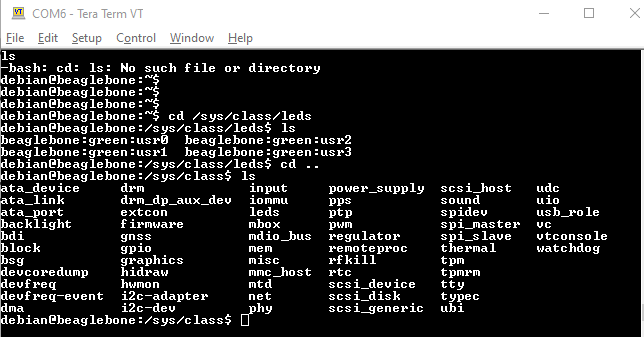
# BeagleBone Black RevC

-V.Balaji

## SYSFS

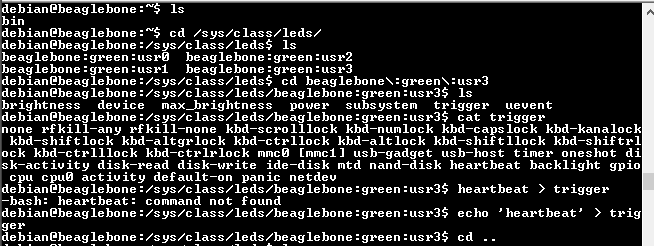
* SYSFS is a fly file system that exists on the RAM
* It is a windows that is used to peek into the various subsystem of the Linux Kernel like Networking subsystem, Memory subsystem, Bus, Device Drivers
* There are lots of examples in this repo balaji303/BeagleboneBlackC
* For SYSFS

## example



HeartBeat

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





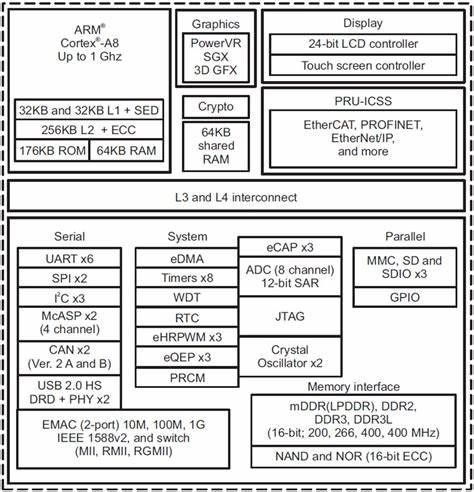






AM3358









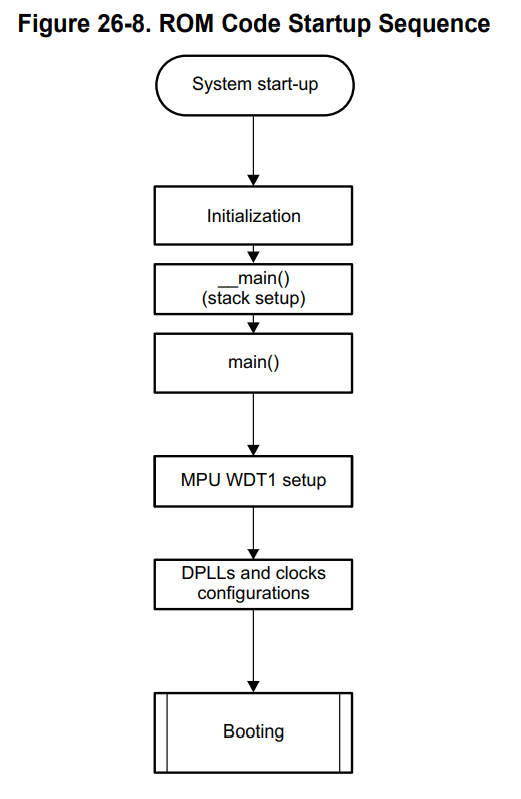




PLL Engine of SoC



@balaji303

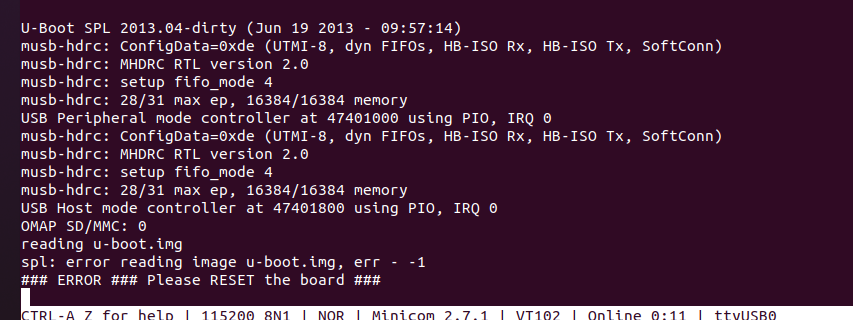


21





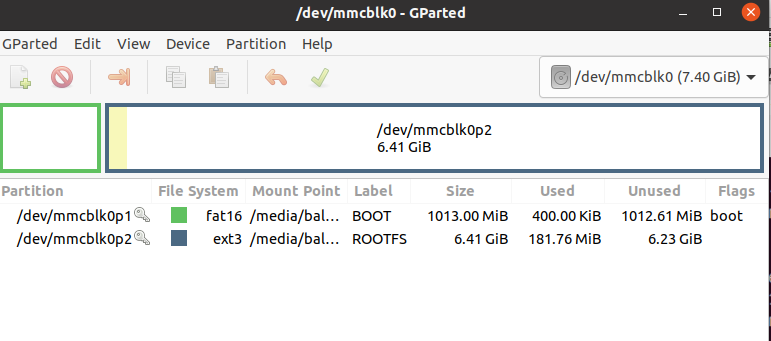




If only MLO is present in the SD card’s BOOT partition

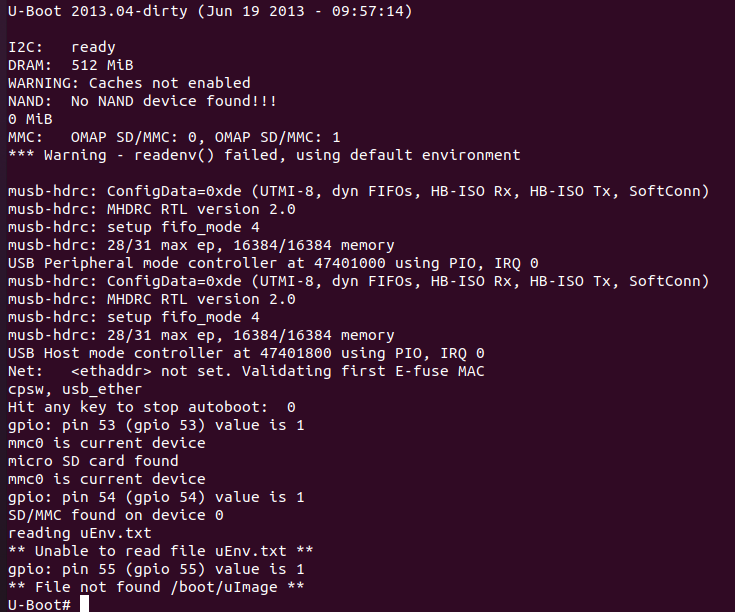
The error SPL searched for U-Boot but cant find

(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

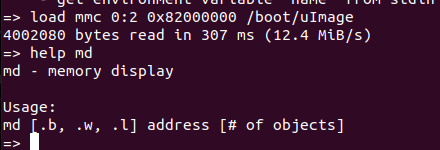
Enter U-Boot prompt by pressing space key after the device is powered ON

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

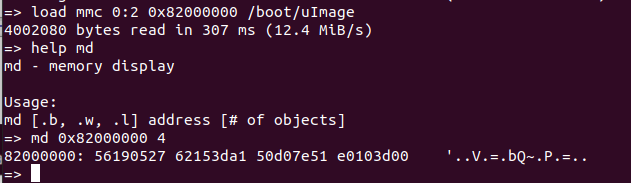
0x82000000 => is the load address for DDR memory

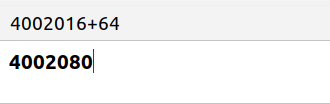
*boot*/uImage => uImages location in 2ed partition



Prints the 4 values from the uImage header file

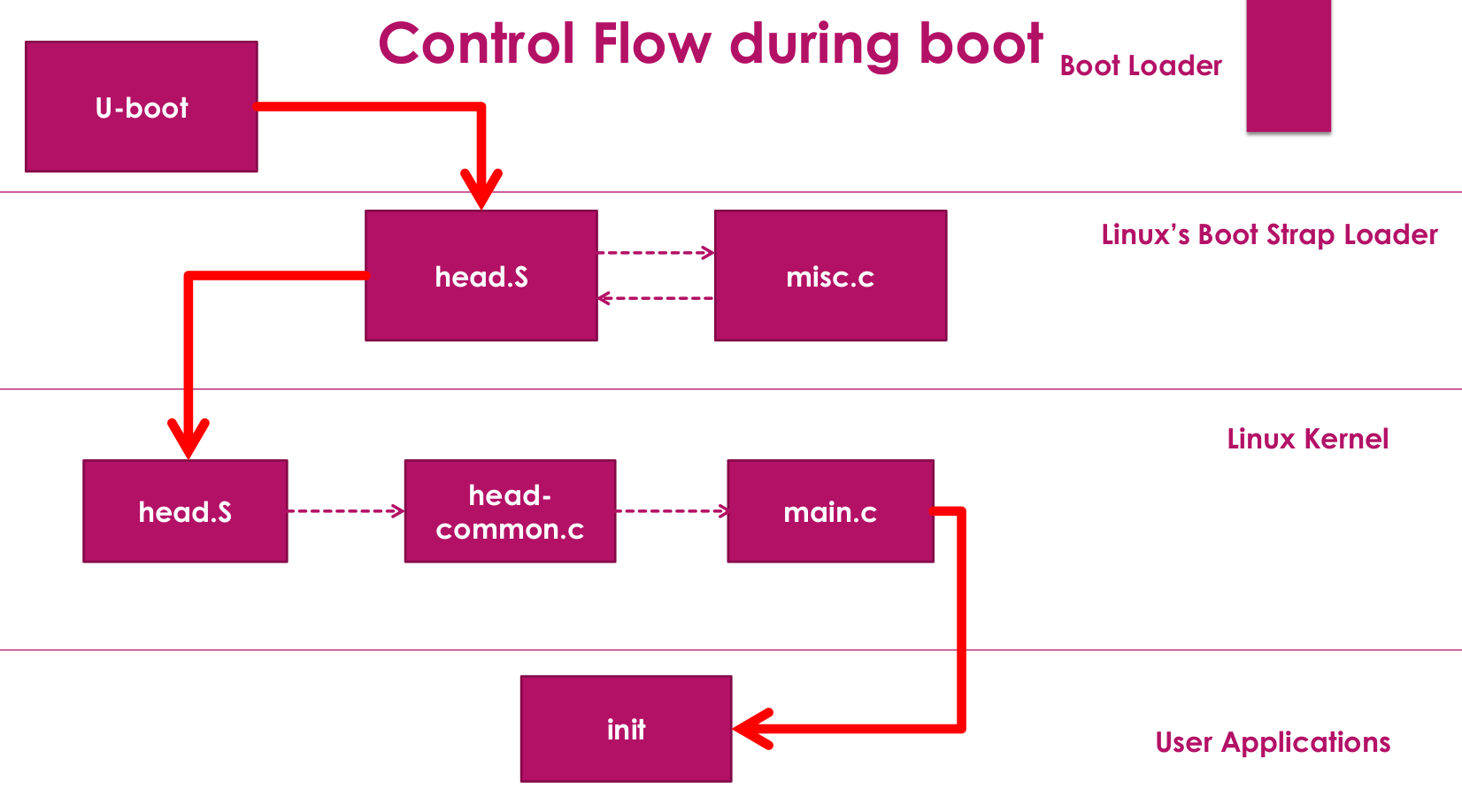
The below values are in little indian, I.e e0103d00 should be considered as 003d10e0

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

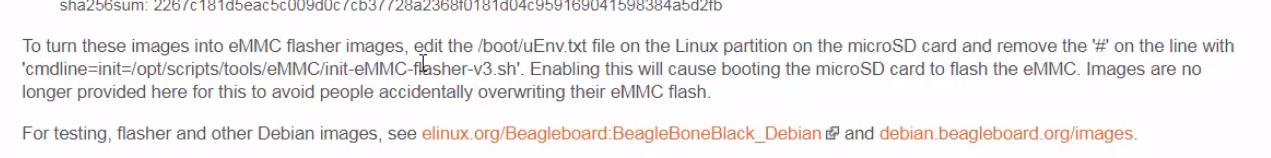
In Kernel Init

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

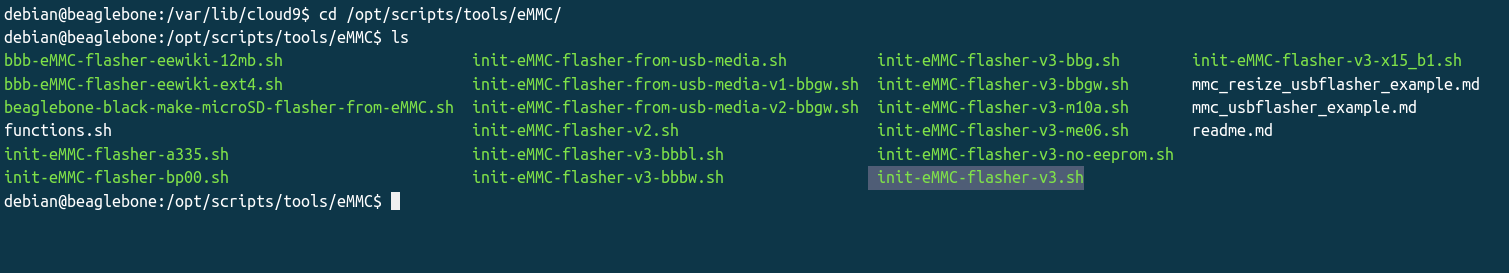
2. First the Kernel will try to run the init in the following order

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

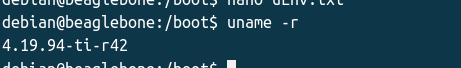
3.To check the version of the image in BBB run → lsb\_release -da

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

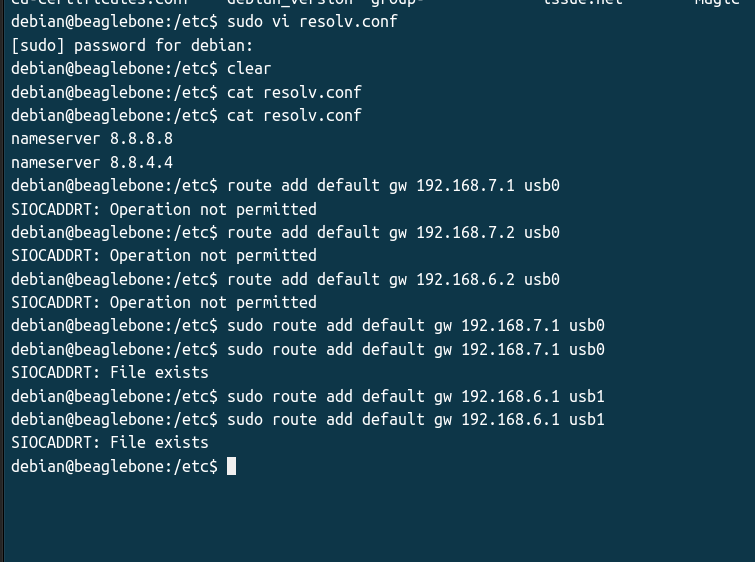
4. Goto the location /*opt*/scripts/tools/eMMC/

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

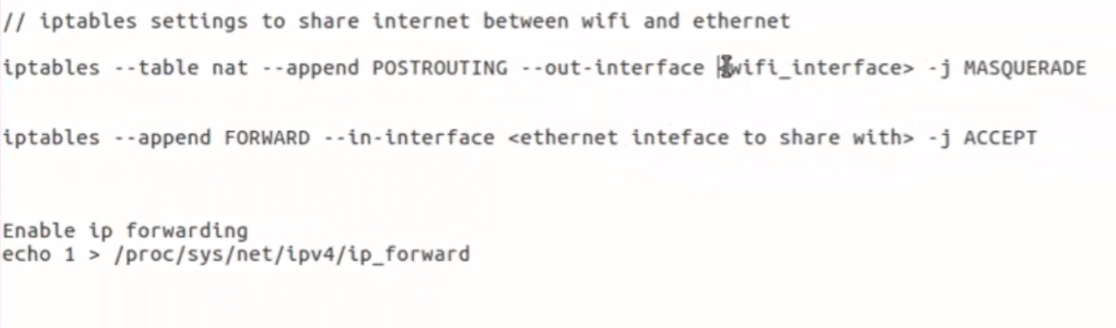
5. The other mehod to flash the content of SD card to eMMC is goto /*boot*/uEnv.txt and uncommment the line cmdline = init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh

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)
10. qwe

we