./

Learning Report – Embedded LINUX



Course Code: <CODE>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| 99002664 | 18/10/2020 | DEEKSHA P |  | VIVEK K | Activity and Task update |
| 99002664 | 7/11/2020 | DEEKSHA P |  | VIVEK K | Challenges update |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

Contents

[ACTIVITIES AND TASKS 4](#_Toc55658462)

[ACTIVITY 1 – CONFIGURATION OF THE BOARDS 4](#_Toc55658463)

[ACTIVITY 2 - DIFFERENCES BETWEEN RASPBERRY PI, DRAGON, IMX7 SABRE, BBB 5](#_Toc55658464)

[ACTIVITY 3 - DIFFERENCES IN DIFFERENT VERSIONS OF BBB AND EVOLUTION OF THE BEAGLEBONE. 6](#_Toc55658465)

[**BEAGLEBONE BLACK:** 7](#_Toc55658466)

[**BEAGLEBONE BLACK WIRELESS:** 8](#_Toc55658467)

[**BEAGLEBONE BLACK INDUSTRIAL TEMPERATURE:** 8](#_Toc55658468)

[**BEAGLEBONE GREEN WIRELESS:** 8](#_Toc55658469)

[**BEAGLEBONE BLUE:** 8](#_Toc55658470)

[**BEAGLEBOARD-XM:** 9](#_Toc55658471)

[**BEAGLEBOARD-X15:** 9](#_Toc55658472)

[**EVOLUTION OF BEAGLEBONE:** 9](#_Toc55658473)

[ACTIVITY 4 - PIN EXPANSION HEADER OF BBB AND LOCATE THE VARIOUS PERIPHERALS OF BONE. 11](#_Toc55658474)

[PIN EXPANSION HEADER OF BEAGLEBONE BLACK: 11](#_Toc55658475)

[**VARIOUS PERIPHERALS OF BEAGLEBONE BLACK:** 13](#_Toc55658476)

[ACTIVITY 5 – TESTING MLO AND U-IMAGE ON BBB 16](#_Toc55658477)

[ACTIVITY 6 - SHARING WI-FI WITH BBB 22](#_Toc55658480)

[ACTIVITY 7 - MANUAL BOOTING PROCESS 24](#_Toc55658481)

[ACTIVITY 8 - BOOTING PROCESS BY CREATING YOUR OWN uEnv.txt 24](#_Toc55658482)

[ACTIVITY 9 - FLASHING THE EMMC WITH THE DEBIAN LATEST VERSION 26](#_Toc55658483)

[ACTIVITY 10 - SERIAL BOOTING 27](#_Toc55658484)

[ACTIVITY 11 - CHANGING THE LOGIN NAME AND LOGO 28](#_Toc55658485)

[Challenge 1: Make uEnv.txt to Boot from MMC0 29](#_Toc55658486)

[Challenge 2: uEnvt.txt file to automate TFTP boot 30](#_Toc55658487)

[Challenge 3: Generic uEnv.txt 34](#_Toc55658488)

[Challenge 4: Increase the AUTOLOAD timings 34](#_Toc55658489)

[Challenge 6:  Busybox - Dynamic Compilation 36](#_Toc55658490)

[References 38](#_Toc55658491)

**List of Figures**

[**Figure 1: Beagle Bone Black 12**](#_Toc55659902)

[**Figure 2: Peripherals of BeagleBone Black 15**](#_Toc55659903)

[**Figure 3: Block Diagram of AM335X SOC 18**](#_Toc55659904)

[**Figure 4: Partition in Micro SD card 19**](#_Toc55659905)

[**Figure 5: MLO step in booting 21**](#_Toc55659906)

[**Figure 6: U-boot image booting step 21**](#_Toc55659907)

[**Figure 7: Log reading the uEnv.txt 22**](#_Toc55659908)

[**Figure 8: Loaded uImage in booting step 22**](#_Toc55659909)

[**Figure 9: Starting kernel boot 22**](#_Toc55659910)

[**Figure 10: Logs after starting the kernel 23**](#_Toc55659911)

[**Figure 11: Adding the server IP to the connection manager 24**](#_Toc55659912)

[**Figure 12: WIFI and Ethernet label 24**](#_Toc55659913)

[**Figure 13: The image depicting when we ping from 8.8.8.8 25**](#_Toc55659914)

[**Figure 14: The loading of uEnv.txt file 26**](#_Toc55659915)

[**Figure 15: The image depicting after booting the uEnv.txt file 26**](#_Toc55659916)

[**Figure 16: Etcher software to flash the eMMC 27**](#_Toc55659917)

[**Figure 17: Log of flashing onto eMMC 27**](#_Toc55659918)

[**Figure 18: Booting after loading the dtb file 28**](#_Toc55659919)

[**Figure 19: Starting kernel after flashing the latest version 29**](#_Toc55659920)

[**Figure 20: Changing Banner and Login name 30**](#_Toc55659921)

[**Figure 21: Beagle Bone Black Boot Log 31**](#_Toc55659922)

[**Figure 23: uEnv.txt entry 32**](#_Toc55659923)

[**Figure 24: IP address and server IP address 33**](#_Toc55659924)

[**Figure 25: Import env and run command 33**](#_Toc55659925)

[**Figure 26: Starting kernel autoboot 34**](#_Toc55659926)

[**Figure 27: Autoboot using tftp 34**](#_Toc55659927)

[**Figure 28: Generic uEnv file 35**](#_Toc55659928)

[**Figure 29: The image depicting the auto boot time being 5 seconds 36**](#_Toc55659929)

[**Figure 30: The image depicting the auto.config changed to 10 seconds 36**](#_Toc55659930)

[**Figure 31: The image depicting the auto boot time being 10 seconds 37**](#_Toc55659931)

[**Figure 32: Menuconfig** 38](#_Toc55659932)

# 

**List of Tables**

[**Table 1: Difference between boards 8**](#_Toc55658702)

[**Table 2: Difference between different versions of BBB 9**](#_Toc55658703)

[**Table 3: Expansion Header 14**](#_Toc55658704)

[**Table 4: Various Peripherals 16**](#_Toc55658705)

# 

# ACTIVITIES AND TASKS

# ACTIVITY 1 – CONFIGURATION OF THE BOARDS

**Step by step configuration of the boards and set up in the Linux**

Step 1: Connect the RX of TTL cable to TX of BBB board, TX of TTL cable to RX of BBB board and connect the common ground.

Step 2: Install minicom by using the command -sudo apt-get install minicom.

Step 3: Open terminal File->New tab and enter the command –dmesg and search for ttyUSB0

Step 4: Enter the command sudo minicom –s and go to serial port setup and change modem to ttyUSB0 by entering A.

Step 5: To check the bit rate hit E and to change the baud rate hit on C.

Step 6: Click on save setup as df1 and exit.

Step 7: Power up the BBB using the cable. For Armstrong users type the command root and hit enter.

**Step by step configuration of the boards and set up in the Windows**

Step 1: Download tera term for windows

Step 2: By following the steps install the drivers which are not digitally signed.

Step 3: Connect TTL cable and check the prolific COM in device manager.

Step 4: Open tera term and click on serial port set up and the prolific COM.

Step 5: Click on setup followed by serial port check if baud rate is 115220 and click on new settings.

Step 6: Then connect the power up cable.

# ACTIVITY 2 - DIFFERENCES BETWEEN RASPBERRY PI, DRAGON, IMX7 SABRE, BBB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **Raspberry PI** | **Dragon** | **imx7 Sabre** | **BeagleBoneBlack** |
| **Cost** | 35$ | 7,985rs | $179 | 45$ |
| **Processor** | 700 MHz ARM1176JZFS | 64-bit capable Qualcomm® Snapdragon™ 410E processor | Freescale’s i.MX6 Quad processor; 1 GHz;4 core, cortex A9; | 1 GHz TI Sitara AM3359 ARM Cortex A8 |
| **RAM** | 512 MB SDRAM @ 400 MHz | 1 GB LPDDR3 SDRAM @ 533MHz | 1 GB DDR3 SDRAM up to 533 MHz memory | 512 MB DDR3L  @ 400 MHz |
| **Storage** | SD | eMMC:eMMC 4.5, 8 GB  SD:Micro SD Card Slot | 8 GB eMMC Flash | 2 GB on-board eMMC, MicroSD |
| **Video Connections** | 1 HDMI,  1 Composite | Video Playback**:**Up to 1080p video playback  Codec Support**:**H.264 (AVC) | * 2x LVDS, * HDMI,   LCD expansion connector. | 1 MICRO-HDMI |
| **Operating System** | Raspbian (recommended), Ubuntu,Android, ArchLinux,FreeBSD, Fedora, RISC OS, others. | Linux, Android,   Windows | Android and Linux OSs. | Angstrom(default), Ubuntu, Android, ArchLinux, Gentoo, Minix, RISC OS, others. |
| **Supported Resolutions** | Extensive from 640x350 up to 1920x1200, this includes 1080P. | 1920x1080;  1280x720; | 1024x600 | 1280X1024  (5:4), 1024X768(4:3), 1280X720(16:9), 1440X900(16:10) all at 16Bit |
| **Audio** | Stereo over HDMI, Stereo from 3.5mm jack | PCM/AAC+/MP3/WMA, ECNS, Audio+ post-processing (optional} | CODEC | Stereo over HDMI |
| **Power consumption** | 150-350 mA @5V under varying conditions | 8V~18V@3A | 5v/5A output;  100/240V; | 210-460 mA @5V under varying conditions |
| **GPIO Capability** | 8 PINS | 36 PINS |  | 65 PINS |
| **Peripherals** | 2 USB Hosts,  1 Micro-USB Power,  1 10/100 Mbps Ethernet, RPI Camera connector. | USB 2.0 expansion;  2 x USB 2.0 Host 1 x USB 2.0 OTG;  Integrated ISP with support for image sensors up to 13MP | Gigabit ethernet interface;  USB Type-A (TE 1-1734775-1) and micro-AB interface (Hirose ZX62D-AB- 5P8) | 1 USB Host,  Mini-USB Client,  1 10/100 Mbps Ethernet. |

**Table 1: Difference between boards**

# ACTIVITY 3 - DIFFERENCES IN DIFFERENT VERSIONS OF BBB AND EVOLUTION OF THE BEAGLEBONE.

|  |  |  |  |
| --- | --- | --- | --- |
| BeagleBone Board | Processor | Distinguished Features | Common Applications |
| BeagleBone Black | AM3358 1GHz ARM® Cortex-A8 | BeagleBoard.org's flasgship low-cost, community-supported development platform | Ideal for developers and hobbysits. Gaming consoles, real-time tasks, audio systems, and more! |
| BeagleBone Black Wireless | AM3358BZCZ100 Microprocessor | BeagleBone Black with WiFi and BLE on-board - 802.11 b/g/n 2.4GHz | Similar to the BeagleBone Black, but where wireless connectivity is an integral part of the design |
| BeagleBone Blue | AM3358 1Ghz ARM® Cortex-A8 (Integrated into the OSD3358) | BeagleBoard's Linux-based robotics computer | Great for use in robotics applications or where systems control is an integral part of the design |
| BeagleBone Green | AM3358 1GHz ARM® Cortex-A8 | This version accommodates SeeedStudio Grove headers and 4x USB2.0 host. | Similar to the BeagleBone Black, but with integration of the large host of Grove Connectors by SeeedStudios |
| BeagleBone Green Wireless | AM3358 1GHz ARM® Cortex-A8 | The Green, with WiFi and BLE on-board -2.4 GHz TI WLinkTM8 Module | Automate your Green system with onboard Grove Connectors connected with onboard WiFi and BLE |
| BeagleBoard XM | AM37x 1GHz ARM Cortex-A8 | More sophisticated community-supported platform with mobile power specs | Project to development for innovators, with laptop like performance and next level expandability |

Table 2: Difference between different versions of BBB

**BEAGLEBONE BLACK:**

The BeagleBone Black is arguably the most popular development board Beagle Board produces. This SBC is the go-to starter if you are unfamiliar with Beagle Board products. BeagleBone Black comes loaded with Linux right out of the box, meaning that it can be used as a standalone computer immediately or reprogrammed using a different device.

The real heart of BeagleBone Black lies within the TI AM335 processor, which delivers speeds of up to 1 GHz. There are 4GBs available for onboard Flash storage supported by 512MBs of RAM. One of the greatest things about the BeagleBone Black is the two separate 46 pin expansion headers, meaning you have 92 IO pins available to control whatever your heart desires. Speaking of control, developers have access to the Programmable Real-Time Unit Subsystem onboard the Sitara processor. This PRUSS consists of two 32-bit MCUs enabling low-latency control for real-time applications. BeagleBone Black can be used in a wide range of applications, from home-automation to IoT devices to low-cost rapid prototyping and proof of concept devices. BeagleBone Black offers a great introduction to the Beagle Board community, no matter what your application. Beagle Board’s other dev. board options don't stray far from the BeagleBone Black platform.

**BEAGLEBONE BLACK WIRELESS:**

The BeagleBone Black Wireless swaps out a 10/100 Ethernet port for an onboard 2.4 GHz WiFi and Bluetooth connection device. BeagleBone Black Wireless is a perfect spin to use in your IoT devices right out of the box, instead of adding modules or capes onto the standard BeagleBone Black. Also, BeagleBone Black Wireless introduces the OSD3358 system-in-package (SIP) which makes derivative designs much easier.

**BEAGLEBONE BLACK INDUSTRIAL TEMPERATURE:**

The BeagleBone Industrial Temp was created in partnership with Arrow. It's exactly the same as the standard model except that it meets industrial temperature standards and is built for Industrial IoT development. With a temperature rating from -40 to +85 degrees Celsius, this board perfect for rugged applications that operate in extreme environments.

**BEAGLEBONE GREEN WIRELESS:**

Seeed Studio also makes the BeagleBone Green Wireless. The Green Wireless offers all the capability of the Green along with a built-in 2.4 GHz TI WLinkTM8 Module with two antennas. Note that the Green Wireless doesn’t use the OSD3358 SIP and also isn’t as compatible with BeagleBone Black capes as BeagleBone Black Wireless is, but it does work well with Google's IoT kit.

**BEAGLEBONE BLUE:**

BeagleBone Blue has the same amount of native memory and storage as Black, but is much more geared towards robotics because of its compelling set of peripherals. BeagleBone Blue also comes with an integrated Power Management system, allowing the board to natively host eight 6V servo controls, 4 DC motor outputs, and 4 encoder inputs, as well as other dedicated headers for other interfaces like UART, SPI, DSM2 radio, and even GPS. Also, the board comes equipped with a 9-axis Inertial Measurement Unit and a barometer, which makes it a great platform for drone research and tinkering.

**BEAGLEBONE BLUE:**

BeagleBone Blue has the same amount of native memory and storage as Black, but is much more geared towards robotics because of its compelling set of peripherals. BeagleBone Blue also comes with an integrated Power Management system, allowing the board to natively host eight 6V servo controls, 4 DC motor outputs, and 4 encoder inputs, as well as other dedicated headers for other interfaces like UART, SPI, DSM2 radio, and even GPS. Also, the board comes equipped with a 9-axis Inertial Measurement Unit and a barometer, which makes it a great platform for drone research and tinkering.

**BEAGLEBOARD-XM:**

The Xm boasts an extremely powerful AM37 1GHz processor. It has on-board Ethernet and four USB 2.0 ports, as opposed to the normal two on BeagleBone Black, meaning that it can incorporate USB peripherals like a keyboard, mouse, Wi-Fi, Bluetooth, web cameras or USB hubs. The xM does not have an onboard NAND, so the OS and data must be stored on a micro SD card – but that also means you can boot different operating systems from external media dependencies. Also, you can directly attach small camera modules for image processing on-board since the xM has an HD video-capable C64x+™ DSP core and open GL ES 2.0-capable 2d/3d graphics accelerator. These features combine together into a strong platform that can be used for anything from 3d gaming and robotics kiosks to digital signs, in vehicle entertainment, and even media centers.

**BEAGLEBOARD-X15:**

This is a top performing, mainline Linux-enabled, power-users’ dream board. BeagleBoard-x15 boasts a whopping 2GB of RAM, 4GB of flash storage, 2D and 3D GPUs, two 700 MHz DSPs, two M4 micro controllers, and four 32 bit PRUs, all controlled by the dual-core 1.5Ghz Sitara AM5728 processor. All of which is necessary for it to run its 157 GPIO pins! It even has on-board audio outputs and a heat sink! The x15 can be used for just about anything. You could even use this as your daily computer if you wanted. BeagleBoard-X15 is hot off the press, but Arrow.com has it in stock.

**EVOLUTION OF BEAGLEBONE:**

The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The Beagle Board was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip. The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence Or CAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used.

A modified version of the Beagle Board called the BeagleBoard-Xm started shipping on August 27, 2010. The BeagleBoard-Xm measures in at 82.55 by 82.55 mm and has a faster CPU core (clocked at 1 GHz compared to the 720 MHz of the Beagle Board), more RAM (512 MB compared to 256 MB), onboard Ethernet jack, and 4 port USB hub. The BeagleBoard-Xm lacks the onboard NAND and therefore requires the OS and other data to be stored on a micro SD card. The addition of the Camera port to the -xM provides a simple way of importing video via Leopard Board cameras.

Announced in the end of October 2011, the BeagleBone is a bare bone development board with a Sitara ARM Cortex-A8 processor running at 720 MHz, 256 MB of RAM, two 46-pin expansion connectors, on-chip Ethernet, a micro SD slot, and a USB host port and multipurpose device port which includes low-level serial control and JTAG hardware debug connections, so no JTAG emulator is required. The BeagleBone was initially priced at US$89.

A number of BeagleBone "Capes" have recently been released. These capes are expansion boards which can be stacked onto the BeagleBone Board (up to four at one time). BeagleBone capes include but are not limited to:

* LCD touchscreen capes (7" and 3.5")
* DVI-D cape
* Breakout cape
* Breadboard cape
* CAN bus cape
* RS-232 cape
* Battery cape

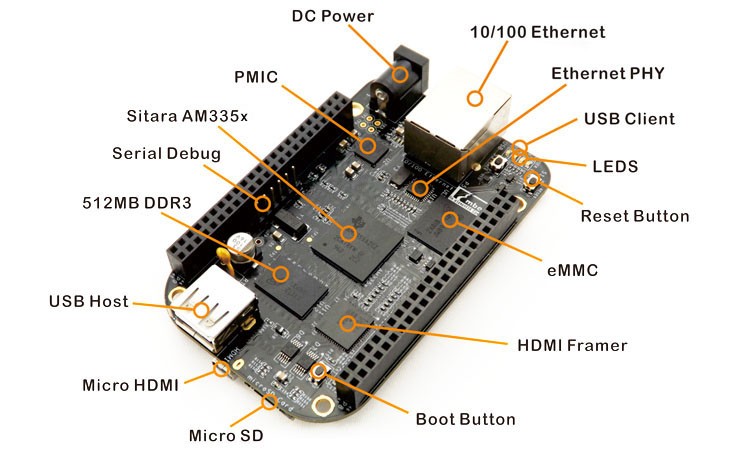
Launched on April 23, 2013 at a price of $45. Among other differences, it increases RAM to 512 MB, the processor clock to 1 GHz, and it adds HDMI and 2 GB of eMMC flash memory. The BeagleBone Black also ships with Linux kernel 3.8, upgraded from the original Beagle Bone’s Linux kernel 3.2, allowing the BeagleBone Black to take advantage of Direct Rendering Manager (DRM).

BeagleBone Black Revision C (released in 2014) increased the size of the flash memory to 4 GB. This enables it to ship with Debian GNU/Linux installed. Previous revisions shipped with Ångström Linux.

# ACTIVITY 4 - PIN EXPANSION HEADER OF BBB AND LOCATE THE VARIOUS PERIPHERALS OF BONE.

PIN EXPANSION HEADER OF BEAGLEBONE BLACK:

* Each digital I/O pin has 8 different modes to choose from, including GPIO.
* The PROC column is the pin number on the processor.
* The MODE columns are the different mode setting available for each pin.
* MODE5  is missing because it really just don’t do anything. The only pin that works in MODE5 is GPIO0\_7 in expansion header P9. It can be set as mmc0\_swdp.



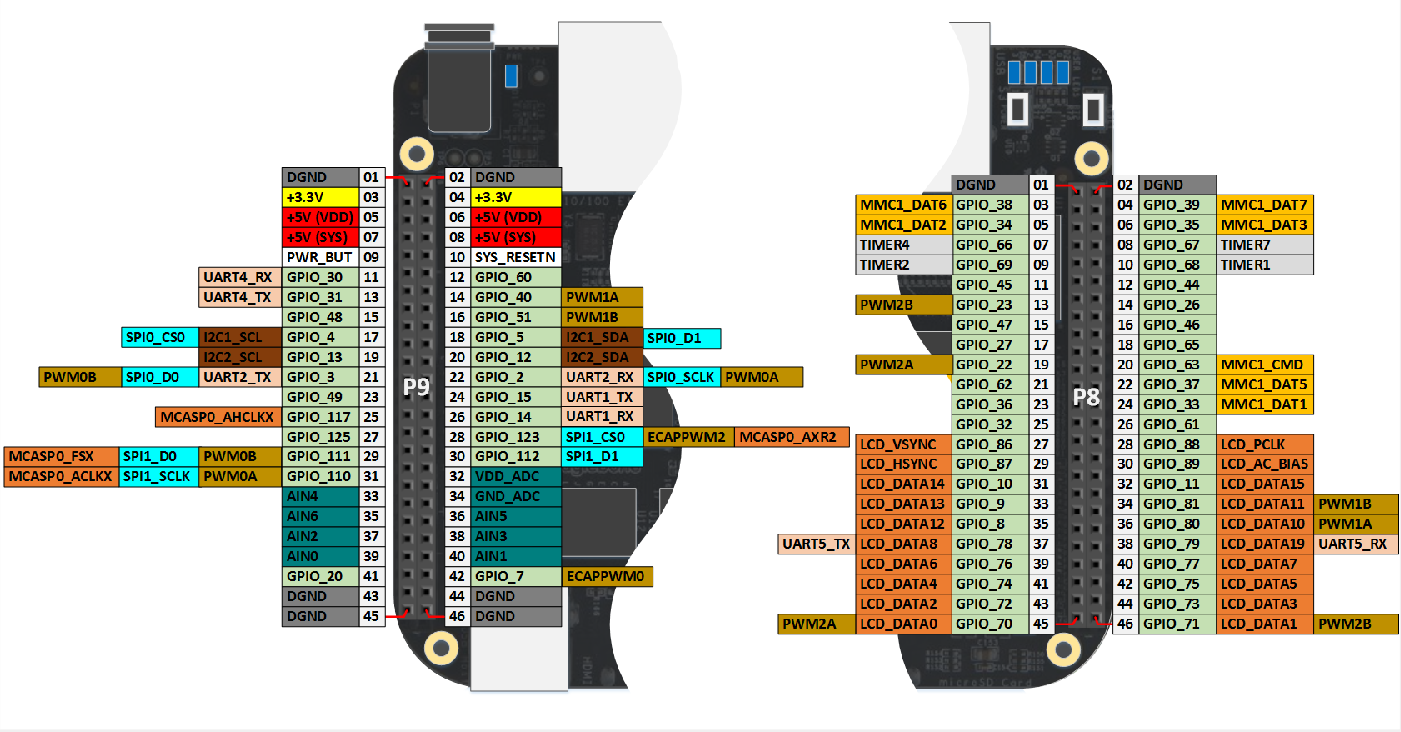
**Figure 1: Beagle Bone Black**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PIN** | **PROC** | **NAME** | **MODE0** | **MODE1** | **MODE2** | **MODE3** | **MODE4** | **MODE6** | **MODE7** |
| 1,2 |  |  |  |  | GND |  |  |  |  |
| 3 | R9 | GPIO1\_6 | gpmc\_ad6 | mmc1\_dat6 |  |  |  |  | GPIO1[6] |
| 4 | T9 | GPIO1\_7 | gpmc\_ad7 | mmc1\_dat7 |  |  |  |  | GPIO1[7] |
| 5 | R8 | GPIO1\_2 | gpmc\_ad2 | mmc1\_dat2 |  |  |  |  | GPIO1[2] |
| 6 | T8 | GPIO1\_3 | gpmc\_ad3 | mmc1\_dat3 |  |  |  |  | GPIO1[3] |
| 7 | R7 | TIMER4 | gpmc\_advn\_ale |  | timer4 |  |  |  | GPIO2[2] |
| 8 | T7 | TIMER7 | gpmc\_oen\_ren |  | timer7 |  |  |  | GPIO2[3] |
| 9 | T6 | TIMER5 | gpmc\_be0n\_cle |  | timer5 |  |  |  | GPIO2[5] |
| 10 | U6 | TIMER6 | gpmc\_wen |  | timer6 |  |  |  | GPIO2[4] |
| 11\* | R12 | GPIO1\_13 | gpmc\_ad13 | lcd\_data18 | mmc1\_dat5\* | mmc2\_dat1 | eQEP2B\_in |  | GPIO1[13] |
| 12\* | T12 | GPIO1\_12 | gpmc\_ad12 | lcd\_data19 | mmc1\_dat4\* | mmc2\_dat0 | eQEP2A\_in |  | GPIO1[12] |
| 13\* | T10 | EHRPWM2B | gpmc\_ad9 | lcd\_data22 | mmc1\_dat1\* | mmc2\_dat5 | ehrpwm2B |  | GPIO0[23] |
| 14\* | T11 | GPIO1\_26 | gpmc\_ad10 | lcd\_data21 | mmc1\_dat2\* | mmc2\_dat6 | ehrpwm\_tripzone |  | GPIO0[26] |
| 15\* | U13 | GPIO1\_15 | gpmc\_ad15 | lcd\_data16 | mmc1\_dat7\* | mmc2\_dat3 | eQEP2\_strobe |  | GPIO1[15] |
| 16\* | V13 | GPIO1\_14 | gpmc\_ad14 | lcd\_data17 | mmc1\_dat6\* | mmc2\_dat2 | eQEP2\_index |  | GPIO1[14] |
| 17\* | U12 | GPIO1\_27 | gpmc\_ad11 | lcd\_data20 | mmc1\_dat3\* | mmc2\_dat7 | ehrpwm0\_synco |  | GPIO0[27] |
| 18 | V12 | GPIO2\_1 | gpmc\_clk\_mux0 | lcd\_memory\_clk | gpmc\_wait1 | mmc2\_clk |  | mcasp0\_fsr | GPIO2[1] |
| 19\* | U10 | EHRPWM2A | gpmc\_ad8 | lcd\_data23 | mmc1\_dat0\* | mmc2\_dat4 | ehrpwm2A |  | GPIO0[22] |
| 20\* | V9 | GPIO1\_31 | gpmc\_csn2 | gpmc\_be1n | mmc1\_cmd\* |  |  |  | GPIO1[31] |
| 21\* | U9 | GPIO1\_30 | gpmc\_csn1 | gpmc\_clk | mmc1\_clk\* |  |  |  | GPIO1[30] |
| 22 | V8 | GPIO1\_5 | gpmc\_ad5 | mmc1\_dat5 |  |  |  |  | GPIO1[5] |
| 23 | U8 | GPIO1\_4 | gpmc\_ad4 | mmc1\_dat4 |  |  |  |  | GPIO1[4] |
| 24 | V7 | GPIO1\_1 | gpmc\_ad1 | mmc1\_dat1 |  |  |  |  | GPIO1[1] |
| 25 | U7 | GPIO1\_0 | gpmc\_ad0 | mmc1\_dat0 |  |  |  |  | GPIO1[0] |
| 26 | V6 | GPIO1\_29 | gpmc\_csn0 |  |  |  |  |  | GPIO1[29] |
| 27\* | U5 | GPIO1\_22 | lcd\_vsync\* | gpmc\_a8 |  |  |  |  | GPIO2[22] |
| 28\* | V5 | GPIO1\_24 | lcd\_pcik\* | gpmc\_a10 |  |  |  |  | GPIO2[24] |
| 29\* | R5 | GPIO1\_23 | lcd\_hsync\* | gpmc\_a9 |  |  |  |  | GPIO2[23] |
| 30\* | R6 | GPIO1\_25 | lcd\_ac\_bias\_en\* | gpmc\_a11 |  |  |  |  | GPIO2[25] |
| 31\* | V4 | UART5\_CTSN | lcd\_data14\* | gpmc\_a18 | eQEP1\_index | mcasp0\_axr1 | uart5\_rxd | uart5\_ctsn | GPIO0[10] |
| 32\* | T5 | UART5\_RTSN | lcd\_data15\* | gpmc\_a19 | eQEP1\_strobe | mcasp0\_ahclkx | mcasp0\_axr3 | uart5\_rtsn | GPIO0[11} |
| 33\* | V3 | UART4\_RTSN | lcd\_data13\* | gpmc\_a17 | eQEP1B\_in | mcasp0\_fsr | mcasp0\_axr3 | uart4\_rtsn | GPIO0[9] |
| 34\* | U4 | UART3\_RTSN | lcd\_data11\* | gpmc\_a15 | ehrpwm1A | mcasp0\_ahclkr | mcasp0\_axr2 | uart3\_rtsn | GPIO2[17] |
| 35\* | V2 | UART4\_CTSN | lcd\_data12\* | gpmc\_a16 | ehrpwm1\_tripzone | mcasp0\_aclkr | mcasp0\_axr2 | uart4\_ctsn | GPIO0[8] |
| 36\* | U3 | UART3\_CTSN | lcd\_data10\* | gpmc\_a14 | ehrpwm0\_synco | mcasp0\_axr0 |  | uart3\_ctsn | GPIO2[16] |
| 37\* | U1 | UART5\_TXD | lcd\_data8\* | gpmc\_a12 |  | mcasp0\_aclkx | uart5 | uart2\_ctsn | GPIO2[14] |
|  |  |  |  |  |  |  | \_txd |  |  |
| 38\* | U2 | UART5\_RXD | lcd\_data9\* | gpmc\_a13 |  | mcasp0\_fsx | uart5\_rxd | uart\_rtsn | GPIO2[15] |
| 39\* | T3 | GPIO2\_12 | lcd\_data6\* | gpmc\_a6 |  | eQEP2\_index |  |  | GPIO2[12] |
| 40\* | T4 | GPIO2\_13 | lcd\_data7\* | gpmc\_a7 |  | eQEP2\_strobe | pr1\_edio\_data\_out7 |  | GPIO2[13] |
| 41\* | T1 | GPIO2\_10 | lcd\_data4\* | gpmc\_a4 |  | eQEP2A\_in |  |  | GPIO2[10] |
| 42\* | T2 | GPIO2\_11 | lcd\_data5\* | gpmc\_a5 |  | eQEP2B\_in |  |  | GPIO2[11] |
| 43\* | R3 | GPIO2\_8 | lcd\_data2\* | gpmc\_a2 |  | ehrpwm2\_tripzone |  |  | GPIO2[8] |
| 44\* | R4 | GPIO2\_9 | lcd\_data3\* | gpmc\_a3 |  | ehrpwm\_synco |  |  | GPIO2[9] |
| 45\* | R1 | GPIO2\_6 | lcd\_data0\* | gpmc\_a0 |  | ehrpwm2A |  |  | GPIO2[6] |
| 46\* | R2 | GPIO2\_7 | lcd\_data1\* | gpmc\_a1 |  | ehrpwm2B |  |  | GPIO2[7] |

**Table 3: Expansion Header**

**VARIOUS PERIPHERALS OF BEAGLEBONE BLACK:**

* Up to 8, I/O pins can be configured with PWM (pulse width modulator) to generate signals to control motors without taking up any extra CPU cycle
* Pin number (32-40) in header P9 constitutes a single 12-bit analog to digital converter having 8 channels
* There are two I2C ports. The first I2C bus is utilized to read EEPROMS. It can also be used for other digital I/O operations without interfering with that function. The second I2C is available to configure according to the need of the user
* There are 2 SPI ports for fast shifting of data
* For advanced users, the BeagleBone black consists of 25 PRU low latency I/Os. They can make use 2 built-in 32 bit 200 MHz microcontrollers called PRU (Programmable Real-time Unit) in order to perform some real-time task



**Figure 2: Peripherals of BeagleBone Black**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PIN** | **PROC** | **NAME** | **MODE0** | **MODE2** | **MODE3** | **MODE4** | **MODE6** | **MODE7** |
| 1,2 |  |  |  |  | GND |  |  |  |
| 3,4 |  |  |  |  | DC\_3.3V |  |  |  |
| 5,6 |  |  |  |  | VDD\_5V |  |  |  |
| 7,8 |  |  |  |  | SYS\_5V |  |  |  |
| 9 |  |  |  |  | PWR\_BUT |  |  |  |
| 10 | A10 | RESET\_OUT |  |  |  |  |  |  |
| 11 | T17 | gpmc\_wait0 | mii2\_crs | gpmc\_csn4 | rmii2\_crs\_dv | mmc1\_sdcd | uart4\_rxd\_mux2 | gpio0[30] |
| 12 | U18 | gpmc\_be1n | mii2\_col | gpmc\_csn6 | mmc\_dat3 | gpmc\_dir | mcasp0\_aclkr\_mux3 | gpio1[28] |
| 13 | U17 | gpmc\_wpn | mii2\_rxerr | gpmc\_csn5 | rmii2\_rxerr | mmc2\_sdcd | uart4\_txd\_mux2 | gpio0[31] |
| 14 | U14 | gpmc\_a2 | mii2\_txd3 | rgmii2\_td3 | mmc2\_dat1 | gpmc\_a18 | ehrpwm1A\_mux1 | gpio1[18] |
| 15 | R13 | gpmc\_a0 | gmii2\_txen | rmii2\_tctl | mii2\_txen | gpmc\_a16 | ehrpwm1\_tripzone | gpio1[16] |
| 16 | T14 | gpmc\_a3 | mii2\_txd2 | rgmii2\_td2 | mmc2\_dat2 | gpmc\_a19 | ehrpwm1B\_mux1 | gpio1[19] |
| 17 | A16 | spi0\_cs0 | mmc2\_sdwp | I2C1\_SCL | ehrpwm0\_synci |  |  | gpio0[5] |
| 18 | B16 | spi0\_d1 | mmc1\_sdwp | I2CL\_SDA | ehrpwm0\_tripzone |  |  | gpio0[4] |
| 19 | D17 | uart1\_rtsn | timer5 | dcan0\_rx | I2C2\_SCL | spi1\_cs1 |  | gpio0[13] |
| 20 | D18 | uart1\_ctsn | timer6 | dcan0\_tx | I2C2\_SDA | spi1\_cs0 |  | gpio0[12] |
| 21 | B17 | spi0\_d0 | uart2\_txd | I2C2\_SCL | ehrpwm0B |  | EMU3\_mux1 | gpio0[3] |
| 22 | A17 | spi0\_sclk | uart2\_rxd | I2C2\_SDA | ehrpwm0A |  | EMU2\_mux1 | gpio0[2] |
| 23 | V14 | gpmc\_a1 | gmii2\_rxdv | rgmii2\_rxdv | mmc2\_dat0 | gpmc\_a17 | ehrpwm0\_synco | gpio1[17] |
| 24 | D15 | uart1\_txd | mmc2\_swdp | dcan1\_rx | I2C1\_SCL |  |  | gpio0[15] |
| 25 | A14 | mcasp0\_ahclkx | eQEP0\_strobe | mcasp0\_axr3 | mcasp1\_axr1 | EMU4\_mux2 |  | gpio3[21] |
| 26 | D16 | uart1\_rxd | mmc1\_sdwp | mcasp0\_axr2 | I2C1\_SDA |  |  | gpio0[14] |
| 27 | C13 | mcasp0\_fsr | eQEP0B\_in |  | mcasp1\_fsx | EMU2\_mux2 |  | gpio3[19] |
| 28 | C12 | mcasp0\_ahclkr | ehrpwm0\_synci |  | spi1\_cs0 | eCAP2\_in\_PWM2\_out |  | gpio3[17] |
| 29 | B13 | mcasp0\_fsx | ehrpwm0B |  | spi1\_d0 | mmc1\_sdcd\_mux1 |  | gpio3[15] |
| 30 | D12 | mcasp0\_axr0 | ehrpwm0\_tripzone |  | spi1\_d1 | mmc2\_sdcd\_mux1 |  | gpio3[16] |
| 31 | A13 | mcasp0\_aclkx | ehrpwm0A |  | spi1\_sclk | mmc0\_sdcd\_mux1 |  | gpio3[14] |
| 32 |  |  |  |  | VADC |  |  |  |
| 33 | C8 |  |  |  | AIN4 |  |  |  |
| 34 |  |  |  |  | AGND |  |  |  |
| 35 | A8 |  |  |  | AIN6 |  |  |  |
| 36 | B8 |  |  |  | AIN5 |  |  |  |
| 37 | B7 |  |  |  | AIN2 |  |  |  |
| 38 | A7 |  |  |  | AIN3 |  |  |  |
| 39 | B6 |  |  |  | AIN0 |  |  |  |
| 40 | C7 |  |  |  | AIN1 |  |  |  |
| 41 | D14 | xdma\_event\_intr1 |  | tclkin | clkout2 | timer7\_mux1 | EMU3\_mux0 | gpio0[20] |
| D13 | mcasp0\_axr1 | eQEP0\_index |  | mcasp1\_axr0 | emu3 |  | gpio3[20] |  |
| 42 | C18 | eCAPO\_in\_PWM0\_out | uart3\_txd | spi1\_cs1 | pr1\_ecap0\_ecap | spi1\_sclk | xdma\_event\_intr2 | gpio0[7] |
|  |  |  |  |  | \_capin\_apwm\_o |  |  |  |
| B12 | mcasp0\_aclkr | eQEP0A\_in | mcasp0\_axr2 | mcasp1\_aclkx |  |  | gpio3[18] |  |

**Table 4: Various Peripherals**

# ACTIVITY 5 – TESTING MLO AND U-IMAGE ON BBB

**BEAGLEBONE LINUX BOOTING PROCESS**

**STAGE 5**

**STAGE 4**

**STAGE 3**

**STAGE 2**

**STAGE 1**

**Stages in Boot Loading:**

U-Boot is both a first stage and second stage bootloader. When U-Boot is compiled we get two images, first stage (MLO) and second stage (u-boot.img) images. It is loaded by the system’s ROM code (this code resides inside the SoC’s and it is already preprogrammed) from a supported boot device. The ROM code checks for the various bootable devices that is available. And starts the execution from the device which is capable of booting. This can be controlled through jumpers, though some resistor based methods also exists.

Stage 2 bootloader is sometimes called a small SPL (Secondary Program Loader). SPL would do initial hardware configuration and load the rest of U-Boot i.e. second stage loader. Regardless of whether the SPL is used, U-Boot performs both first-stage and second-stage booting.

In third stage, U-Boot initializes the memory controller and SDRAM. This is needed as rest of the execution of the code depends on this. Depending upon the list of devices supported by the platform it initializes the rest. For example, if your platform has capability to boot through USB and there is no support for network connectivity, then U-Boot can be programmed to do exactly the same.

While setting up the linux kernel, setting up of the memory controller is the only mandatory thing expected by linux kernel. If memory controller is not initialized properly then linux kernel won’t be able to boot.

**Block Diagram of the Target:**

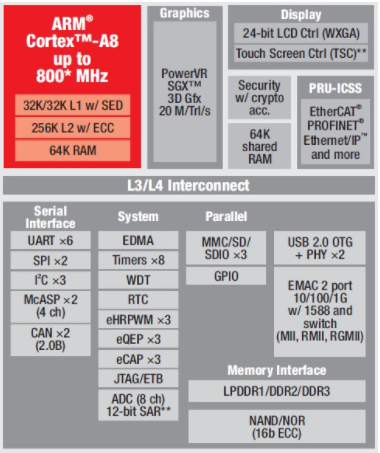


Figure 3: Block Diagram of AM335X SOC

**Partition:**

We are using a 8GB Micro SD Card and using “gparted” (gui based partition tool) to partition it. It is a much easier approach to use gparted and create the file systems. We have created two partitions:

1. BOOT

* New size: 2048mb (~2GB)
* File system: FAT16 with boot flag enabled
* Label: BOOT

1. ROOT

* New size: The entire remaining size
* File system: EXT3
* Label: ROOTFS

Choosing the size of the partition is availability as well as a personal choice. One important thing to note here is that the FAT16 partition has the boot flag set. This is needed for us to boot the device using Micro SD card. The below Figure 4 shows a clear picture of partitions in the Micro SD card.

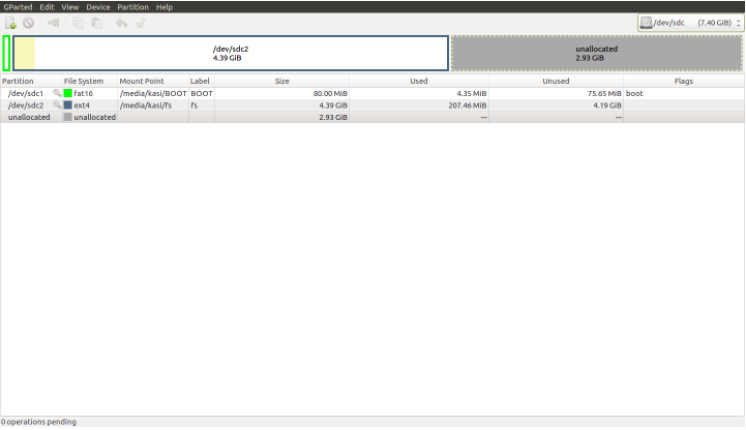


Figure 4: Partition in Micro SD card

After creating the partitions in Micro SD card, remove the card from the build machine and insert it again. In most of the modern distro’s partition in the Micro SD card get auto mounted which will confirm us that the partitions are created correctly. It will help us cross verify the created partitions.

**Booting into Linux on a BeagleBone proceeds in five steps, summarized here.**

1. At power up or reset, code in the on-board ROM initiates loading the U-Boot SPL (Second Program Load) image from the microSD card.

* This and other boot-related files are contained in the first partition (mmcblk0p1) of the microSD card. It must be formatted for a FAT filesystem.
* By default, the file containing the SPL image is named MLO (don't ask me why;it's a bit of cut-down u-boot code called x-loader in other contexts. Maybe MLO = Memory Load OMAP?).
* For technical reasons, this MLO file should start within the early sectors of the partition. Copying this file to a newly formatted partition first before copying any others has always satisfied this condition for me.
* As an aside, on some boards, the functionality of the SPL is contained in NAND memory.
* for the gory details about what occurs on-chip in this step, see the TI OMAP35x Applications Processor Initialization Reference Guide (Technical Reference Manual SPRUFD6A). I found it by searching the Internet for sprufd6a.pdf

2. The SPL initiates loading of a full U-Boot image from the microSD card.

* By default, the image is contained in a file named u-boot.img
* Again, it is contained in the first partition (mmcblk0p1) of the microSD card.

3. u-boot.img initializes itself, loads environmental variables, and executes commands. OMAP Bootloader Project

* User-defined environmental variables are loaded during this process.
* By default, the user-defined variables are contained in a file named uEnv.txt
* Again, it is contained in the first partition (mmcblk0p1) of the microSD card.
* u-boot is capable of obtaining network information via DHCP and loading it into environmental variables.

4. The executed commands initiate loading the Linux kernel.

* By default, the kernel is named uImage and is located...wait for it...in the first partition (mmcblk0p1) of the microSD card.
* Through environmental variables, the default behavior can be changed, for example, to load a kernel from a remote server via BOOTP/TFTP.
* kernel parameters can be passed in during this process just as they are in the traditional LILO/GRUB boot process.

5. The kernel initializes and mounts the root filesystem.

* By default, the root filesystem is contained in the second partition (mmcblk0p2) of the microSD card, formatted for an ext3 file system.
* Through kernel parameters passed in step 4, the kernel may directed to mount a root filesytem from, say, a remote server via NFS.

U-Boot is incredibly flexible with an enormous selection of configuration options in its source code. Free software being free software, any user can rebuilt U-Boot and assign different default settings and actions. That way madness lies, said King Lear, not because it's a bad idea intrinsically but because it isolates one from the larger community of BeagleBone enthusiasts.

* By default, the ROM in the Sitara AM3359 will boot from the MMC1 interface first (the onboard eMMC), followed by MMC0 (MicroSD), UART0 and USB0.
* If the boot switch (S2) is held down during power-up, the ROM will boot from the SPI0 Interface first, followed by MMC0, USB0 and UART0. This allows the BeagleBone Black to bypass the onboard eMMC and boot from the removable uSD (provided no valid boot device is found on SPI0.) This can be used to recover from a corrupted onboard eMMC.
* The Sitara AM3359 will try to load and execute the first stage bootloader called "MLO" from a Fat 12/16 or 32 bit MBR based filesystem. If using eMMC, this file is loaded using RAW mode. This means the ROM looks for a TOC at four specific offsets.
* MLO Booting (uBoot SPL Second Program Loader):

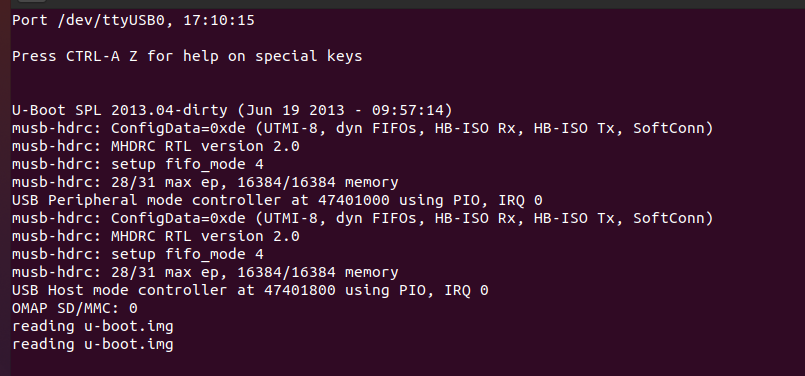


Figure 5: MLO step in booting

* MLO is a first stage uBoot Bootloader designed to load a second stage uBoot bootloader with enhanced features. This second stage bootloader is also found on the FAT partition with the filename of "u-boot.img"

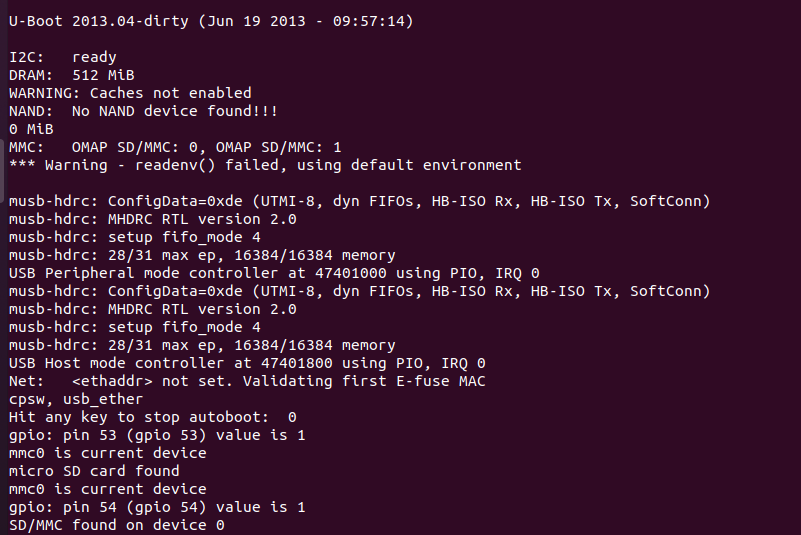


Figure 6: U-boot image booting step

* uBoot will load using a default environment space. This default space includes a variable bootenv=uEnv.txt and associated script that allows additional variables to be added or overwritten by adding them to an uEnv.txt file placed on the FAT partition. uBoot will attempt to load this file and append the extra variables:

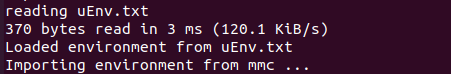


Figure 7: Log reading the uEnv.txt

* uBoot will then load the Linux Kernel and compiled Device Tree Binary blob from eMMC:

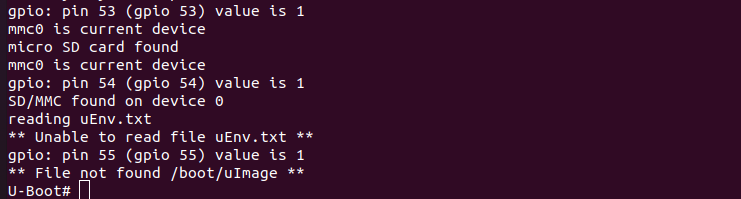


Figure 8: Loaded uImage in booting step

* And boot with the ext4 root file system being loaded from /dev/mmcblk0p2

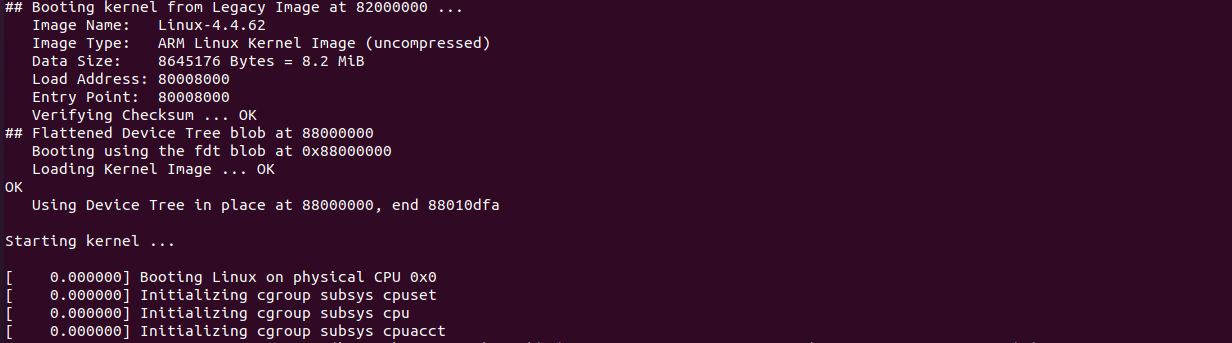


Figure 9: Starting kernel boot

* Final boot step after starting the kernel:

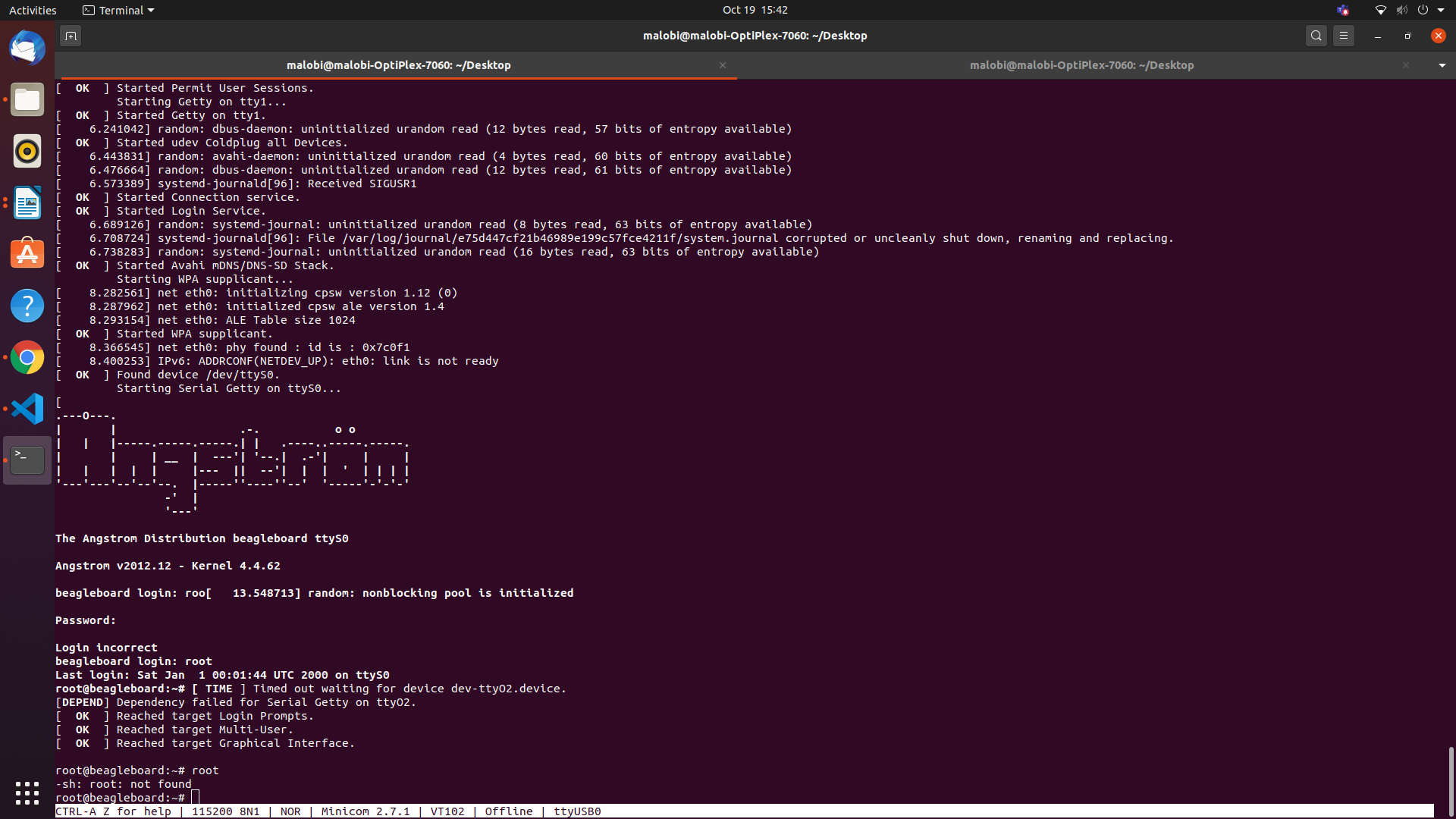


Figure 10: Logs after starting the kernel

# 

# ACTIVITY 6 - SHARING WI-FI WITH BBB

1. Initially boot the BBB
2. Enter the command “vi /etc/resolv.conf”
3. Add server

* nameserver 8.8.8.8
* nameserver 8.8.4.4

1. Find the ip address of USB0 by giving “ifconfig” command. After that add the gateway

“route add default gw 192.168.7.1 usb0”

1. On the host machine, enter into the root by giving the command “sudo –i”
2. enable the ip forwarding by giving the command

“echo 1 > /proc/sys/net/ipv4/ip\_forward”

1. Enable ip table settings to share internet between ethernet and wifi.

“iptables --table nat --append POSTROUTING -- out-interface <Wi-fi Interface> -j MASQUERADE”

“iptables --append FORWARD --in-interface <Ethernet interface> -j ACCEPT”

1. Finally in BBB root enter ping 8.8.8.8 so the Wi-Fi will be enabled.

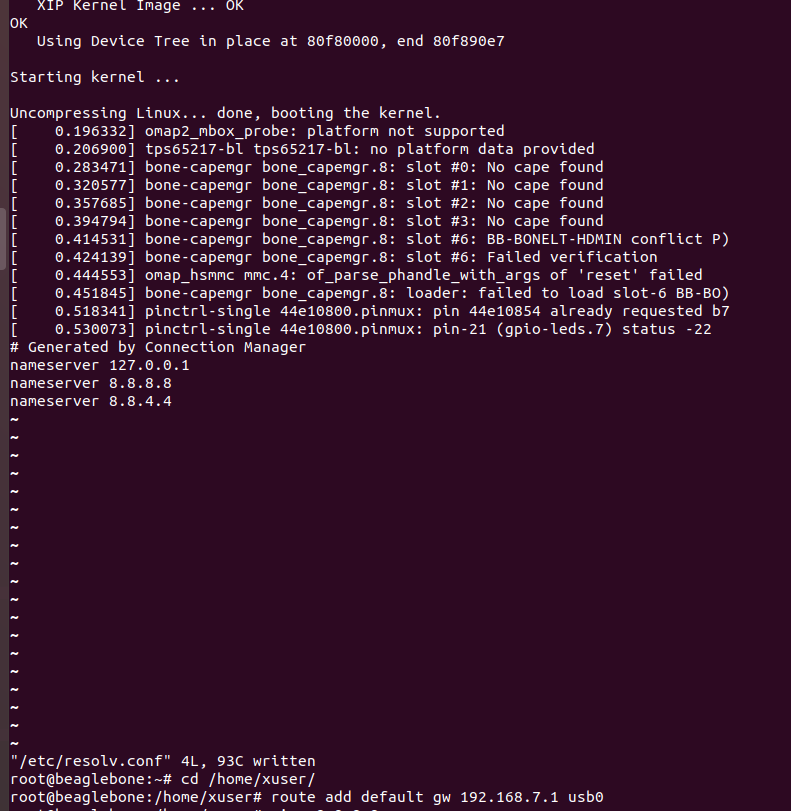


Figure 11: Adding the server IP to the connection manager

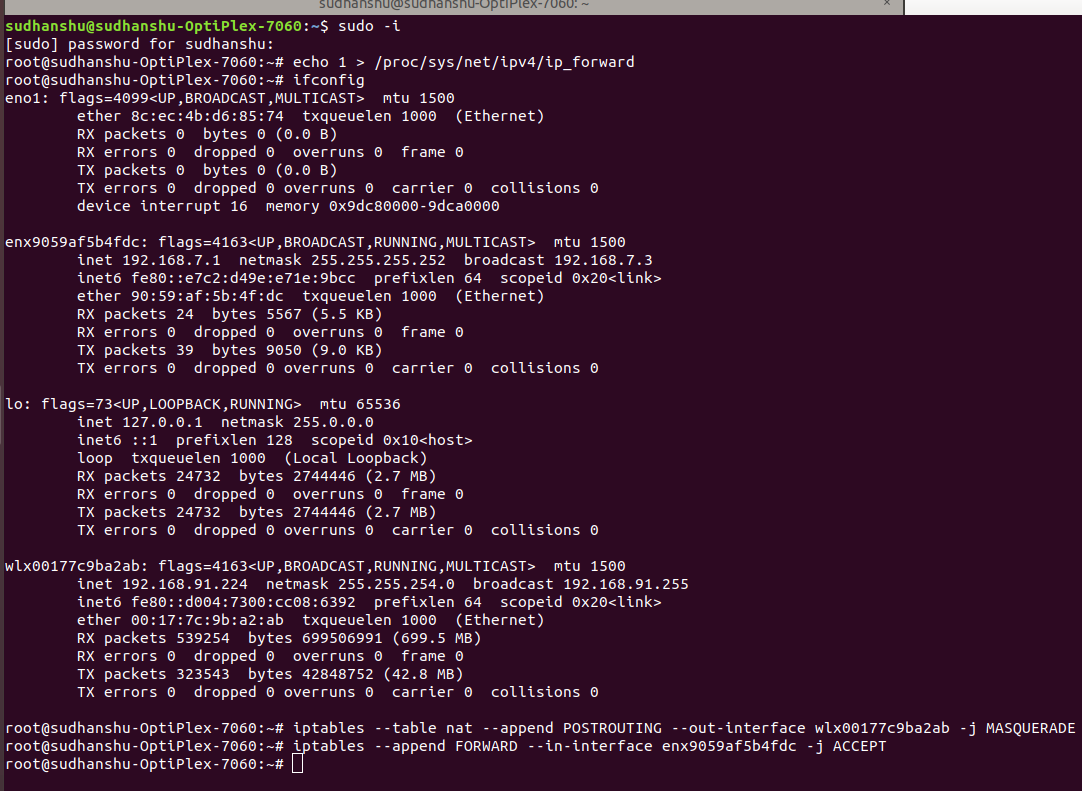


Figure 12: WIFI and Ethernet label

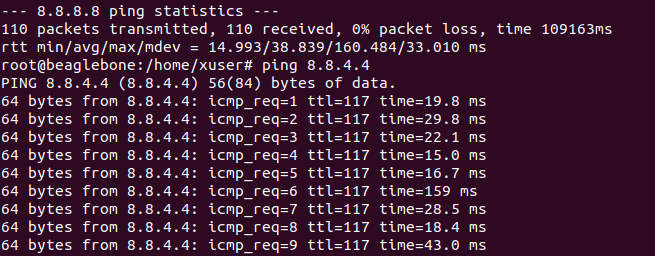


Figure 13: The image depicting when we ping from 8.8.8.8

# ACTIVITY 7 - MANUAL BOOTING PROCESS

1. Start the booting process from the SD card.
2. Pause at U-Boot by pressing any key while booting.
3. Load the boot image into the DDR of BBB by using the command

“load mmc 0:2 0x82000000 /boot/uImage”

1. Load the dtb file onto BBB by using the command

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

1. Continue the booting process by giving the “boot” command

# ACTIVITY 8 - BOOTING PROCESS BY CREATING YOUR OWN uEnv.txt

1. Start the booting process
2. Pause at U-Boot by pressing any key while booting.
3. Enter the command “loady”
4. Press ctrl+A then click S
5. Select ymodem and go to the uEnv.txt file location and load the uEnv.txt file.



Figure 14: The loading of uEnv.txt file

1. Enter the command “env import –t <address> <Number of bytes>”
2. Continue the boot process by using the command “boot”

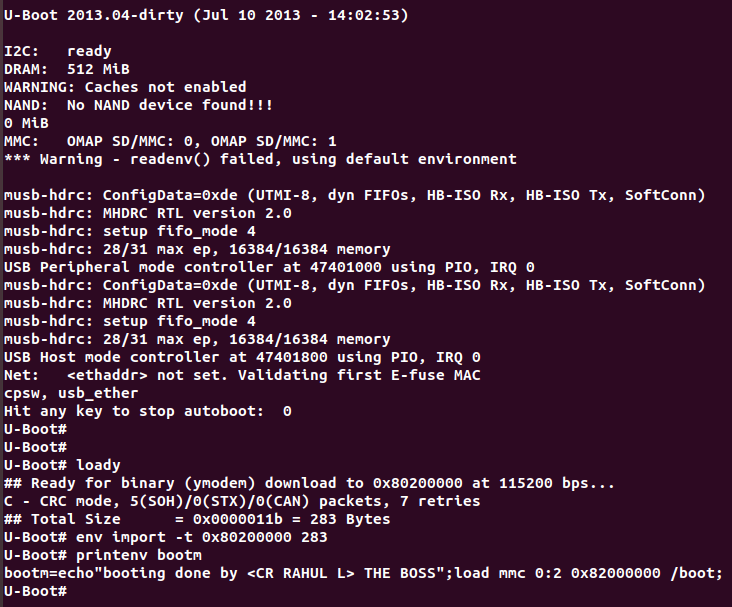


Figure 15: The image depicting after booting the uEnv.txt file

# ACTIVITY 9 - FLASHING THE EMMC WITH THE DEBIAN LATEST VERSION

1. Download the latest debian image from the beagleboard website- beagleboard.org
2. Download Balena Etcher software
3. Insert the SD card
4. Open the downloaded debian image using the balena Etcher and select the required target to load which is SD card. Flash the image to sd card.



Figure 16: Etcher software to flash the eMMC

1. Now insert the SD card to BBB and start flashing onto eMMC.

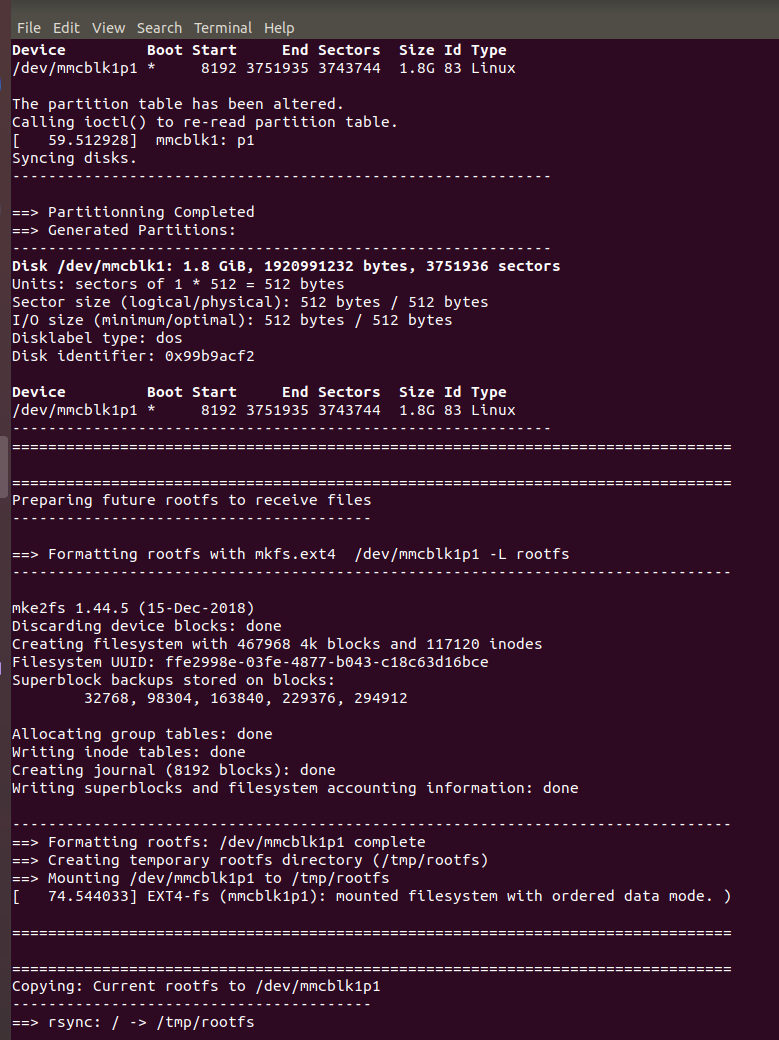


Figure 17: Log of flashing onto eMMC

# ACTIVITY 10 – SERIAL BOOTING

1. Download the serial-boot zip file.
2. Connect the TTL cable and enter into the minicom.
3. By pressing the S2 button give the power to the board using the power adaptor to boot from the UART.
4. Press CTRL+A and then press S and enter into the xmodem. Import the uboot-spl-bin file from the downloaded directory.
5. Enter command “loadx 0x82000000” and load U-Image to this address via xmodem.
6. Enter command “loadx 0x88000000” and load dtb file to this address via xmodem.
7. Enter command “loadx 0x88080000” and load intramfs to this address via xmodem.

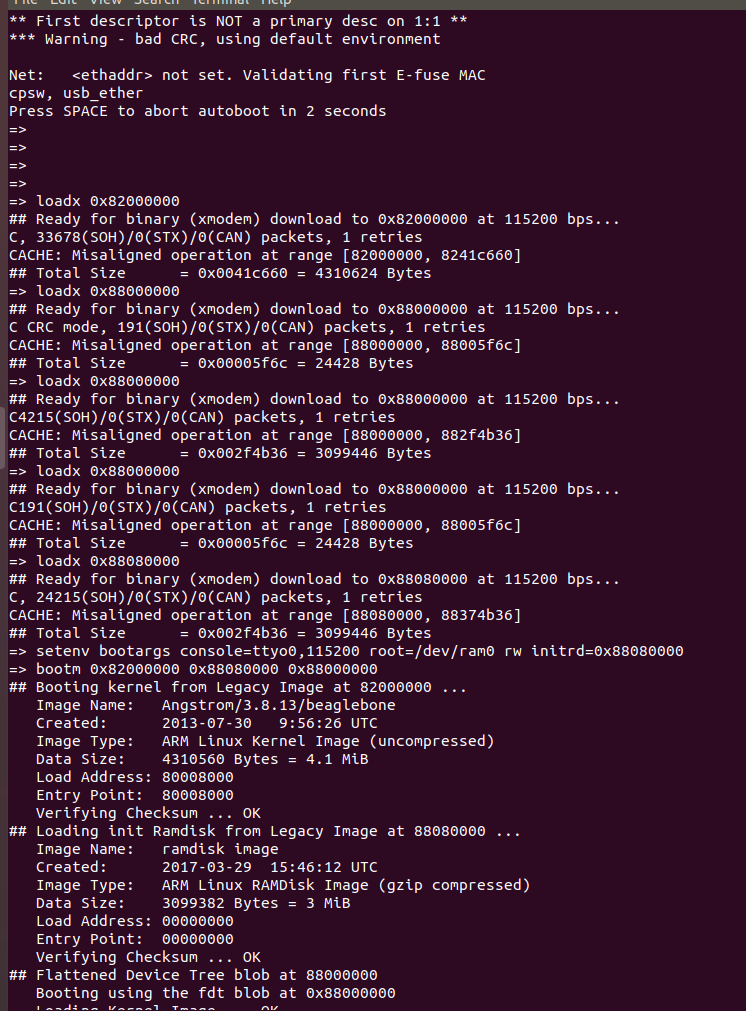


Figure 18: Booting after loading the dtb file

1. Enter the command “setenv bootargs console=ttyO0,115200 root=/dev/ram0 rw initrd=0x88080000”
2. Enter “bootm 0x82000000 0x88000000 0x88080000”

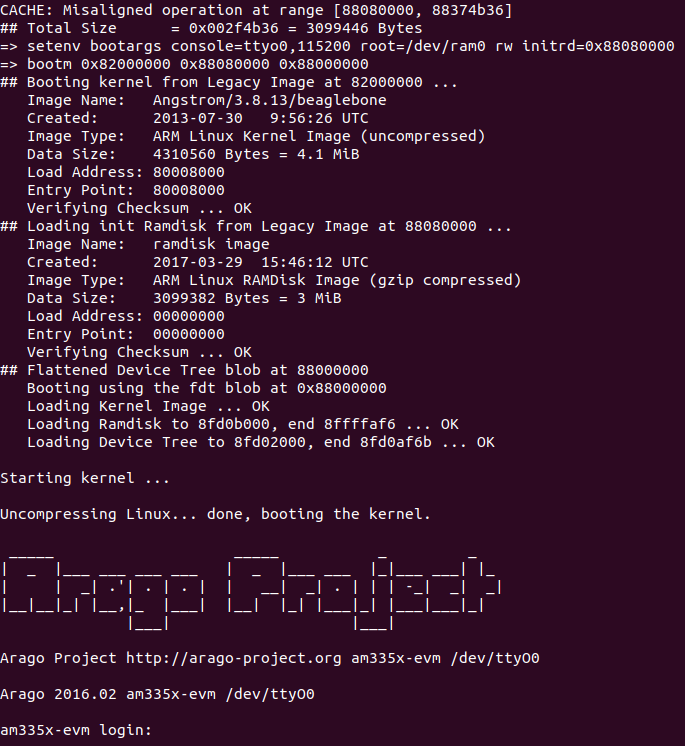


Figure 19: Starting kernel after flashing the latest version

# ACTIVITY 11 - CHANGING THE LOGIN NAME AND LOGO

1. First change the login name in /etc/hostname file of the rootfs partition in the SD card.
2. Next change the logo in /etc/issue file of the rootfs partition in the SD card.
3. Finally boot the board using the SD card.
4. The updated logo appears and will enter the board using the new login name.

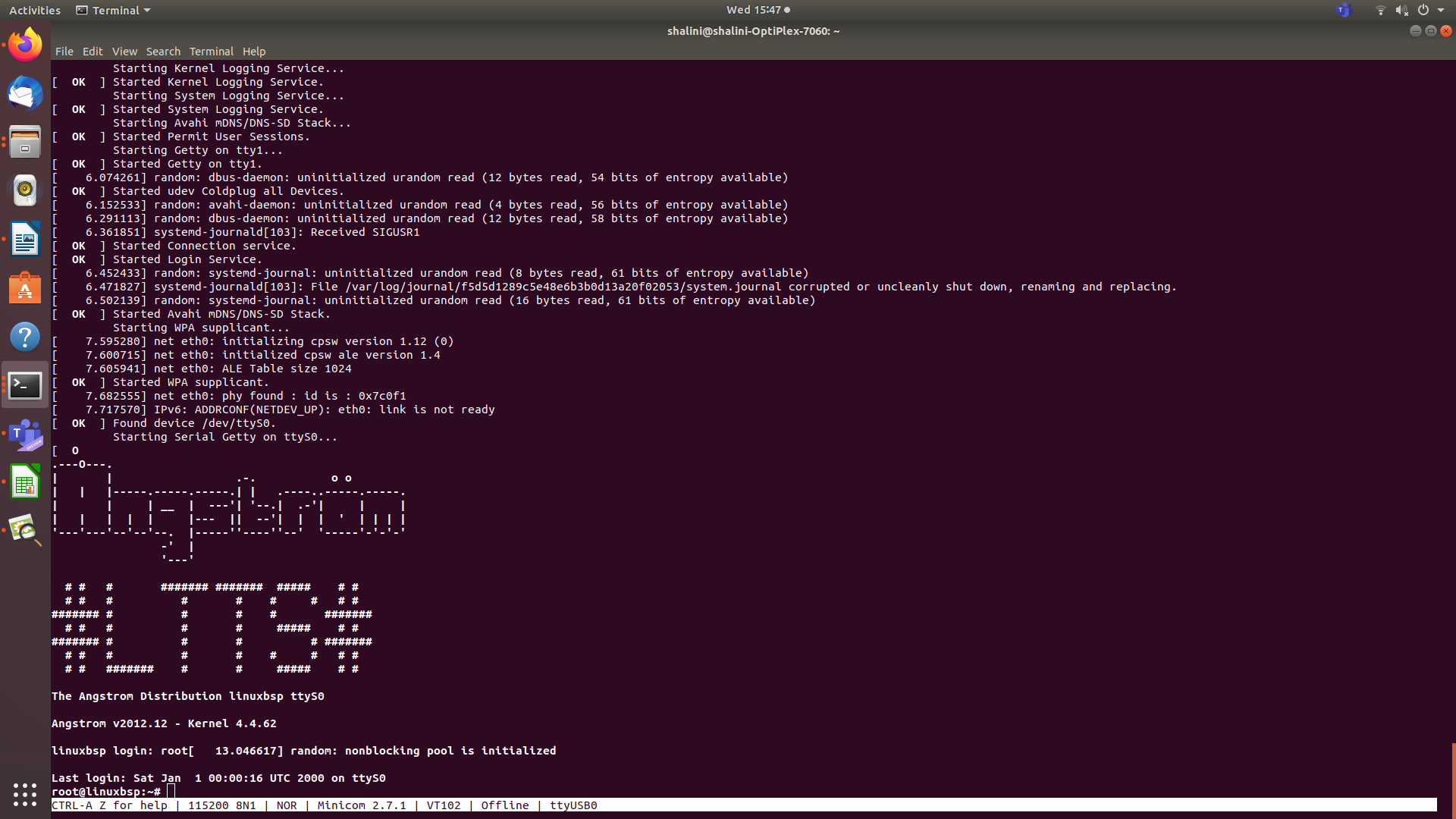


Figure 20: Changing Banner and Login name

# Challenge 1: Make uEnv.txt to Boot from MMC0

In this case, we loaded the boot images from MMC1 interface (eMMC) and used the file system present on the MMC0 (MiroSD card), Challenge for you is to change this uEnv.txt such that boot images are loaded from MMC0(MicroSD).

* 1. Change the uEnv.txt file to print and boot the files required by the user. The uEnv.txt file use for this exercise is put in the image below.
  2. Copy the same file to the rootfs of the SD card and boot from the SD card.
  3. Use the command printenv bootm to check what the command is present in the bootm variable.
  4. Then use the command boot to boot to the board.

**uEnv.txt file:**

mypcip=setenv serverip 192.168.1.2  
 ipaddr=192.168.27.1  
 bootargs=console=ttyO0,115200 root=/dev/mmcblk0p2 rw  
 bootm=echo"Booting from Memory";load mmc 0:2 0x82000000 /boot/uImage;load mmc 0:2 0x88000000 /boot/am335x-boneblack.dtb;bootm 0x82000000 – 0x88000000;

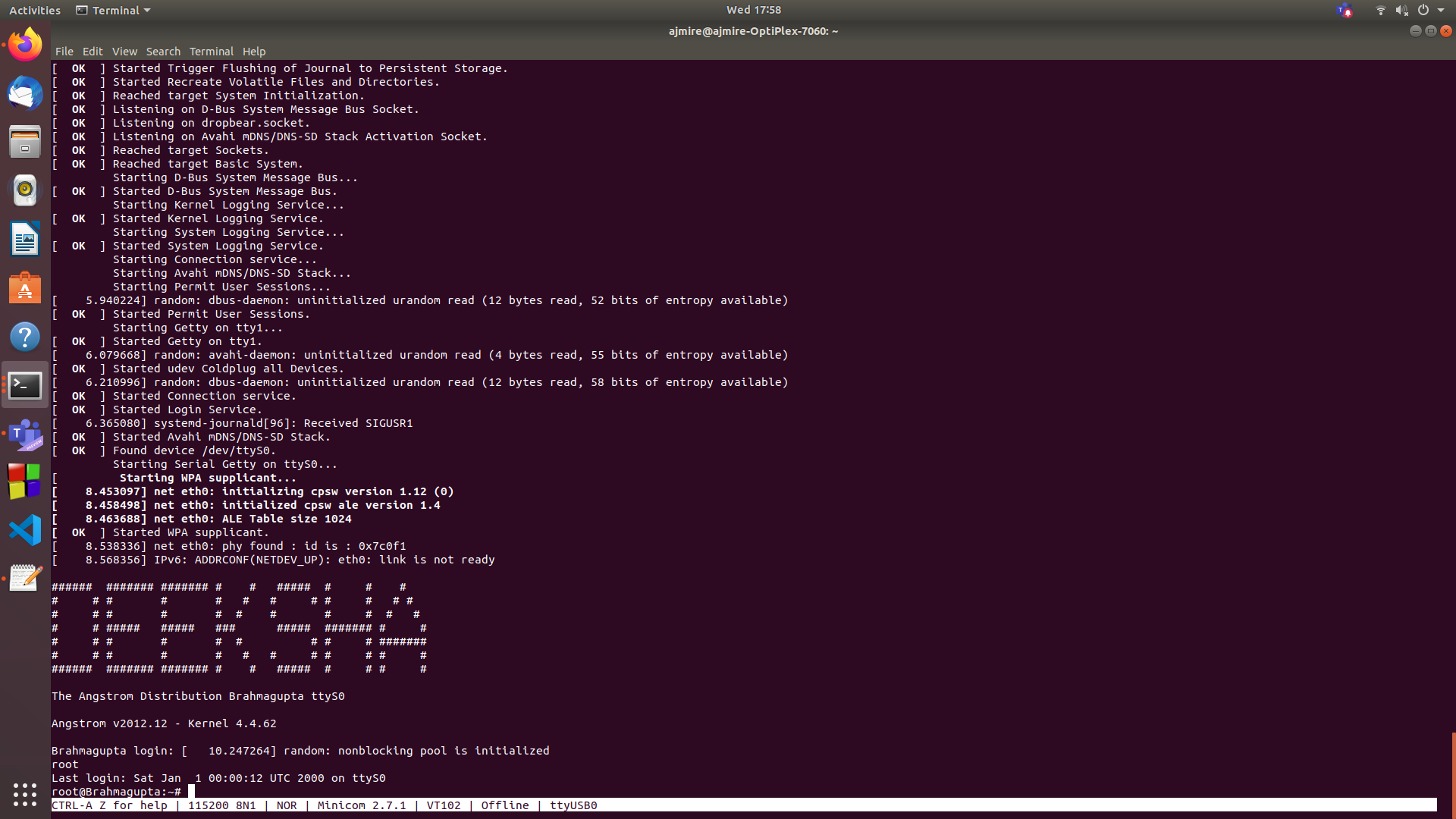


Figure 21: Beagle Bone Black Boot Log

# Challenge 2: uEnvt.txt file to automate TFTP boot

Write a uEnvt.txt file to automate TFTP boot. In this section you learnt how to do the TFTP boot and we are typing tftpboot commands each time we want to download the images. So, the challenge for you is to automate TFTP boot by writing uEnv.txt file. That means the boot images have to be downloaded automatically when you reset your board and should boot properly. Also, dont use the S2(boot) button during power up. The moment you press the reset button or give power to the board it should start fetching "dtb", 'uImage", "initramfs" from TFTP

First to boot the TFTP manually

**In the PC**

* In a new terminal use the command sudo apt install tftpd-hpa
* In the files app, choose other options and choose my computer and select srv and create a folder tftp using the command sudo mkdir tftp.
* Then copy the files initramfs, am335x-boneblack.dtb and uImage given in the serial boot folder.
* Next use the command ifconfig to check the eth0(Ethernet number) and then use the command sudo ifconfig (Ethernet\_number) 192.168.110.10 up.

**In the board:**

* Use sudo minicom and stop the auto boot by clicking some key and enter the uboot.
* Next use the following commands to set up the TFTP
* Setenv serverip 192.168.110.10
  + Setenv ipaddr 192.168.110.20
  + Ping 192.168.110.10 -> this should give alive status
  + Tftp 0x82000000 uImage
  + Tftp 0x88000000 am335x-boneblack.dtb
  + Tftp 0x88080000 initramfs
* To boot next use the command setenv bootargs console=ttyO0,115200n8 root=/dev/ram0 rw initrd 0x88080000
* Bootm 0x82000000 0x88080000 0x88000000
* This will boot till the login.

**To automate:**

* Create a uEnv.txt file and then load it to the BBB
* Use the command loady and press enter. Then Cntrl+A then S and choose y-modem and select the newly created uEnv.txt and load it.
* Then use the command env import –t 0x80200000 342 to import and then use the command run bootm to run the booting process automatically.

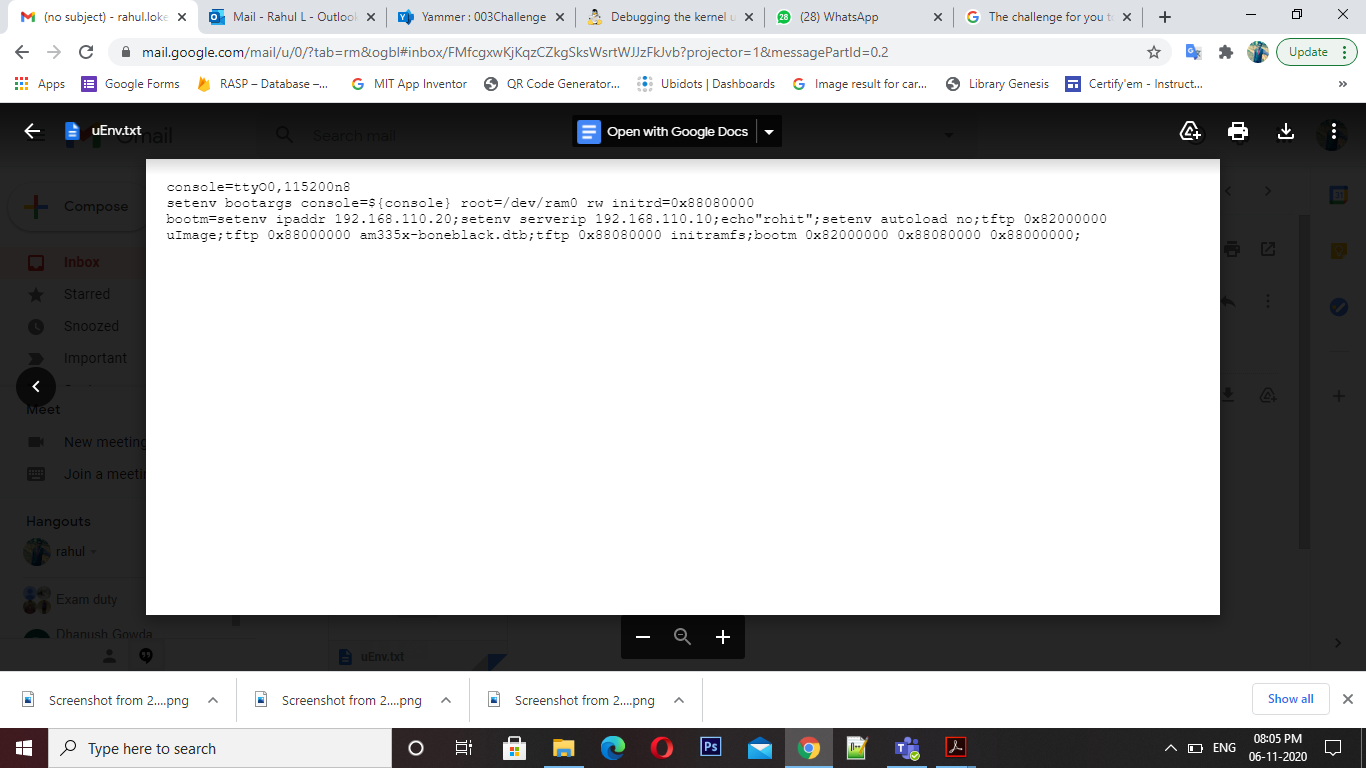
****

Figure 23: uEnv.txt entry

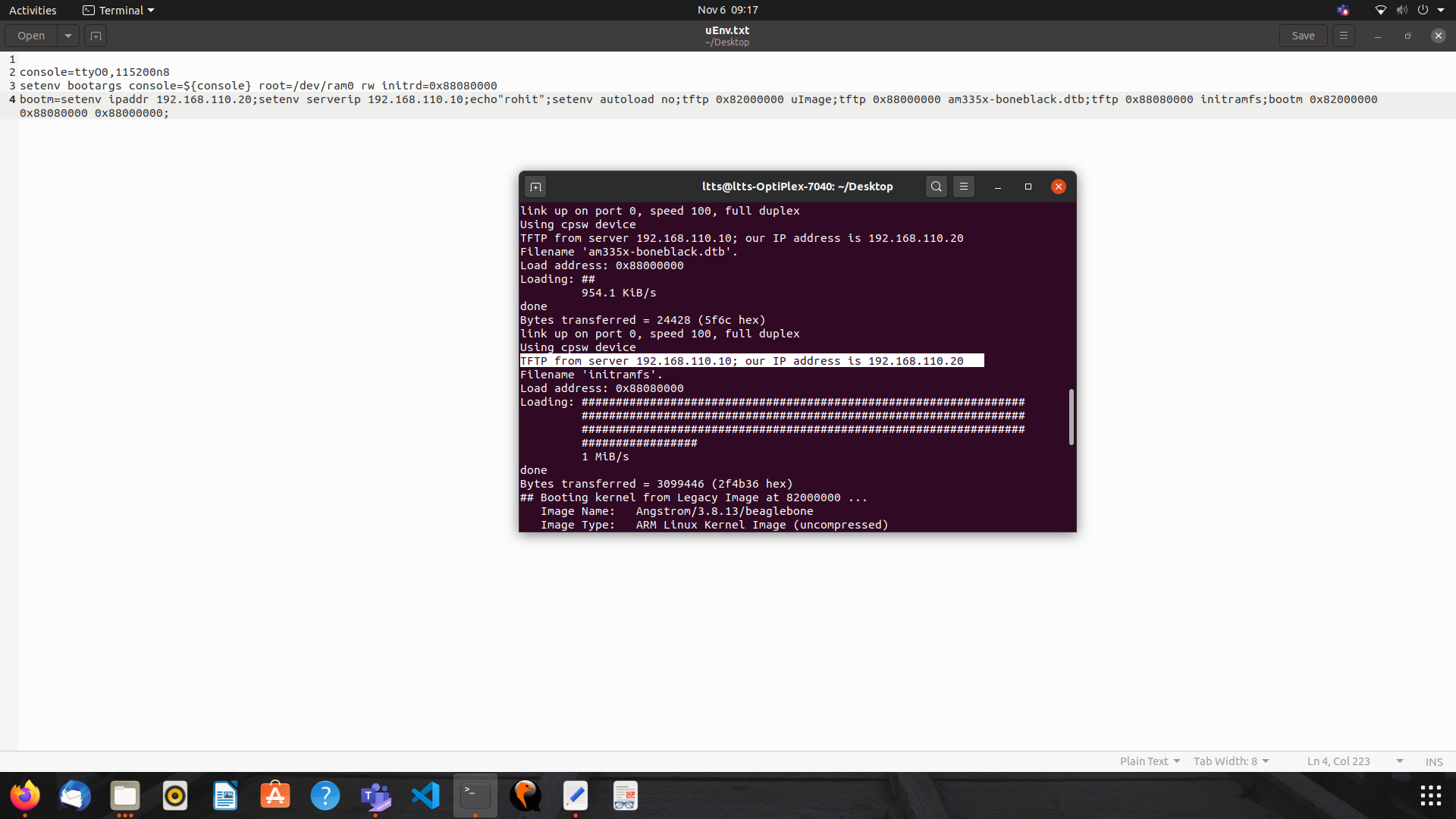
****

Figure 24: IP address and server IP address

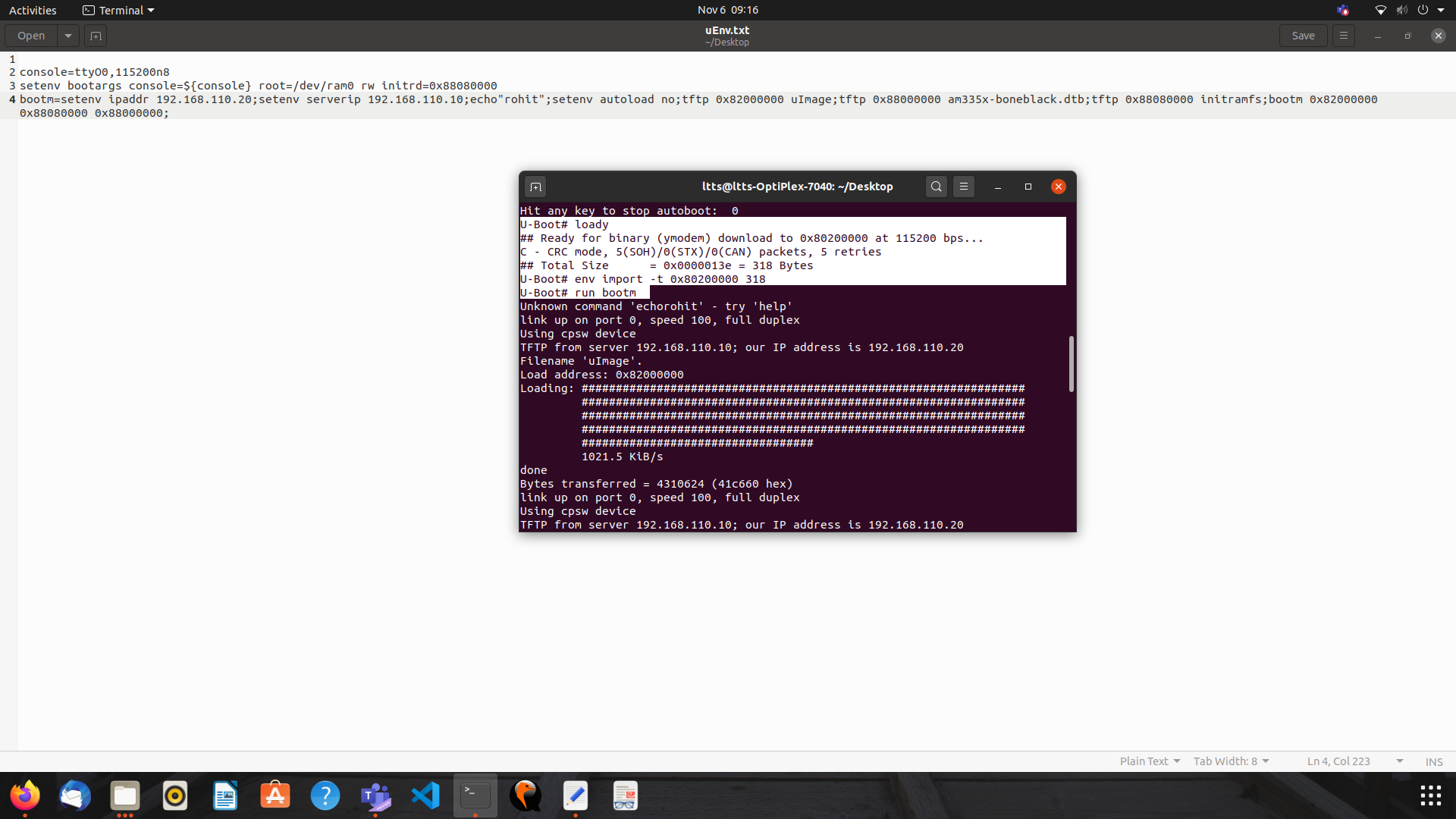
****

Figure 25: Import uEnv and run command

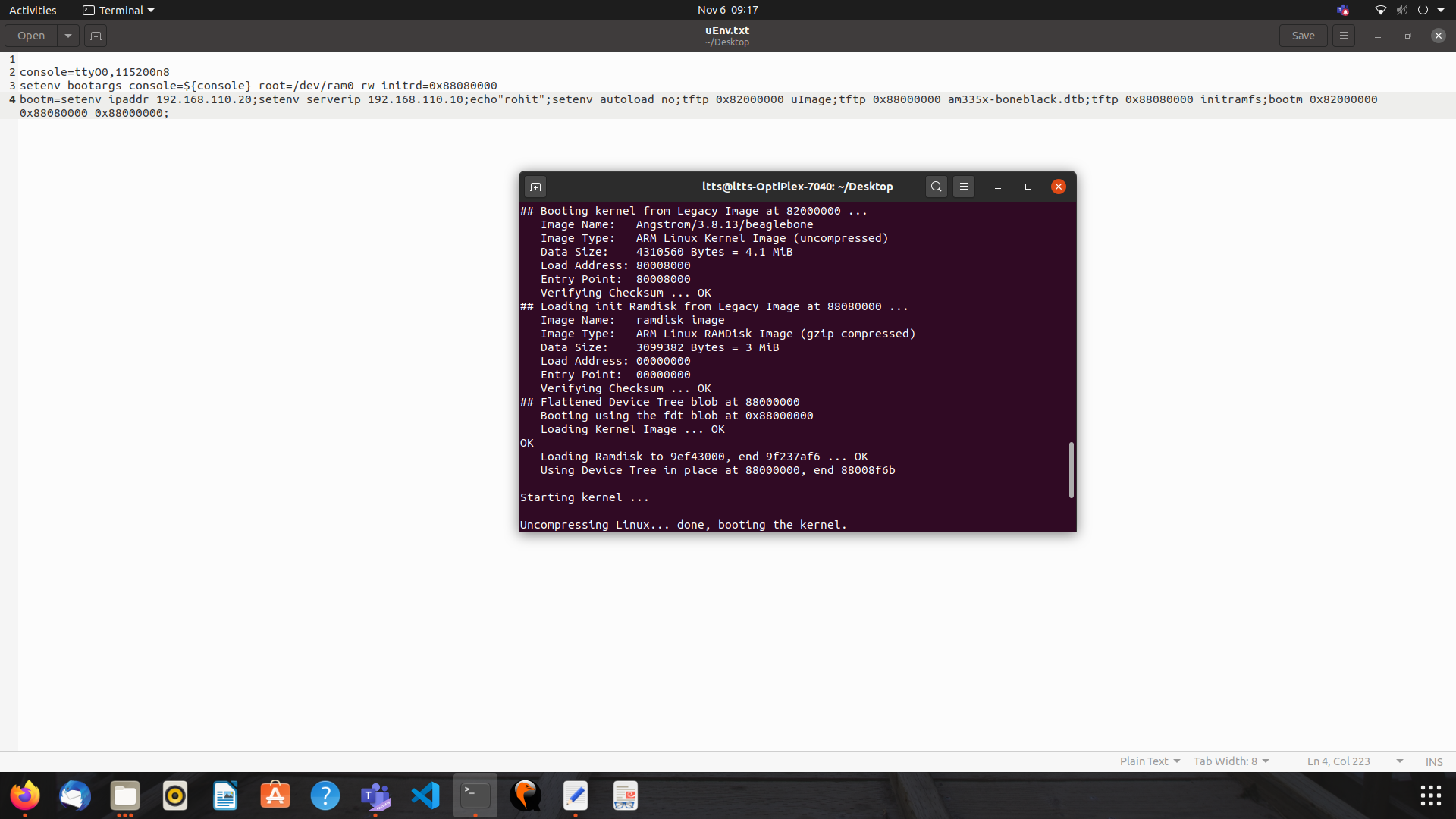
****

Figure 26: Starting kernel autoboot

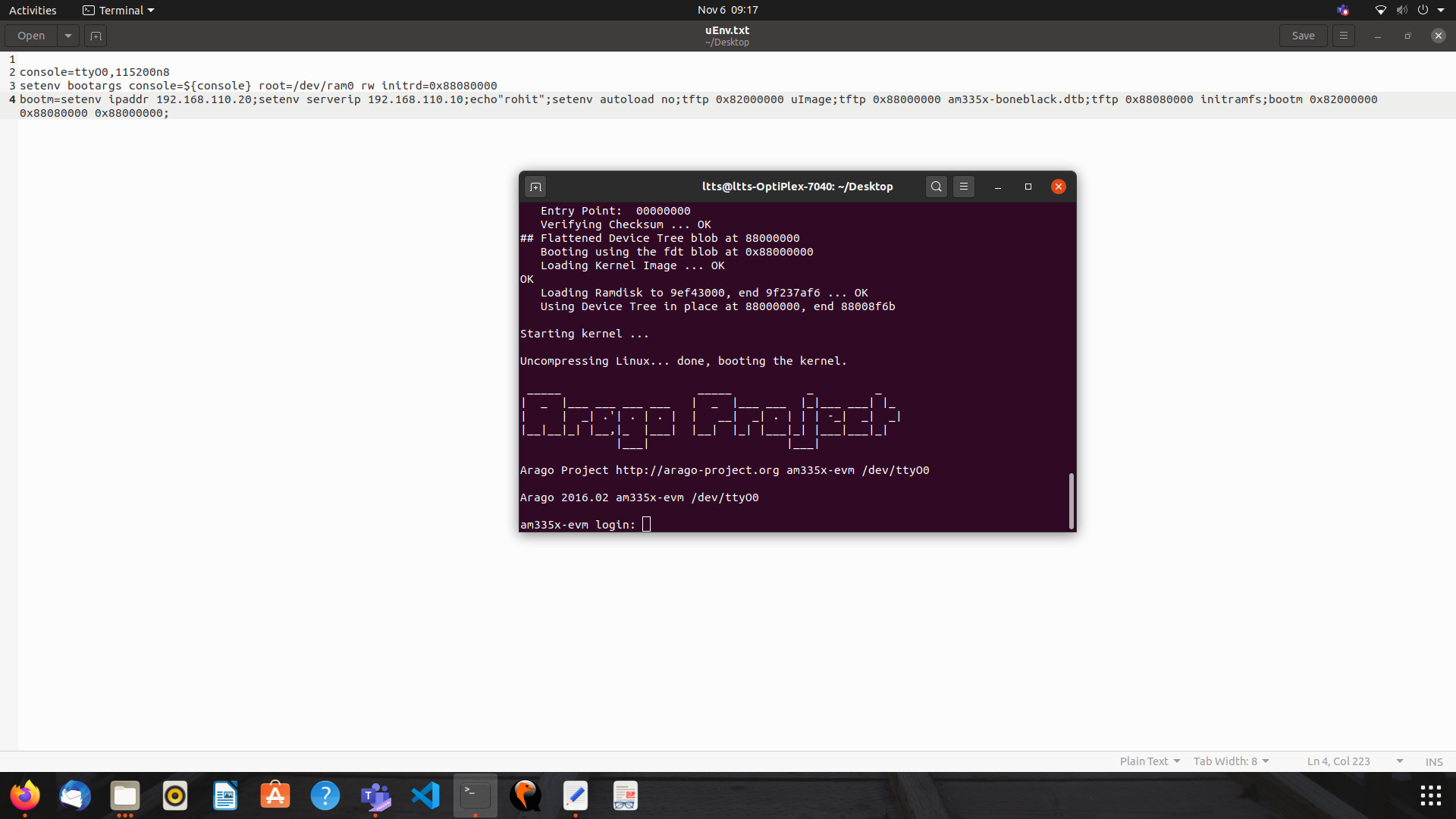
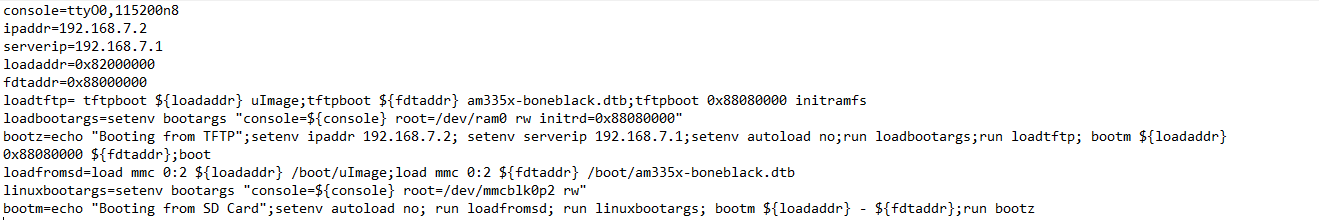
****

Figure 27: Autoboot using tftp

# Challenge 3: Generic uEnv.txt



**Figure 28: Generic uEnv file**

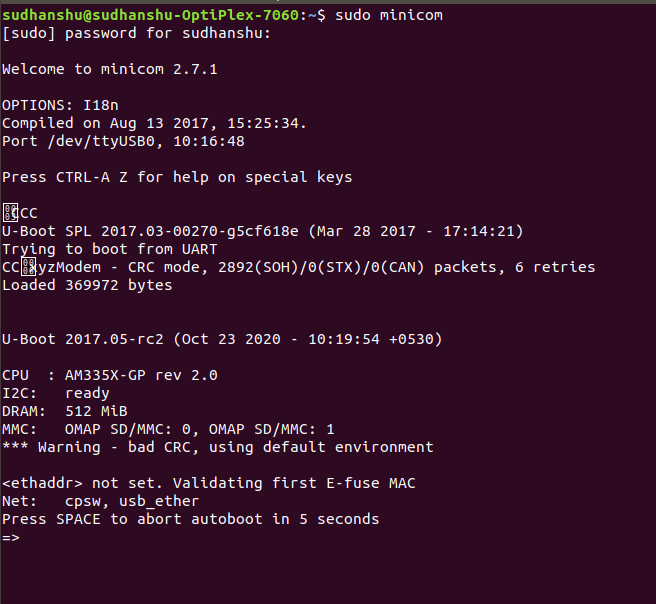
Load the generic uEnv.txt file.

It boots from SD card first, if it is not found then it tries to load from TFTP, it finally loads from eMMC if either of them are not found.

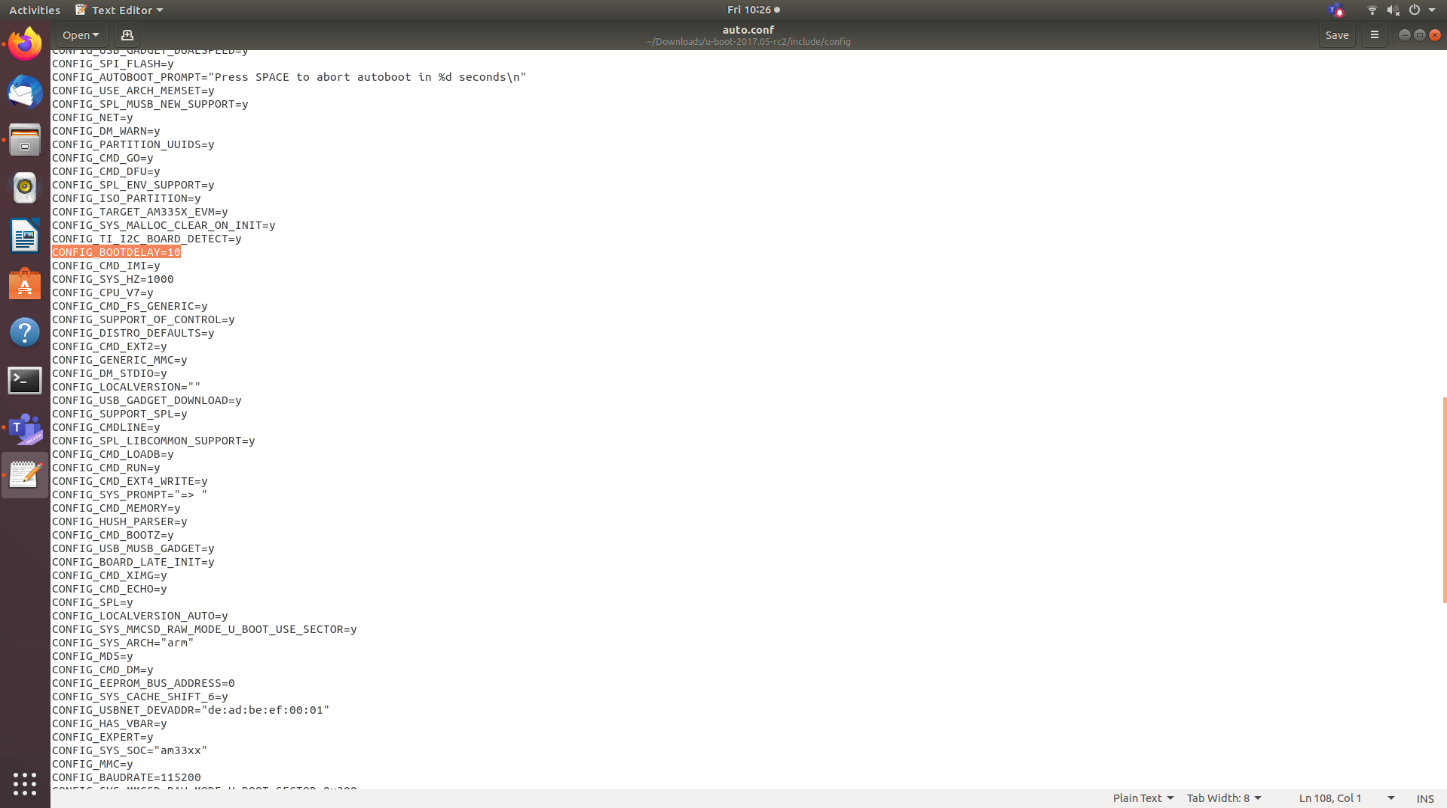
# Challenge 4: Increase the AUTOLOAD timings

When the uboot boots it just waits for 5 seconds before going to auto loading mode (reading uEnv.txt , loading uImage, etc). Within 5 seconds if you press "space" bar then you will get the uboot command prompt. Challenge for you is to increase that autoload timing to 10 seconds and confirm this change by testing on the hardware. you have to change the uboot source code and then rebuild it to generate the uboot image.

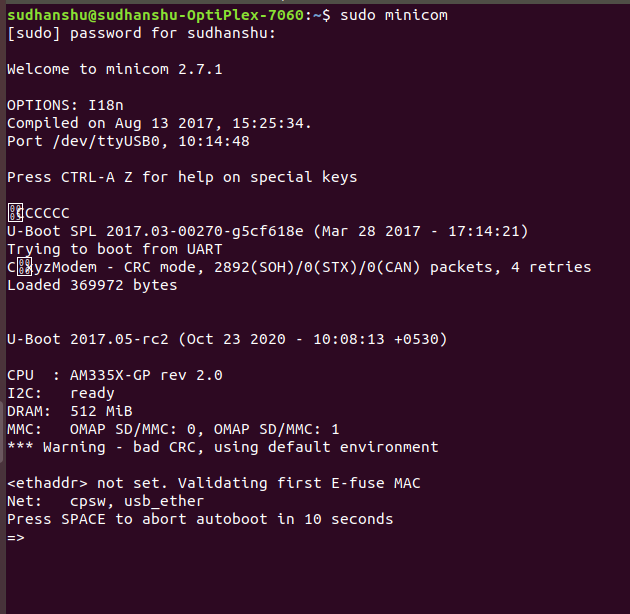
* Use the commands given in the compilation\_commands to change the u-boot.
* STEP 1: distclean: deletes all the previously compiled/generated object files. Use the command make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf-distclean
* STEP 2: apply board default configuration for uboot. Use the command make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- am335x\_boneblack\_defconfig
* STEP 3: run menuconfig, if you want to do any settings other than default configuration. Use the command make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- menuconfig
* In the menu config change the auto boot time to the required value.
* STEP 4: compile. Use the command make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- -j6
* Now load the newly generated u-boot image to the boad using the serial booting to check the changes in the auto booting time.



**Figure 29: The image depicting the auto boot time being 5 seconds**



**Figure 30: The image depicting the auto.config changed to 10 seconds**



**Figure 31: The image depicting the auto boot time being 10 seconds**

# Challenge 6:  Busybox - Dynamic Compilation

In the entire "Busybox" lectures we have used "Static" binaries. That means all the generated utilities/commands of busybox are "statically linked" binaries. If you want to test any applications which are cross compiled by "Dynamic linking " then those applications wont execute on your Busybox file system.

The challenge for you is to reconfigure and re-compile the busybox to generate "dynamically linked\*/8" binaries/utilities and you should also able to test any applications which are dynamically linked.

Generate MLO and u-boot.img by following the steps of u-boot compilation in Commands\_Compilation file (Link is given below)

 Generate uImage and create modules in a directory (RFDYN)by following the steps of u-boot compilation in Commands\_Compilation file (Link is given below)

 Download busybox

o   <https://busybox.net/>

 Apply default configuration

o   make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- defconfig

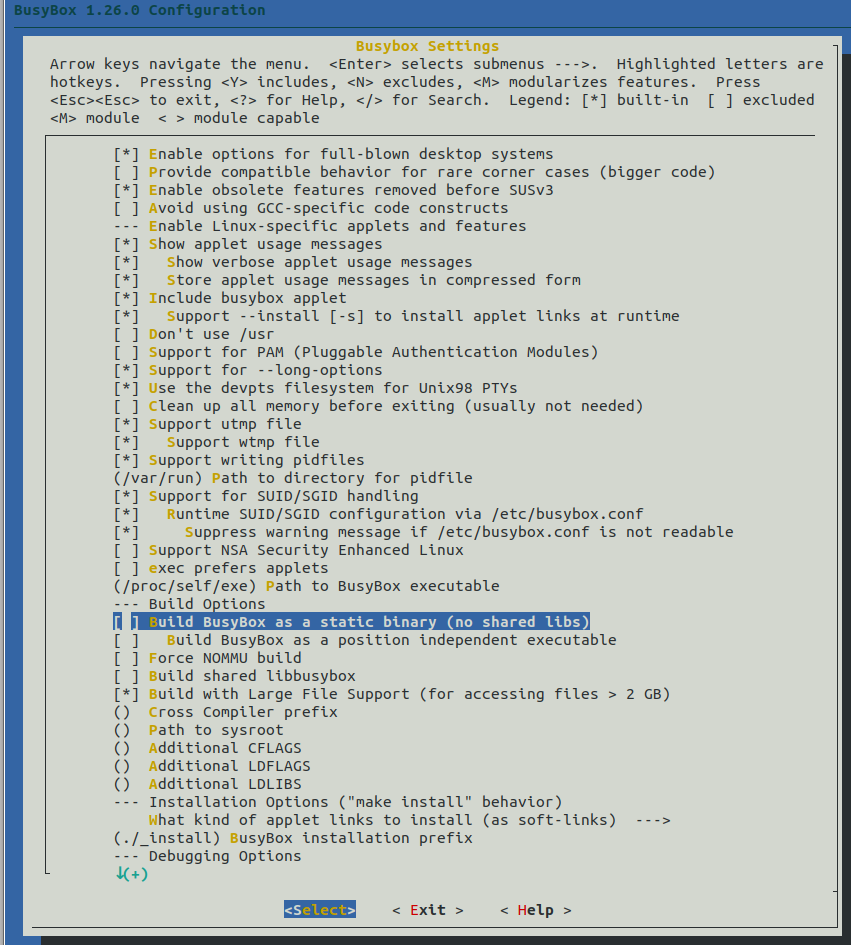
 Change default settings if you want

o   make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- menuconfig

 In the menu configuration uncheck the of build static libraries.

 Generate the busy box binary and minimal file system

o   make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- CONFIG\_PREFIX=<install\_path> install



**Figure 32: Menuconfig**

**References**

<https://en.wikipedia.org/wiki/BeagleBoard>

<https://elinux.org/EBC_Exercise_21a_Boot_Sequence>

<https://wiki.beyondlogic.org/index.php/BeagleBoneBlack_Boot_Process>

<https://sites.google.com/site/manisbutareed/bringing-my-beagles-to-heel/booting-into-linux-on-a-beaglebone>