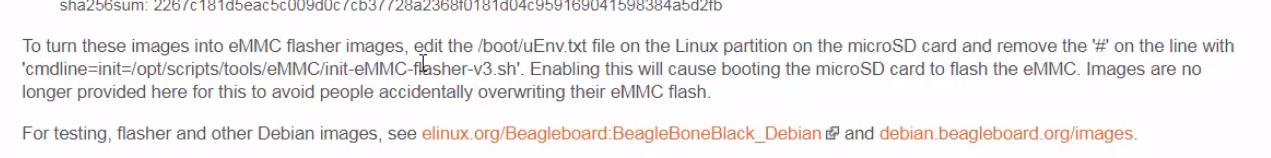
In Kernel Init

1. free\_initmem() - Prints the amount of free space in RAM

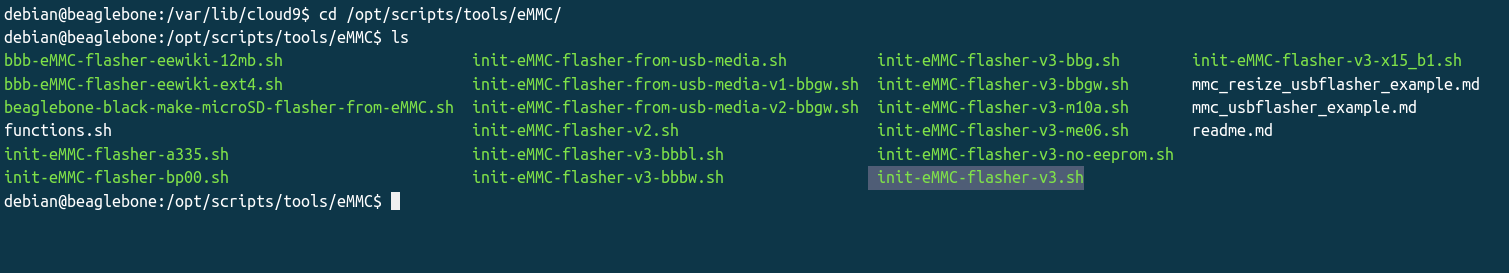
2. First the Kernel will try to run the init in the following order

1. /sbin/init
2. /etc/init
3. /bin/init
4. /bin/sh

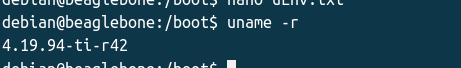
3.To check the version of the image in BBB run → lsb\_release -da

Follow the above pic to load the image from SD card to eMMC

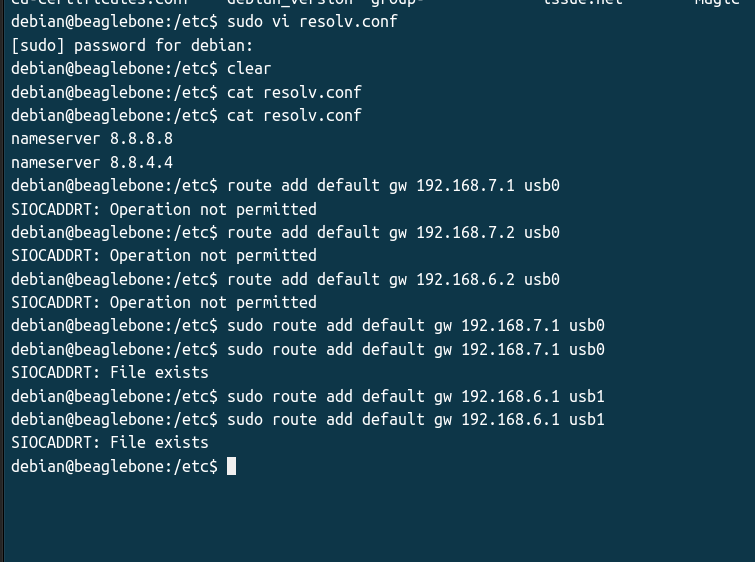
4. Goto the location /*opt*/scripts/tools/eMMC/

The selected is the shell script that flashes the content of SD card to eMMC

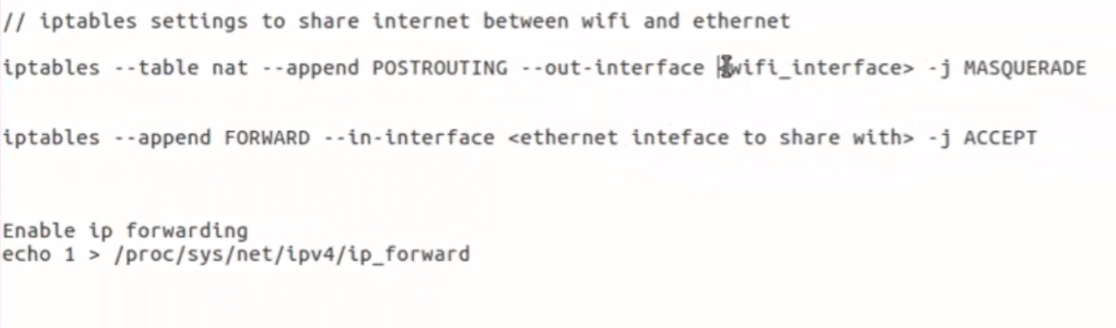
5. The other mehod to flash the content of SD card to eMMC is goto /*boot*/uEnv.txt and uncommment the line cmdline = init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh

6. To check the kernel version use this comment

7. To share the internet between the beagle bone and Linux you have to enable the Ip table setting and then enable the IP forwarding setting



8. To share the internet between the PC and Beagle bone

iptables --table nat --append POSTROUTING –-out-interface <wifi> -j MASQUERADE

iptables --append FORWARD –-in-interface <ethernet> -j ACCEPT

iptables --table nat --append POSTROUTING --out-interface wlp1s0 -j MASQUERADE

iptables --append FORWARD --in-interface enx30e283d4b993 -j ACCEPT

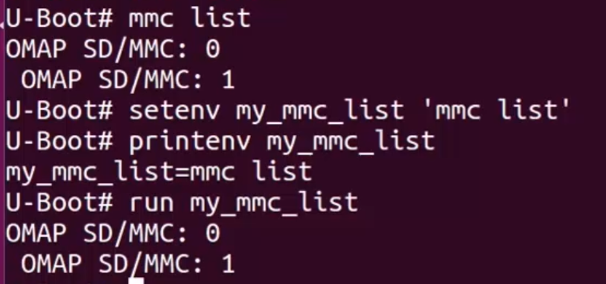
TRIED a LOT unable to solve the error ping: www.google.com: Temporary failure in name resolution

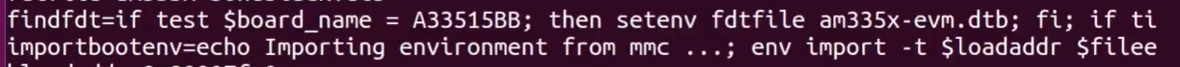
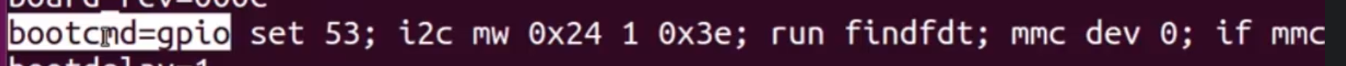
**LINUX DEVICE TREE or FLATTENED DEVICE TREE**

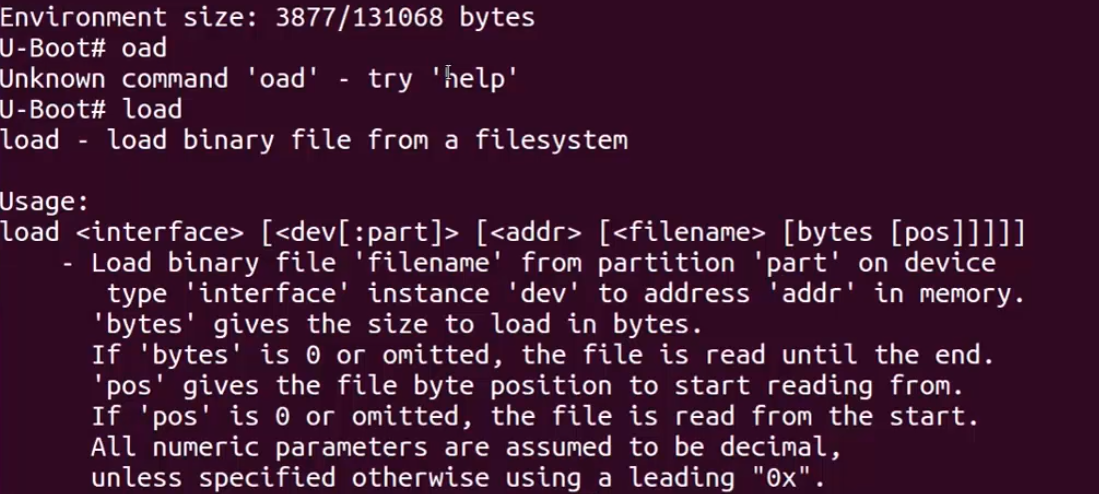
1. Devices like Pen drive (using USB) can announce their presence to operating system
2. Device like temp sensor (using I2C, SPI, Uart) can’t announce their presence to operating system these are called Platform devices
3. when a driver for a perticular platform device is loaded the Linux Kernel calls the “Probe” function of the driver if there is any match in its platform device database. In the “Probe” function of the driver, you can do the initializes.
4. Each Board will have a different version of the config file for the board
5. So Linux want to solve this problem of having different init sequence
6. ARM came up with an idea of creating a DTS (Device Source Tree) file created by the vendor
7. This file DTS has a lot of data structure to initializes the peripherals
8. DTS will be compiler using a compiler called DTC (Device Tree Compiler)
9. after compiling it creates a file called DTB (Device Tree)

U-BOOT --

1. **printenv <var>**  → prints the environmental variable from the uEnv.txt file
2. **setenv serverip 192.168.7.2** →creates and sets the envionmental variable to uEnv.txt file
3. Multiple lines of command can also be stored in the uEnv.txt and those can be executed using the **run <command>**
4. create own command

6. If we type boot all it does is just run the boot command → bootcmd

7. The boot command runs findfdt which is nothing but



8. We have partitioned the SD card as

Partition 01: BOOT of type FAT

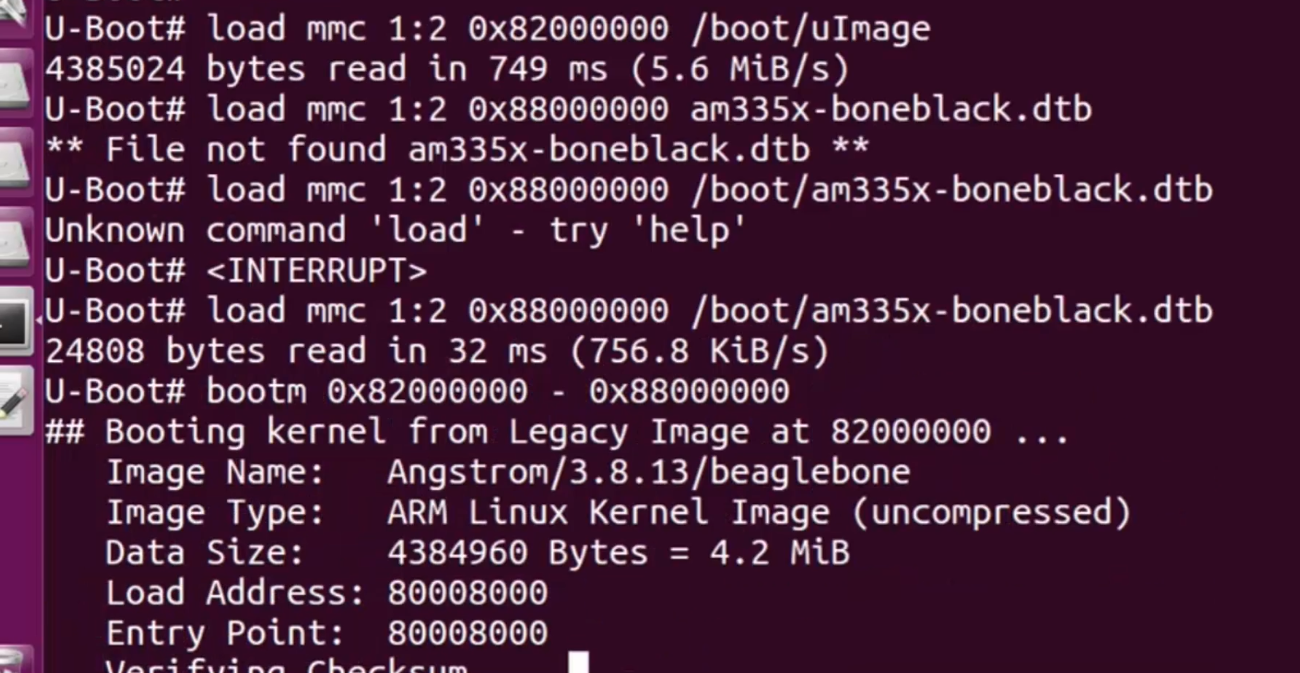
Partition 02: ROOTFS of type EXT3/4

9. dev 0: is the MMC0 interface which is SD card

dev 1: is the MMC1 interface which is eMMC

10. If we just try to load the kernel to the required address and boot using U-Boot it throws an error because the Device tree binary is not present

11. After loading the DTB to the required address and if we boot it works fine [lec 40]

  
  
  
  
  
  
  
  
  
  
  
For booting use the command bootm 0x82000000 - 0x88000000

  
  
  
  
  
  
If the boot is started there will not be prints from the Kernel because the Kernel doesn’t know that the port ttyO0 is the port available

So, we need to define this port and baudRate as an environment variable using bootargs

Use the following command

**setenv bootargs console = ttyO0, 115200**

Then if we use the command **printenv bootargs i**t prints the env value

After setting the env and trying the commands in order as follows

1. load mmc 1:2 0x82000000 *boot*/uImage

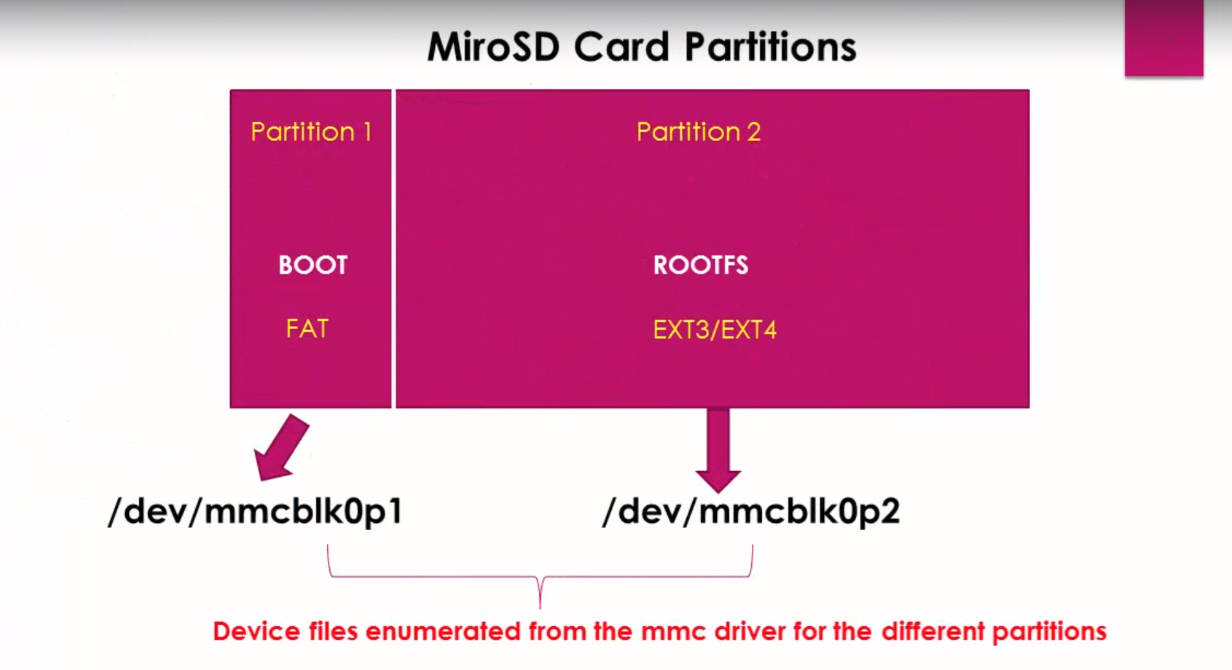
2. load mmc 1:2 0x88000000 *boot*/am335x-boneblack.dtb

3. setenv bootargs console = ttyO0, 115200

4. bootm 0x82000000 – 0x88000000

while using this the boot gets failed because it doesn’t know the location of ROOTFS file system for that we are altering the 3rd command

setenv bootargs console = ttyO0, 115200

SD card Partition details are above

Instead of typing the commands in U-Boot we can also have a txt file as uEnv.txt in BeagleBone and access these commands from there.

For that we have to create a txt file in PC and put all these commands inside it, but we need to transfer the file from PC to BBB using some protocols

u-Boot offers a lot of load protocols and commands as follows

|  |  |
| --- | --- |
| **Command** | **Details** |
| loadx | Send/Rec file using xmodem protocol |
| loady | Send/Rec file using ymodem protocol |
| Loadz | Send/Rec file using zmodem protocol |

1. Type loady in uboot

2. press Ctrl+c and S a pop-up appears asking to select modem protocol select yModem

3. select the file and hit “space” key

4. once the transfer begins and it will get complete

5. The file will be stored at the specified location address

1. if the file is downloaded using loadx method, it doesn’t mean that it is set, we need to set the Environment variables from it using a command **– env import -t <memory address> <size in bytes>**

**LINUX FILE SYSTEM:**

|  |  |
| --- | --- |
| Directory | Description |
| bin | Essential command binaries |
| boot | Static files of the boot loader |
| dev | Device files |
| etc | Host-specific system configuration |
| lib | Essential shared libraries and kernel modules |
| media | Mount point for removable media |
| mnt | Mount point for mounting a file system temporarily |
| opt | Add-on application software packages |
| run | Data relevant to running processes |
| sbin | Essential system binaries |
| srv | Data for services provided by this system |
| tmp | Temporary files |
| usr | Secondary hierarchy |
| var | Variable data |

#### bin/ :

This directory contains binaries of Linux commands which are used by both the system admins and users.

You don’t need privileges from your System admin to execute these commands neither you need root access. Remember that this folder will not contain binaries for all the Linux commands. There is a restriction on what types of commands have to be placed in this directory, because these binaries can be executed by the common user.

Below are some the commands you will find it in the bin/ directory.

cat                              Utility to concatenate files to standard output

chgrp                          Utility to change file group ownership

chmod                        Utility to change file access permissions

chown                         Utility to change file owner and group

cp                               Utility to copy files and directories

date                            Utility to print or set the system data and time

dd                               Utility to convert and copy a file

df                                Utility to report file system disk space usage

dmesg                        Utility to print or control the kernel message buffer

echo                           Utility to display a line of text

false                           Utility to do nothing, unsuccessfully

hostname                    Utility to show or set the system's host name

kill                                Utility to send signals to processes

ln                                 Utility to make links between files

login                           Utility to begin a session on the system

ls                                  Utility to list directory contents

mkdir                          Utility to make directories

mknod                      Utility to make block or character special files

more                          Utility to page through text

mount                       Utility to mount a file system

mv                              Utility to move/rename files

ps                                Utility to report process status

pwd                           Utility to print name of current working directory

you can see that commands related to "repairing", "recovering", "restoring", "network configuration" ,”modules install remove”  are not found in this directory.

#### boot/:

This directory contains the boot related files, which are required to boot the Linux. This directory may be read by the boot loader to read the boot images like Linux kernel image, dtb, vmLinux, initramfs, etc.

So this directory may be accessed by boot loader even before the kernel boots and mounts the file system.

#### dev/:

This is the place where you can find the "device files”.

You may be heard or read this statement "in unix/Linux devices are treated like file access" .

 Yes, if you want to access any i/o, networking devices, memory devices, serial devices, parallel devices, input output devices such as keyboard, mouse, display, everything will be treated like a file.

So, this directory will have the file entry for every device

For example: the i2c devices may have a file entries like this

/dev/i2c-0 ,

/dev/i2c-1,

The user Space application can use this device files to access those devices .

The ram may have a device file entry like /dev/ram0

The 2 partitions of the SD card may have entries like this:

/dev/mmcblk0p1

/dev/mmcblkop2.

The serial devices may have entries like this:

 /dev/ttyS0,

/dev/ttyS1,

/dev/ttyO0

It’s the responsibility of the respective drivers to populate this directory with the device files

#### etc/ :

This is the place where all the start-up scripts, networking scripts, scripts to start and stop networking protocols like NFS, networking configuration files,  different run level scripts will be stored.

1) Contains run level scripts, which will be used during different run levels

2) Contains start-up and shutdown scripts

3) Contains various scripts related to services like start/stop networking, start/stop NFS, etc

4) Contains various configuration files, like passwd, hostinfo, etc.

5) Contains various network configuration files

#### lib/ :

The major contents of this directory are

1) The dynamically loadable kernel modules.  ( later you will see, when we compile the kernel modules and when we run “modules install” command , all the kernel modules will go and sit in this directory under the sub directory "modules" .)

2) To store the Essential shared libraries (.so.\*) for dynamic linking.

 for example, 'C' shared library(libc), math library, python libray, etc,

#### media/ :

This is the mount point for the removable media like your USB flash drive, SD cards, camera, cell phone memory, etc .

For example, when i connect my SD card to the PC, there will be 2 device files will be created for each partition 1) /dev/sdb1 and 2) /dev/sdb2

and theses 2 device files are automatically mounted under the /media directory. So that i can access those 2 partitions just like folders.

Some examples:-

/media/cdrom for CD-ROM

/media/<your usb flash drive name>

#### mnt/ :

This is the place where you can mount the temporary file system.

The system admins can use a Linux commands to temporarily mount and un-mount the file system, if they want to transfer any file.

#### opt/ :

"opt" stands for "optional"

This directory will be used when you install any software packages for your Linux distribution.

For example if i run the command apt-get install <some packages name> then the package will be installed in this directory.

#### sbin/ :

The commands which come in the category of system administration will be stored in this directory, which is used by your sys admins for the purpose of networking configurations, repairing, restoring and recovering .

may be sbin stands for "System Admin's bin" ??

It also has root only command and need privileges to execute those commands.

These are the commands which  you will find in sbin/

|  |  |
| --- | --- |
| Command | Description |
| fastboot | Reboot the system without checking the disks (optional) |
| fasthalt | Stop the system without checking the disks (optional) |
| fdisk | Partition table manipulator (optional) |
| fsck | File system check and repair utility (optional) |
| fsck.\* | File system check and repair utility for a specific file system (optional) |
| getty | The getty program (optional) |
| halt | Command to stop the system (optional) |
| ifconfig | Configure a network interface (optional) |
| init | Initial process (optional) |
| mkfs | Command to build a file system (optional) |
| mkfs.\* | Command to build a specific file system (optional) |
| mkswap | Command to set up a swap area (optional) |
| reboot | Command to reboot the system (optional) |
| route | IP routing table utility (optional) |
| swapon | Enable paging and swapping (optional) |
| swapoff | Disable paging and swapping (optional) |
| update | Daemon to periodically flush file system buffers (optional) |

#### home/:

The /home directory contains a home folder for each user.

Each user only has write access to their own home folder and must obtain elevated permissions (become the root user) to modify other files on the system.

This directory will be used to store personal data of the user .

In a single user mode you may have folder like this

/home/ramesh

In a multiple user mode, you may have folder dedicated to each user.

/home/ramesh,  /home/suresh, /home/ram etc.

#### srv/ :

SRV stands for "Service"

The /srv directory contains “data for services provided by the system.” If you are using the Apache HTTP server to serve a website, you’d likely store your website’s files in a directory inside the /srv directory.

#### tmp/ :

Applications store temporary files in the /tmp directory.

#### usr/ :

According to FHS, it’s a "secondary hierarchy", the usr/ directory may contain the below sub directories

Directory                   Description

bin                              Most user commands

include                      Header files included by C programs

lib                                Libraries

local                           Local hierarchy (empty after main installation)

sbin                             Non-vital system binaries

share                          Architecture-independent data

/usr/bin contains binary of the commands for user programs

For example, if you have firefox on your system then just check it must be available under "/usr/bin" not under /bin. Because it is a binary related to user installed programs, similarly, "zip" command also you will find it under /usr/bin.

/usr/sbin contains, again privileged commands which may be used by the system admins, but these commands are for system administration purposes.

/usr/include will hold the header files which will be included from the C programs you write.

/usb/lib will again hold the shared libraries, linker/loader files , which enable your /usb/bin and /usr/sbin commands to execute.

BOOT BBB over UART0

1. We should make our board enter the UART0 boot mode

2. Once BBB entered the UART0 boot mode, it will be waiting for SPL command, we will send the SPL command over xmodem protocol using UART0

3. Once SPL command is receieved, we will wait for U-Boot command once we send the U-Boot command file over xmodem protocol, we will have control over the U-Boot using its commands

4. We will use xmodem protocol and to download other files and continue the boot process

we