./

Learning Report -

Embedded Linux

|  |  |  |
| --- | --- | --- |
| **Name** | **PS Number** | **Email ID** |
|  |  |  |

**Details:**

Contents

[Activity-1: Configuration of the Beagle bone Black 3](#__RefHeading___Toc22268_3967773537)

[Evolution of BBB boards 3](#__RefHeading___Toc962_1483389769)

[Activity-2: Comparison of Raspberry pie, Dragon, imx7 Sabre, BBB 5](#__RefHeading___Toc22270_3967773537)

[Activity-3: Comparison of different versions of Beaglebone 9](#__RefHeading___Toc22272_3967773537)

[Activity-4: Pin expansion header of BBB 11](#__RefHeading___Toc22274_3967773537)

[Activity-5: Testing MLO and U-boot image on BBB 11](#__RefHeading___Toc22276_3967773537)

[Different booting stages in Beaglebone Black 12](#__RefHeading___Toc964_1483389769)

[1. ROM Bootloader (RBL) 12](#__RefHeading___Toc966_1483389769)

[2. Secondary Program Loader/ MLO 13](#__RefHeading___Toc968_1483389769)

[3. U-Boot 14](#__RefHeading___Toc970_1483389769)

[Activity-6: Linux boot sequence after the booting. 15](#__RefHeading___Toc972_1483389769)

[Activity-7: Flashing new Debian image in eMMC from SD card 18](#__RefHeading___Toc974_1483389769)

[Activity-8: Establishing a connection from Beaglebone to local server 20](#__RefHeading___Toc976_1483389769)

[Activity-9: Creating own uEnv.txt file 23](#__RefHeading___Toc978_1483389769)

[Activity-10: Changing the banner and login name 24](#__RefHeading___Toc980_1483389769)

Table of Figures

[Figure 1: Serial device select 12](#Figure!0|sequence)

[Figure 2: Booting Process 13](#Figure!1|sequence)

[Figure 3: Booting till U-Boot 14](#Figure!2|sequence)

[Figure 4: Loading u-boot 16](#Figure!3|sequence)

[Figure 5: Loading from uEnv.txt 17](#Figure!4|sequence)

[Figure 6: Booting to Kernel Level complete 18](#Figure!5|sequence)

[Figure 7: Flashing debian image into SD card 19](#Figure!6|sequence)

[Figure 8: Booting from eMMC 20](#Figure!7|sequence)

[Figure 9: Printing the added server list 21](#Figure!8|sequence)

[Figure 10: ifconfig details 22](#Figure!9|sequence)

[Figure 11: Server connection established 23](#Figure!10|sequence)

[Figure 12: Changed banner and login name 24](#Figure!11|sequence)

Index of Tables

Table 1: Comparison of Raspberry pie, Dragon, imx7 Sabre, BBB 7

Table 2: Comparison of different versions of Beaglebone 9

# Activity-1: Configuration of the Beagle bone Black

Step by step configuration of the board and set up in the window as well as in Linux

1. Connect the Beaglebone Black Board (BBB) to router or network via Ethernet cable.
2. Connect the BBB to your computer via the USB cable (LEDs will Glow).
3. Download and install the USB drivers compatible for your system from the beagleboard website (beagleboard.org)
4. Connect BBB over USB and open 192.168.7.2 USB IP address
5. Reach out to cloud9 IDE.
6. Get the Ethernet IP address. For this open a new terminal on cloud9 IDE. Enter the command “ifconfig”.
7. Find out the IP address.
8. Next we have to remotely login to BBB through ssh client (eg: PuTTY)
9. Open the PuTTY and enter the Ethernet IP address into it and click open. This opens up a Linux terminal window.
10. The default login is root and there is no password. Now you are interacting with your beagle bone board and the setup is complete
11. To set up a GUI for the BBB or to get a remote desktop for the board, follow the steps:
12. Download TightVNC compatible for your system.
13. Latest version of Beaglebone black will have the TightVNC installed. But if it’s not installed, go to the terminal and enter command “sudo apt –get install tightvncserver”.
14. After installing, enter the command “tightvncserver” for configuration and setup the password.
15. To set the screen size “Vncserver :1 –geometry 1280/800 –depth 24 –dpi 96”
16. Now navigate to TightVNC and enter “Ethernet IP address :1” and click connect.
17. Now the GUI for the board appears.

## Evolution of BBB boards

**1. Revision C**

This revision increases the eMMC from 2GB to 4GB. This was mainly due to:

1) Complaints from the community about lack of space left in the eMMC.

2) For those worried about their eMMC wearing out, the added space will help in the area of moving the data around to prevent wear out. Assuming of course you don't try and use it all.

3) Concerns over the long-term availability of the 2GB device. 4GB is currently the low end of the offering. This also gives us two sources.

**2. Revision B**

This version moves to the AM3358BZCZ100 processor from AM3359AZCZ100.

No changes in features or operation of the board resulted from this change.

**3. Revision A6A**

No changes in features or operation of the board.

1) Added optional zero-ohm resistor to tie GND\_OSC1 to system ground.

2) Changed C106 to a 1uF capacitor.

3) Changed C24 to a 2.2uF capacitor. This extends the reset signal to solve an issue where some boards would not boot on power up.

4) Removed R9 and installed R8.

**4. Revision A6**

No changes in features or operation of the board.

1) Noise issues were observed in other designs where the clock oscillator was getting hit due to a suspected issue in ground bounce. A zero-ohm resistor was added to connect the OSC\_GND to the system ground.

2) Moved the enable for the VDD\_3V3B regulator to VDD\_3V3A rail. Change was made to reduce the delay between the ramp up of the 3.3V rails. No evidence of this being an issue, but it really needs to be as close to the same as possible.

**5. Revision A5C**

Production had some fallout of boards when running the HDMI tests in the previous production run. Resistor values were tweaked to improve the test results.

No changes in features or operation of the board.

1) Changed R46,R47,R48 to a 0 ohm.

2) Changed R45 to a 22 Ohm.

**6. Revision A5B**

1) Updated the PCB to incorporate the modification that was being done on Rev A5A. There is NO DIFFERENCE AT ALL in functionality between REV A5A and REV A5B.

2) Made the LEDs dimmer for those that could not sleep due to the brightness of the LEDs.

**7.Revision A5A**

1) Boards are built using the XAM3359AZCZ100 processor.

2) PCB Change: LCD noise issue was resolved by adding 47pf bypass caps on some of the LCD signals.

3) PCB Change...Added access to four battery charger signals on the TPS65217 (TS=Temperature Sense, BAT=Battery connection, BATT\_SENSE=Battery voltage pin, GND=Ground). Pins are not populated but the four signals are in a 2x2 .1x.1 spacing.

4) PCB Change...Added a power button which allows for wake up, power down, and sleep options. It also provides the ability to alert the processor before powering down to provide an orderly shutdown. It is expected that SW will be used in conjunction with the switch to control the various power modes and transitions from one to the other. By holding the button down for 8 seconds, it will force a power down of the board.

5) Added a 100K pull down resistor from J1 pin 1 to J1 pin 4 to fix the unterminated serial port issue.

**8. Revision A4B**

Added a 100K pull down resistor between pins 1 and 4 of J1 to fix the serial port issue.

Revision A4A

Incorporated the capacitors to fix the noise issue on the display

**9.Revision A4**

First prototype release version of the board. Limited distribution. One notable issue here is that the board has an AM3352 processor instead of an AM3359, despite how the part is marked. Part was mismarked as an AM3359. The SGX and PRU are not operational.

# Activity-2: Comparison of Raspberry pie, Dragon, imx7 Sabre, BBB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Raspberry Pi** | **Beaglebone Black** | **Dragonboard 410 C** | **imx7 Sabre** |
| Processor Type | ARM11 processor. | It uses ARM Cortex-A8 processor. | ARM Cortex A53, Quad-core CPU | Two Arm® Cortex®-A7 core operating up to 1 GHz  Single Arm Cortex -M4 core operating up to 200 MHz |
| RAM | For the functioning of raspberry pi, 512 MB SDRAM is used. | For the functioning of beaglebone black, 512 MB DDR3L is used. | 1GB LPDDR3 SDRAM | 1 GB DDR3 |
| Processor Speed | It uses 700 MHz for processing. | It uses 1 GHz for its processing. | @ 533MHz | 533 MHz |
| Operating Systems | Raspbian (Recommended), Ubuntu, Android, ArchLinux, FreeBSD, Fedora, RISC OS, others… | Angstrom (Default), Ubuntu, Android, ArchLinux, Gentoo, Minix, RISC OS, others… | Android 5.1 (Lollipop) on Linux Kernel 3.10,  Linux based on Debian 8.0,  Open Embedded,  Ubuntu Core,  Windows 10 IoT Core |  |
| Flash | It has dedicated SD Card socket for loading operating system. | It uses 4GB (micro SD) for loading OS and data storage. | It uses 8GB (micro SD) for loading OS | 1 SD socket for boot code |
| Storage |  |  | 8GB eMMC 4.51 on board storage and MicroSD card slot | eMMC expansion footprint  NAND flash expansion footprint  QSPI flash expansion footprint |
| Min Power | It requires a power supply of 700mA (3.5W). | It requires min power of 210mA (1.05W) for its functioning. | 8V-18V@3A, Plug specification is inner diameter 1.7mm and outer diameter 4.8mm | Power Management: PF3000 PMIC |
| Network connectivity | Wi-Fi 802.11n / Bluetooth 4.1 & BLE / 10/100 Ethernet | 1 10/100 Mbps Ethernet | On-board Wi-Fi 802.11 b/g/n 2.4GHz | 802.11 a/b/g/n/ac Wi-Fi® on board  Bluetooth V4.0 + EDR on board  Dual Ethernet (both 1 GigE) on board |
| GPIO Pins | It has 12 GPIO pins. | It has 69 GPIO pins. | It has 12 GPIO pins. |  |
| Dev IDE | It uses IDLE, Scratch, Squeak/Linux to perform tasks. | It uses Python, Scratch, Squeak, Cloud9/Linux to perform a particular task. |  |  |
| USB | It has 2 USB 2.0 on board. | It has 1 USB 2.0 on its board. | 2 x USB 2.0 Host 1 x USB 2.0 OTG | 1 USB host connector  1 micro USB OTG connector |
| Audio | Supports HDMI, Analog audio output | It uses Analog output for audio. | PCM/AAC+/MP3/WMA, ECNS, Audio+ post-processing (optional) |  |
| Video | It supports HDMI, Composite output for video. | No such specific video output. | 400MHz graphic processor capable of handling 1080 HD video |  |
| UART | It uses 1 UART to transmit and receive serial data. | It uses 5 UART to transmit and receive serial data. | It uses 1 UART to transmit and receive serial data. |  |
| No. of I/O pins | It has 8 Digital, 0 Analog pins. | It has 65 Digital, 7 Analog pins. |  |  |
| Peripherals | 2 USB Hosts, 1 Micro-USB Power, 1 10/100 Mbps Ethernet, RPi camera connector | 1 USB Host, 1 Mini-USB Client, 1 10/100 Mbps Ethernet | HDMI Full-size Type A connector, one micro USB (device mode only), two USB 2.0 (host mode only), micro SD card slot | Dual Ethernet (both 1 GigE) on board  1 SD socket for boot code  1 mikroBus™ socket  1 USB host connector  1 micro USB OTG connector  Full Mini PCIe socket  SIM card slot  CAN (DB-9) |
| Base Price(in dollars) | 35 | 45 | 67.50 | 249 |

Table 1: Comparison of Raspberry pie, Dragon, imx7 Sabre, BBB

# Activity-3: Comparison of different versions of Beaglebone

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | PocketBeagle | BeagleBone Black | BeagleBone Blue | BeagleBone AI |
| Processor | AM3358  ARM Cortex-A8 | AM3358  ARM Cortex-A8 | AM3358  ARM Cortex-A8 | AM5729  2x ARM Cortex-A15 |
| Maximum Processor Speed | 1GHz | 1GHz | 1GHz | 1.5GHz |
| Co-processors | 2x200-MHz PRUs, ARM Cortex-M3, SGX PowerVR | 2x200-MHz PRUs, ARM Cortex-M3, SGX PowerVR | 2x200-MHz PRUs, ARM Cortex-M3, SGX PowerVR | 4x200-MHz PRUs, 2x ARM Cortex-M4, 2x SGX PowerVR, 2x HD video |
| Analog Pins | 2 (3.3V), 6 (1.8V) | 7 (1.8V) | 4 (1.8V) | 7 (3.3V) |
| Digital Pins | 44 (3.3V) | 65 (3.3V) | 24 (3.3V) | 72 (3.3V) (7 shared with analog) |
| Memory | 512MB DDR3 (800MHz x 16), microSD card slot | 512MB DDR3 (800MHz x 16), 4GB on-board storage using eMMC, microSD card slot | 512MB DDR3 (800MHz x 16), 4GB on-board storage using eMMC, microSD card slot | 1GB DDR3 (2x 512Mx16, dual-channel), 16GB on-board storage using eMMC, microSD card slot |
| USB | USB 2.0 480Mbps Host/Client Port, USB 2.0 on expansion header | USB 2.0 480Mbps Host/Client Port, USB 2.0 Host Port | USB 2.0 480Mbps Host/Client Port, USB 2.0 Host Port | USB 3.0 5Gbps Host/Client Port, USB 2.0 Host Port |
| Network | add-ons | 10/100 Ethernet | 2.4GHz WiFi, Bluetooth, BLE | Gigabit Ethernet, 2.4/5GHz WiFi, Bluetooth, BLE |
| Video | SPI displays | microHDMI, cape add-ons | SPI displays | microHDMI, cape add-ons |
| Supported Expansion Interfaces | 3x UART, 4x PWM, 2x SPI, 2x I2C, 8x A/D converter, 2x CAN bus (w/o PHY), 2x quadrature encoder, USB | 4x UART, 12x PWM/Timers, 2x SPI, 2x I2C, 7x A/D converter, 2x CAN bus (w/o PHY), LCD, 3x quadrature encoder, SD/MMC, GPMC | 4x UART, 2-cell LiPo, 2x SPI, I2C, 4x A/D converter, CAN bus (w/ PHY), 8x 6V servo motor, 4x DC motor, 4x quadrature encoder | 4x UART, 12x PWM/Timers, 2x SPI, 2x I2C, 7x A/D converter, CAN bus (w/o PHY), LCD, 3x quadrature encoder, SD/MMC |
| On-board Sensors | - | - | 10 degree of freedom IMU (accelerometer, gyroscope, magnetometer, thermometer), barometer/thermometer | on-die temperature |
| MSRP | $25 | $49 | $79 | $99 |

Table 2: Comparison of different versions of Beaglebone

# Activity-4: Pin expansion header of BBB

Link to excel

# Activity-5: Testing MLO and U-boot image on BBB

* Step 1: Download gparted using the command
  + sudo apt install gparted
* Step 2: Insert card reader and select /dev/sdb(gb) and open gparted and make two partitions.
* Step 3: Right click on unallocated and click on New and allocate new size as per SD card and give file system as fat16 and give label BOOT.
* Step 4: Right click on unallocated and then remaining space will be allocated to this partition by default. Give file name as ext3/ext4 and give label as ROOTFS.
* Step 5: Right click on BOOT and select manage flags then select boot option and then close.
* Step 6: 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 7: Create a workspace copy MLO file in BOOT and Workspace. Then unmount the SD card.
* Step 8: Then connect SD Card in board and open the terminal.
* Step 9: Install minicom by using the command
  + sudo apt install minicom
  + sudo minicom -s for setup changes
* Then select on serial device as /dev/ttyUSB0

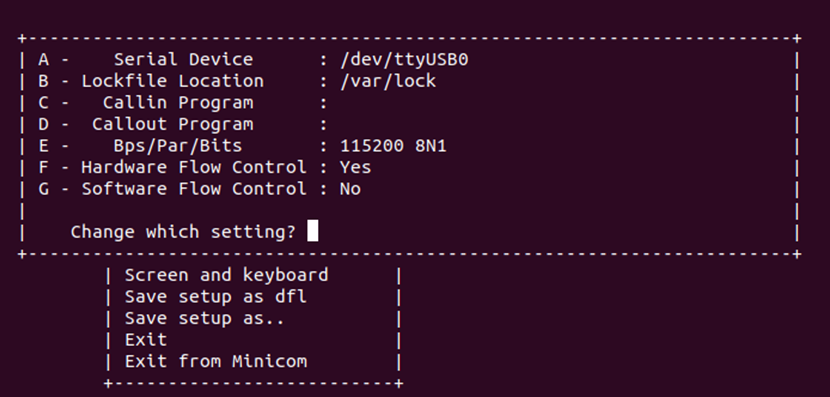


Figure 1: Serial device select

* Step 10: Then enter to save and exit after saving it as the default configuration.
* Step 11: Copy the u-boot.img to the BOOT partition and repeat the step for boot-up.
* Step 12: Open the minicom again, and boot the BBB with the S2 button pressed.

## Different booting stages in Beaglebone Black

### 1. ROM Bootloader (RBL)

The ROM boot loader for the BBB is located at the AM335x ROM. The AM335x have an internal RAM memory of 128KB. As soon as the power is applied to the device, the code at this location is executed. When we apply power to the SOC, it does some system level initializations and then goes for the initialization of the watchdog. The watchdog is configured for 3 minutes. If the watchdog is not fed for some time, watchdog senses some trouble and it will reset the system. So here, if even after three minutes the booting is not further connected to the SPL/MLO, the watchdog resets the system and the sequences restarts from the beginning.

After it does the watchdog initializations, it then moves on to initialize the PLL and clock configurations. PLL stands for phase lock loops. To be simple, PLLs are the one who will be providing different frequencies from a crystal input to various peripheral. In the case of BBB, we are giving a clock input of 24 MHz and we are able to generate clock frequencies of around up to 800 MHz. Then booting process is started.

In the booting, the SOC will look for the configuration of SYSBOOT pins and determine from where is the bootable image to be loaded. The AM335x have the capability of loading bootable image from various sources like the MMC interface, USB interface, UART interface etc. Based on the pins configuration and switch select, the device will then jump to the respective loader. In our case, the device will move on to the SPL/MLO.

If the device is not able to find any bootable image, the device will go into a dead loop and after three minutes, the watchdog will reset the system.

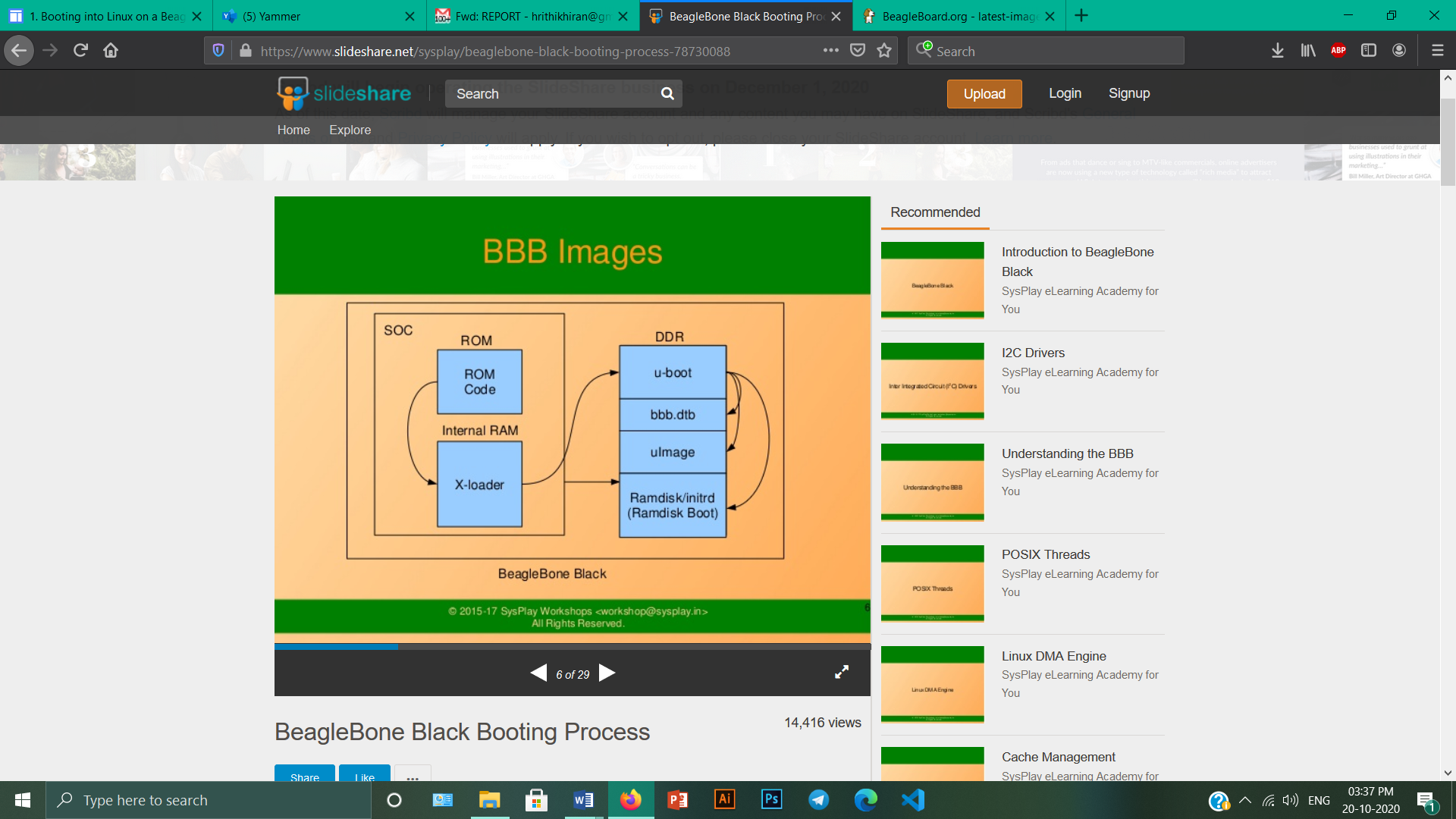


Figure 2: Booting Process

### 2. Secondary Program Loader/ MLO

Now, we have finished the RBL and the SPL was successfully loaded it runs out of the internal SRAM of the SOC. The SOC will do the initializations and preparations for the U-Boot that is the third stage bootloader to be executed.  It is also possible to modify the PLL in order to derive a desired frequency of clock source from the second stage bootloader. It also initializes the DDR memory because the Linux Kernel is going to be executed from this memory.

The SPL additionally does an important operation known as pin muxing. Suppose the U-Boot is to be loaded from the MMC interface. Then the pins of the SOC have to be configured for supporting the MMC interface. The pins of the SOC can be used for various purposes based on the need of the user. This is known as multiplexing or muxing in the embedded systems.

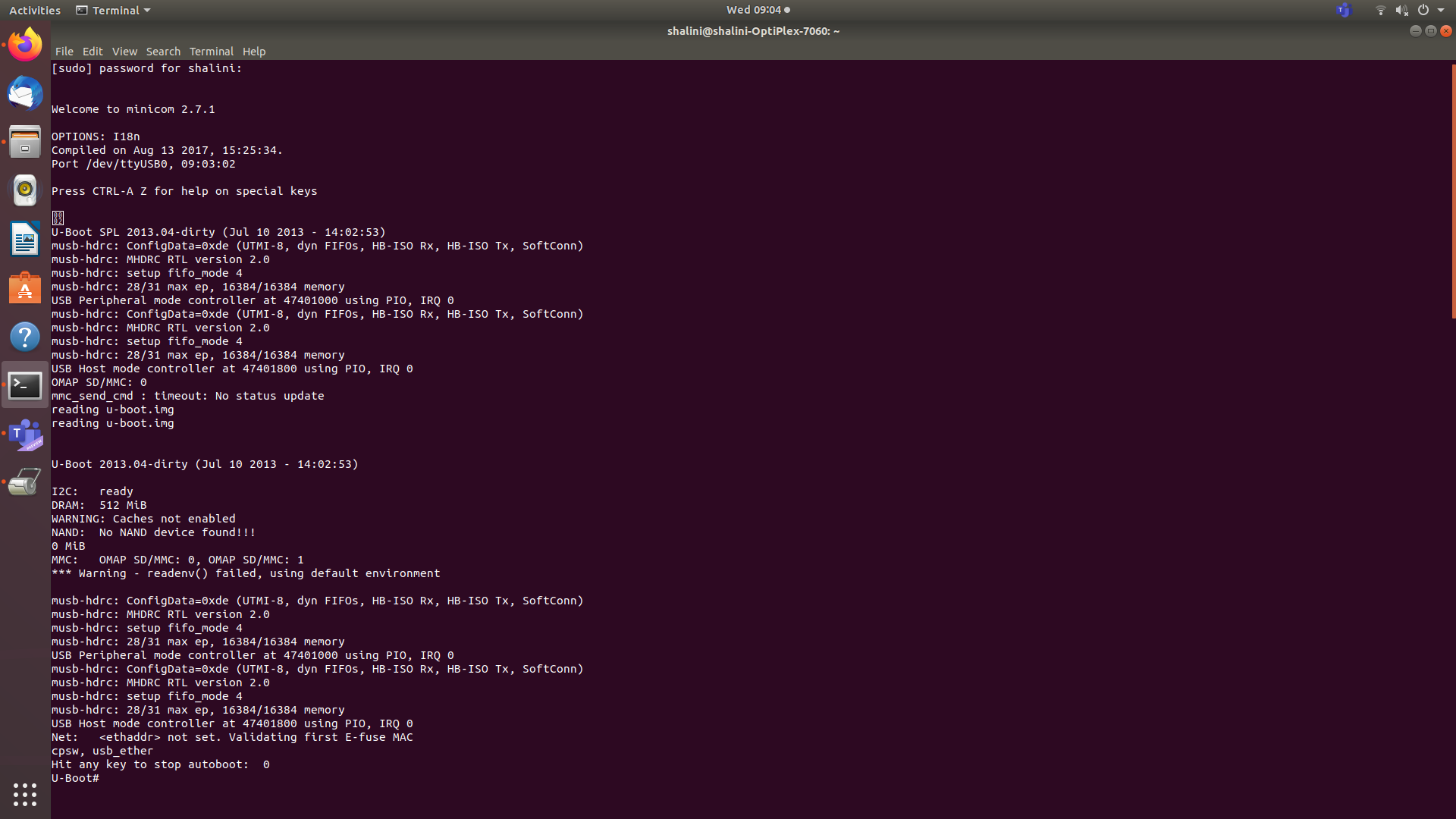


Figure 3: Booting till U-Boot

After this, the SPL checks for the U-Boot. It searches for the U-Boot image file called uboot.img. After these processes, the U-Boot image is copied into the DDR memory and the control is passed to the U-Boot. This is our third stage bootloader.

The RBL could not copy the U-Boot to the internal SRAM because the size of the internal SRAM is only 128KB. So we are using a second stage bootloader to copy the U-Boot to the DDR. Also, the U-Boot cannot be loaded directly to the DDR by the RBL, because DDR is an external memory. The SOC doesn’t know which DDR is being used.

### 3. U-Boot

Now the control have reached the third stage bootloader that is the U-Boot. This guy will load the Linux kernel to the DDR memory. In order for the U-Boot to load the Linux Kernel, we should tell the U-Boot where the Linux Kernel is located, through what interface it is accessible etc. With these information, the U-Boot will load the Linux Kernel into the DDR memory of the BBB.

We write these information in a file called uEnv.txt and the U-Boot will read this text file and find out from where it can load the Linux Kernel and to which address of the DDR memory is the Kernel to be loaded.

The U-Boot checks for a file called uImage. UImage is actually a combination of U-Boot header, which have information about the image like the architecture, OS name etc. and something called zImage, which is actually a compressed version of Linux Kernel.

This is then decompressed and the Linux Kernel is loaded.

# Activity-6: Linux boot sequence after the booting.

* By default, the ROM 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 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)

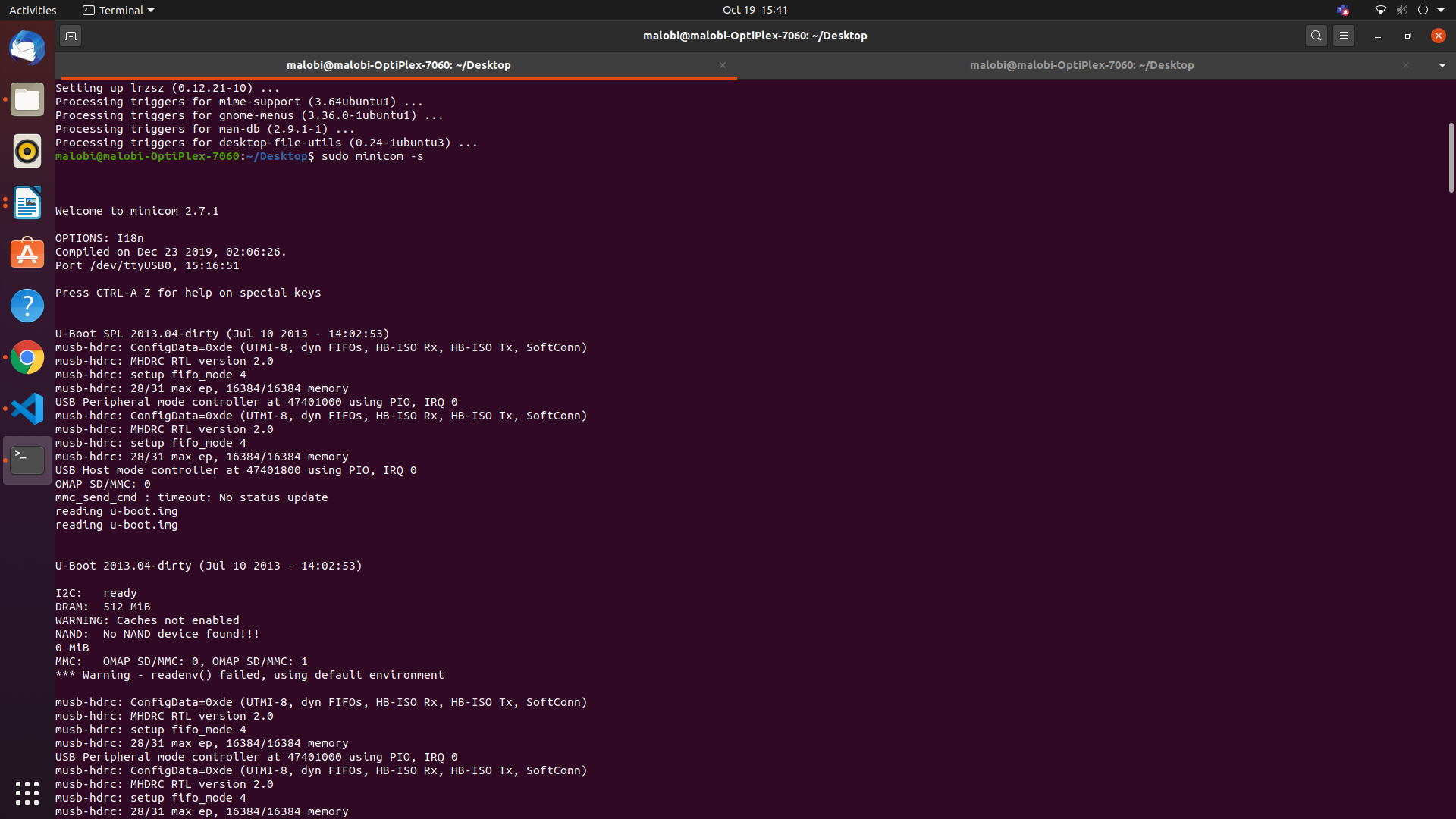


Figure 4: Loading u-boot

* 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"

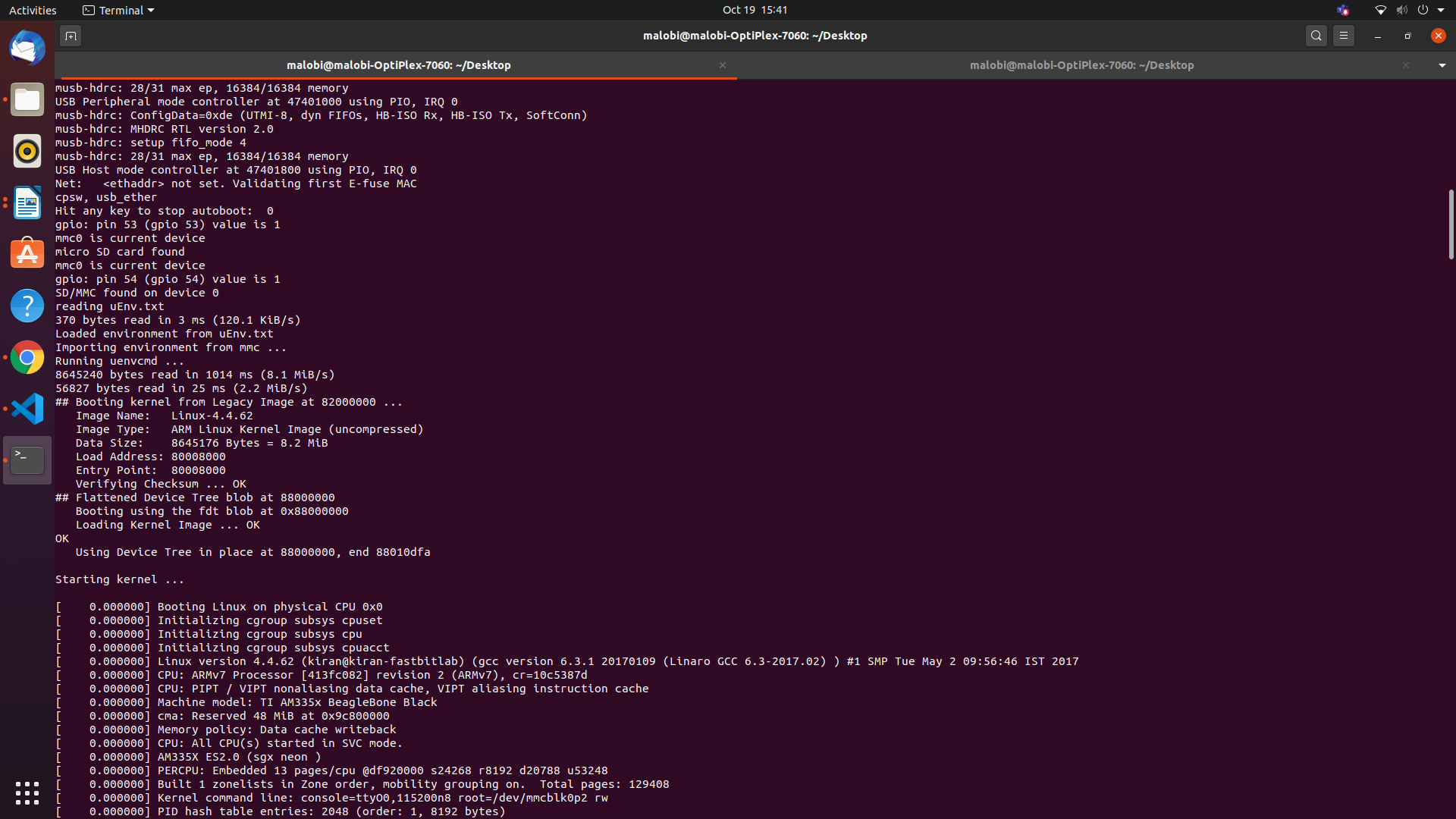


Figure 5: Loading from uEnv.txt

* 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:
* uBoot will then load the Linux Kernel and compiled Device Tree Binary blob from eMMC:
* And boot with the ext4 root filesystem being loaded from /dev/mmcblk0p2

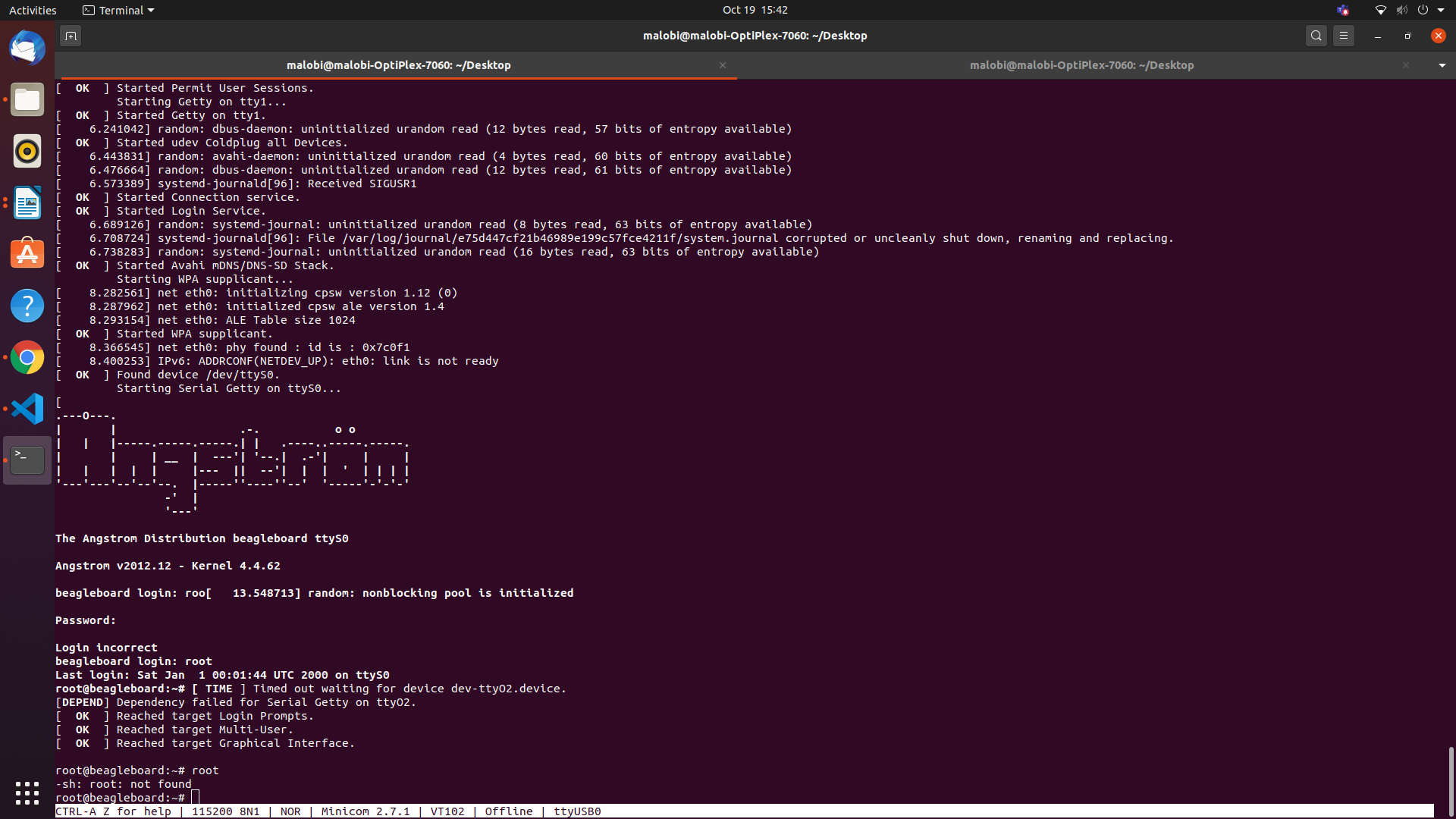


Figure 6: Booting to Kernel Level complete

In this step, the booting process is complete.

# Activity-7: Flashing new Debian image in eMMC from SD card

* Download the latest debian image from the beagleboard website- beagleboard.org
* Download Balena Etcher software.
* Insert the SD card.
* Open the downloaded debian image using the balena Etcher and select the required target to load. Now flash the image to sd card.



Figure 7: Flashing debian image into SD card

* Inside the rootfs folder, open the boot folder and uncomment the last line in the uEnv.txt file.
* Now insert the SD card to BBB and press the s2 button to start flashing onto eMMC.

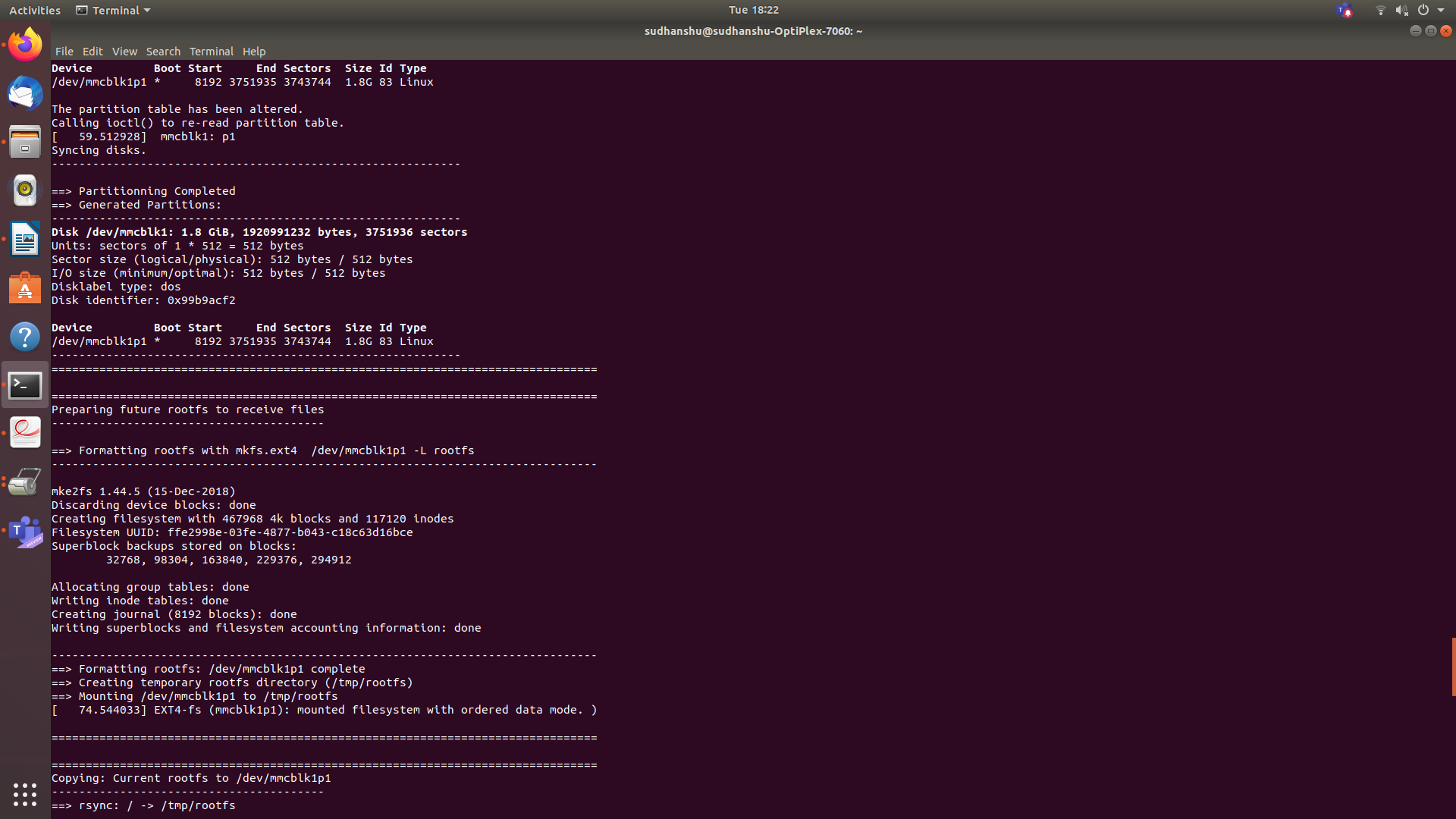


Figure 8: Booting from eMMC

# Activity-8: Establishing a connection from Beaglebone to local server

* Boot the BBB
* Enter the command “vi /etc/resolv.conf”
* Add server
  + nameserver 8.8.8.8
  + nameserver 8.8.4.4

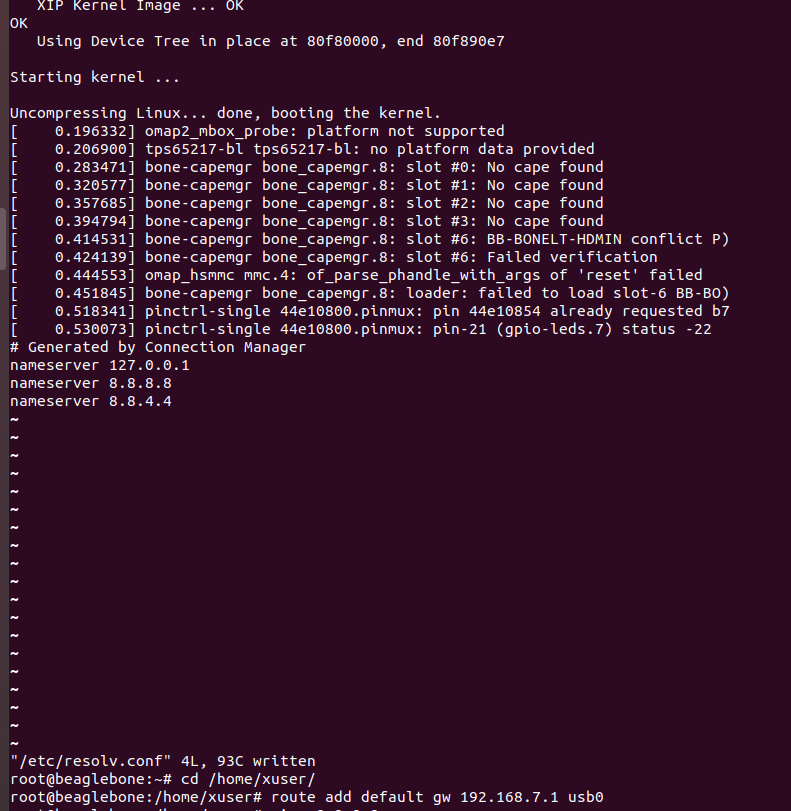


Figure 9: Printing the added server list

* Find the ip address of USB0 by giving “ifconfig” command.
* Now add the gateway “route add default gw 192.168.7.1 usb0”.

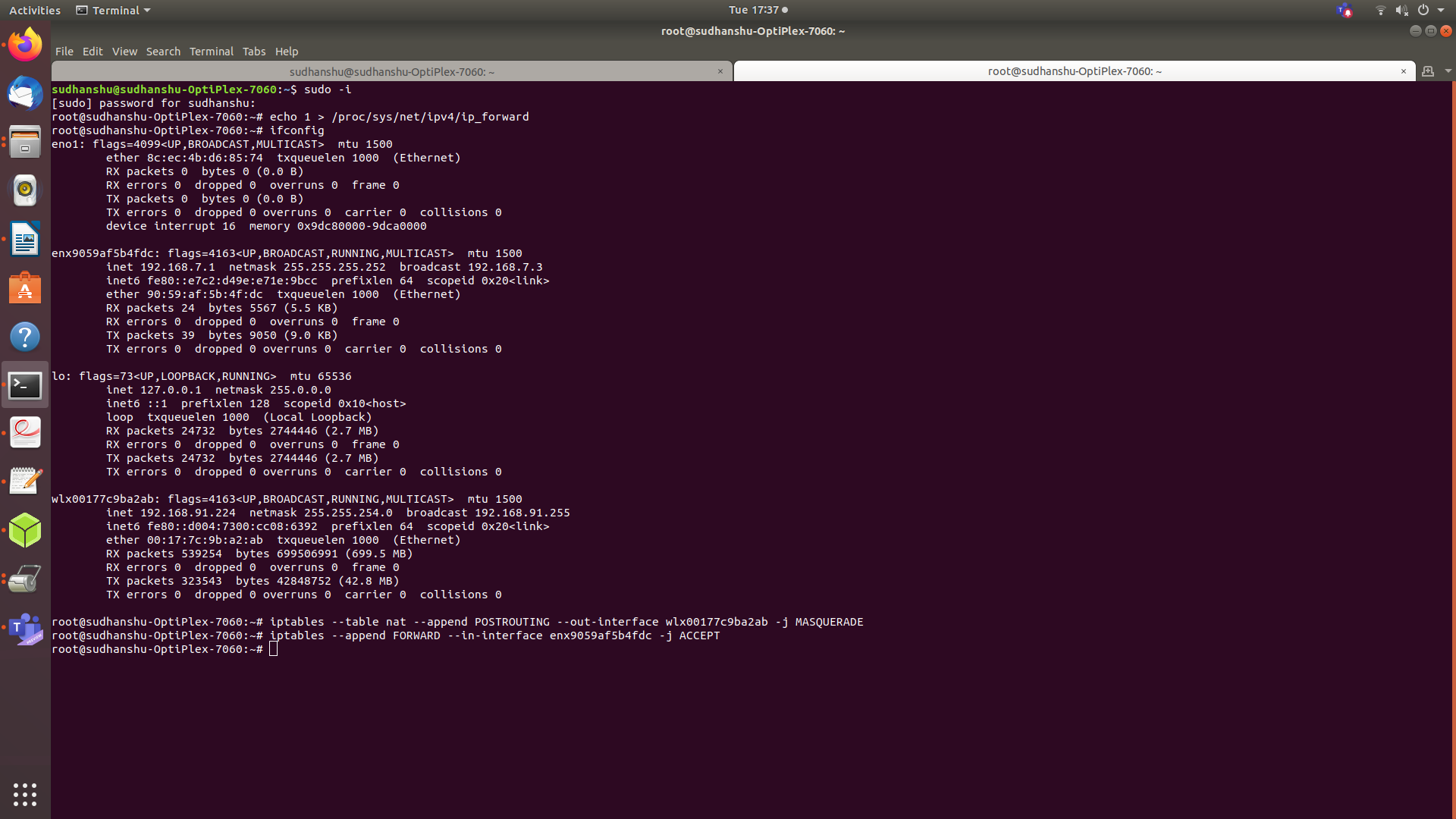


Figure 10: ifconfig details

* Now on the host mahine, enter into the root by giving the command “sudo –i”
* Enable the ip forwarding by giving the command
  + “echo 1 > /proc/sys/net/ipv4/ip forward”
* 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”
* Now in BBB root enter ping 8.8.8.8 The Wi-Fi enabling process is completed.

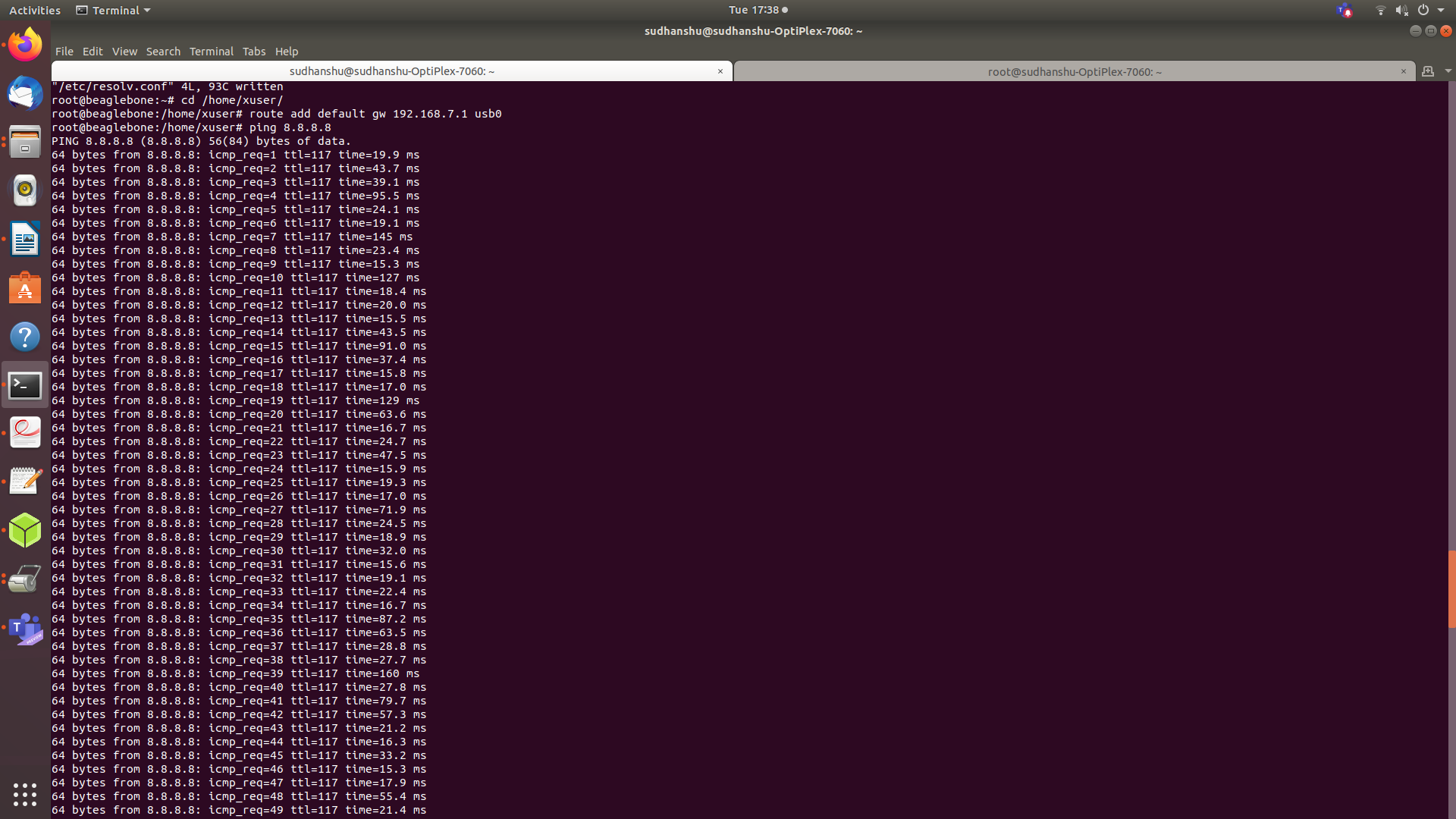
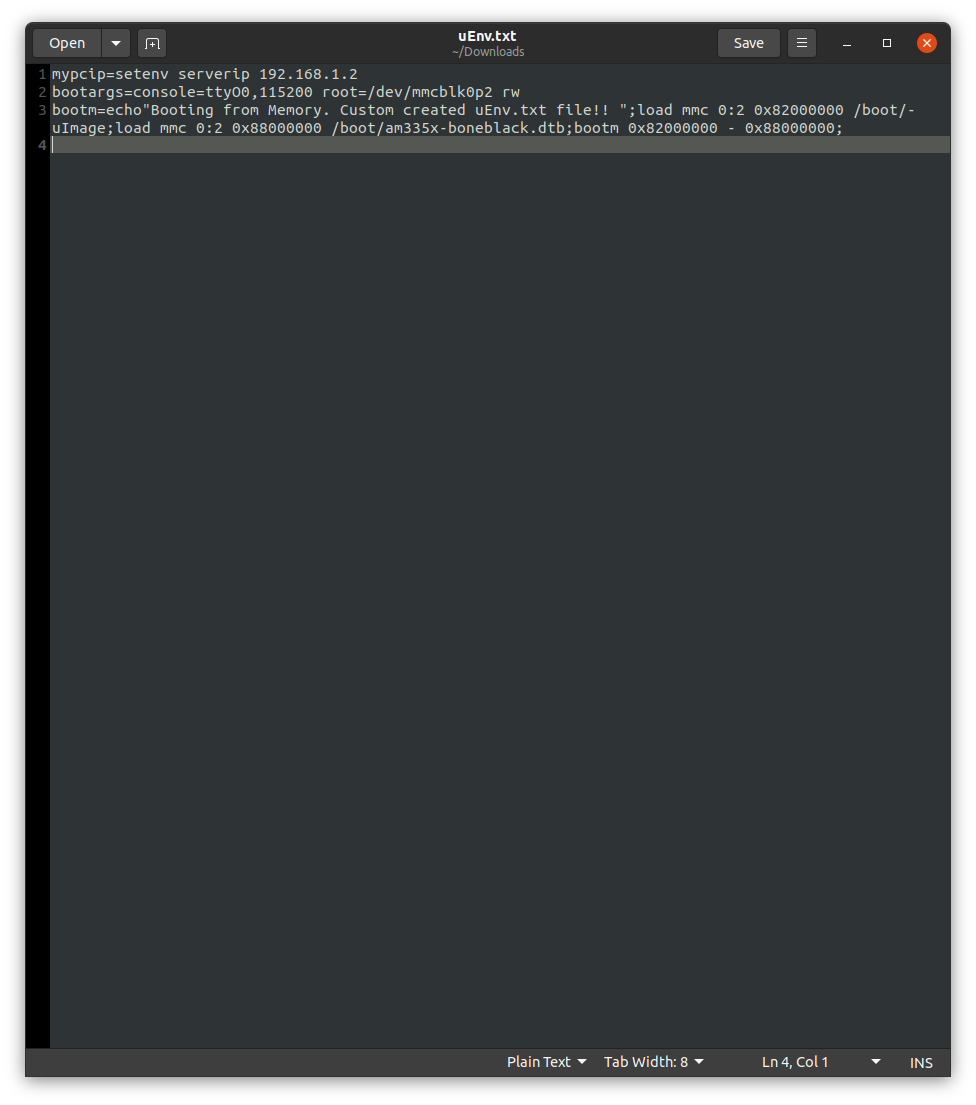
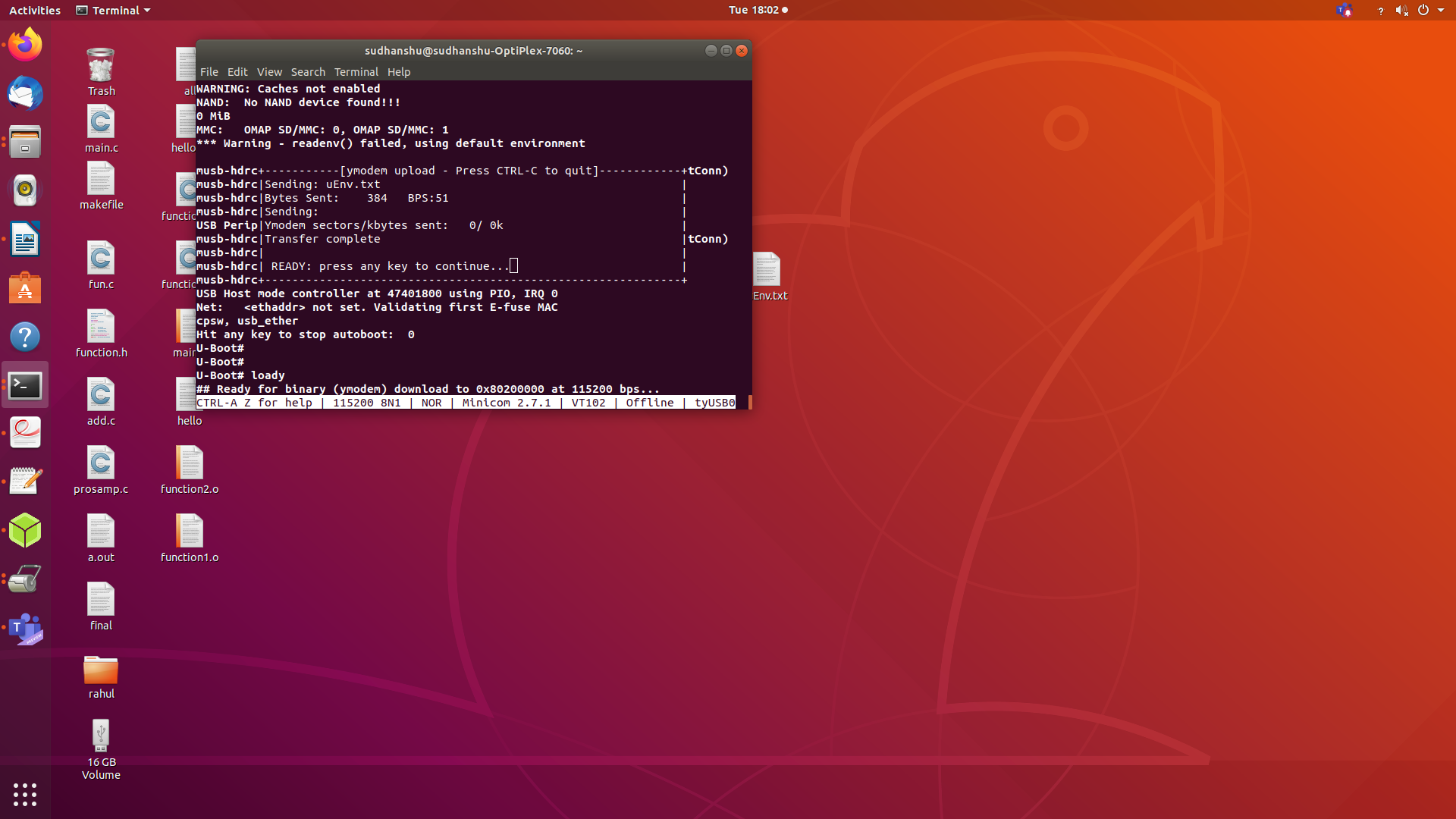


Figure 11: Server connection established

# Activity-9: Creating own uEnv.txt file

Figure 12: uEnv.txt file

* First set ip address of the server with setenv and store inside a variable.
* Then set the console and baud rate and also set it the root to read and writable store inside a variable.
* The variable bootm should contain all the booting and loading contents to be taken from the host.
* Bootm should contain mmc value i.e whether eMMC or SD card, path and address of uImage, path and address of .dtb file. It can also contain additional comments to be printed.

Figure 13: Including uEnv.txt into the board

# Activity-10: Changing the banner and login name

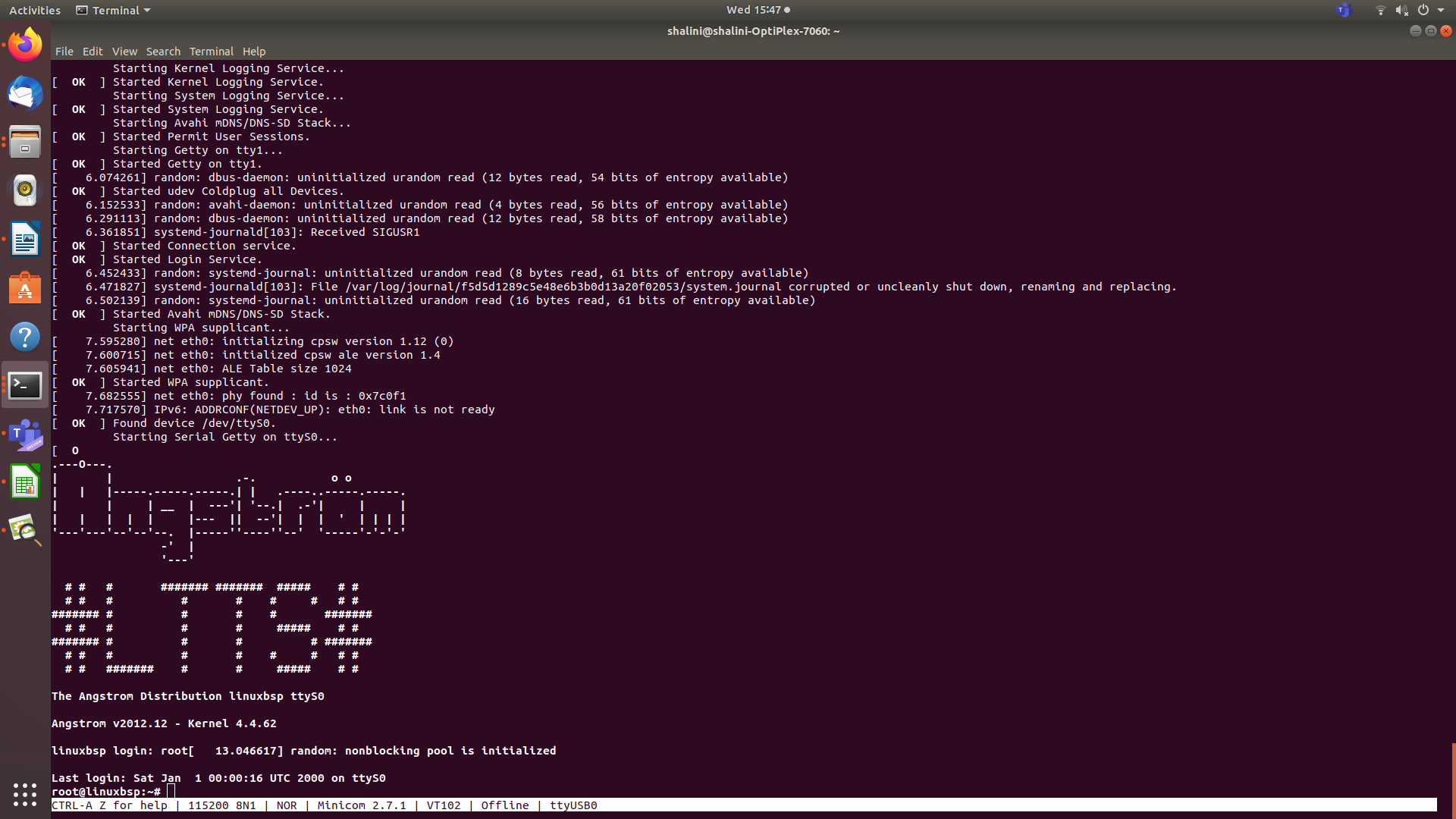


Figure 14: Changed banner and login name

To change the banner:

* Go to the etc directory inside ROOTFS.
* Locate a file named “issue”. Edit the file with admin access to create custom banner.

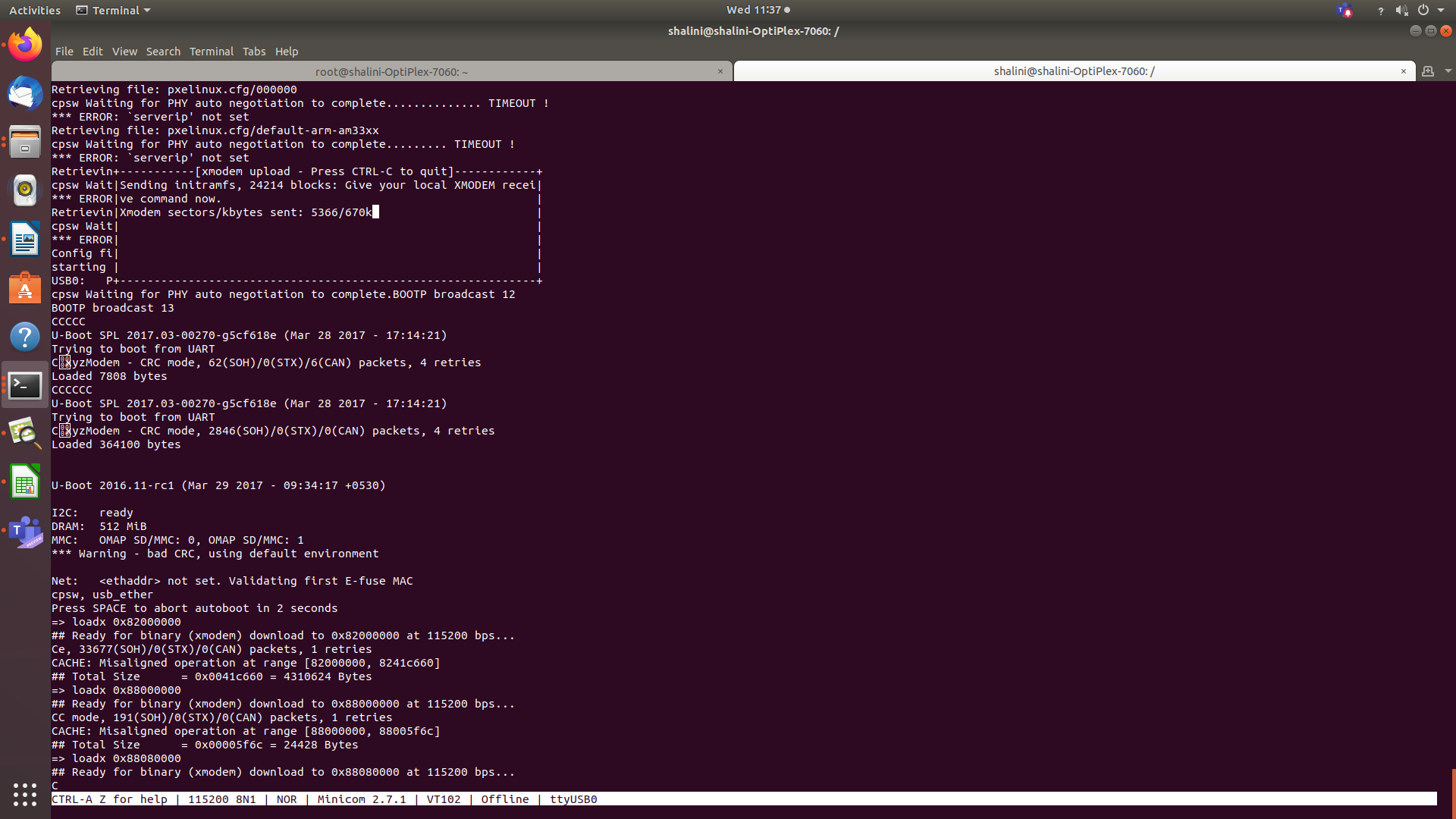
To change the login name:

* Go to etc directory inside ROOTFS.
* Locate a file named “hostname”and edit the login name with admin access.

# Activity-11: Serial Booting via UART

Booting can be done via host machine using UART protocol. UART interface is very reliable and easy to configure.

Note: With USB power, BBB will not fall-back to UART boot mode, so use power supply.

Figure 15: Loading images into the board via Xmodem

Steps for serial hosting:

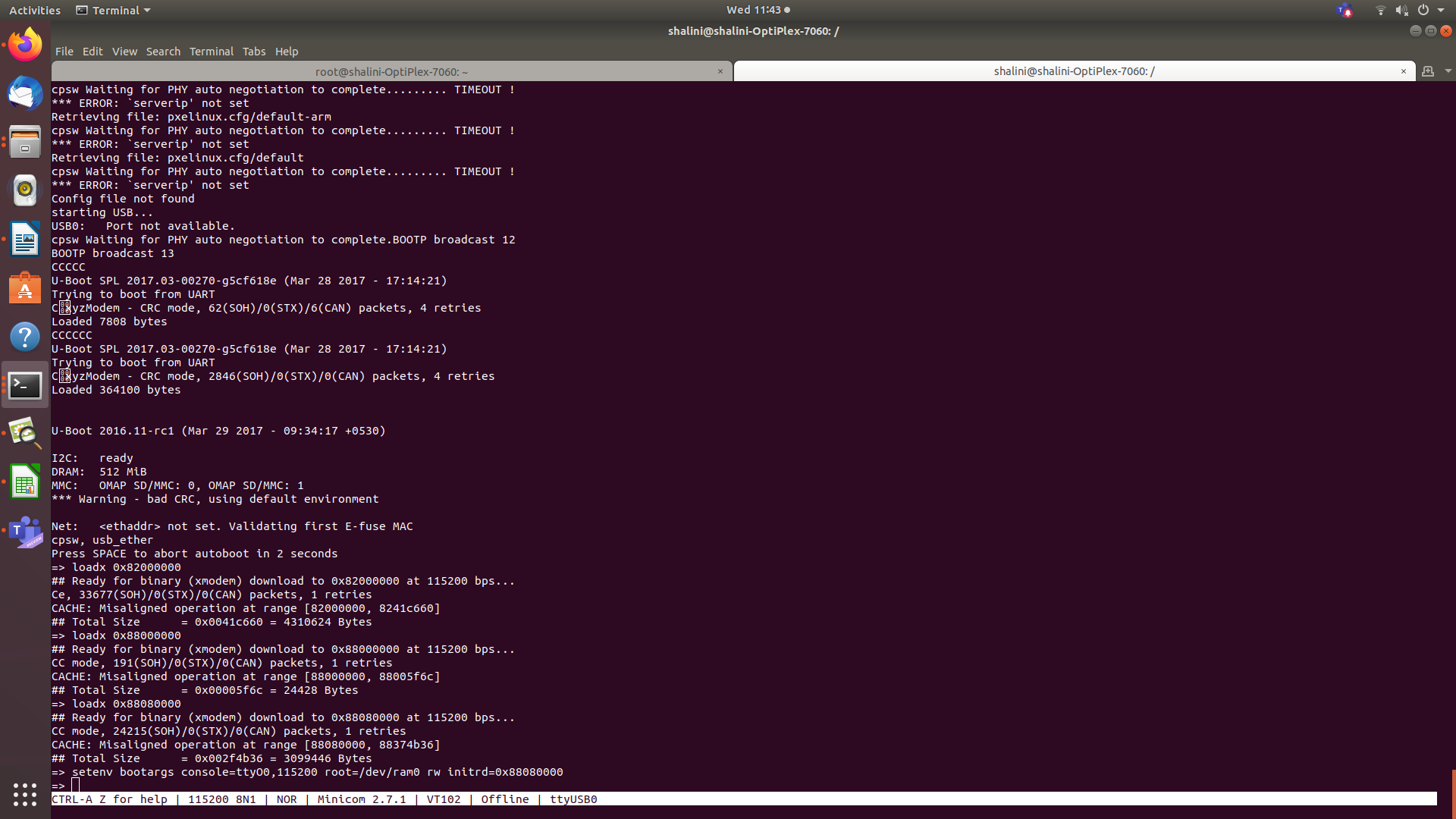
1. Start by pressing S2 as soon as the board is powered up using power adapter.
2. As soon as the screen shows CCCCC, press ctrl+A and then S.
3. Select loadx from the option and locate the u-boot.bin file and press enter. The file starts to get loaded into the board.
4. Then press Ctrl+A and S to select loadx again. This time select the u-boot.img file and press enter.
5. The screen will start loading and just before autoboot takes place, you press any key to enter into manual booting mode.
6. Type the following command to load uImage into the address 0x82000000.

loadx 0x82000000

1. Press Ctrl+A and S to select loadx. Select the uImage file and press enter.
2. Type the following command to load .dtb into the address 0x88000000.

loadx 0x88000000.

1. Press Ctrl+A and S to select loadx. Select the .dtb file and press enter.

Figure 16: Initiating all the commands

1. Type the following command to load initfs into the address 0x88080000.

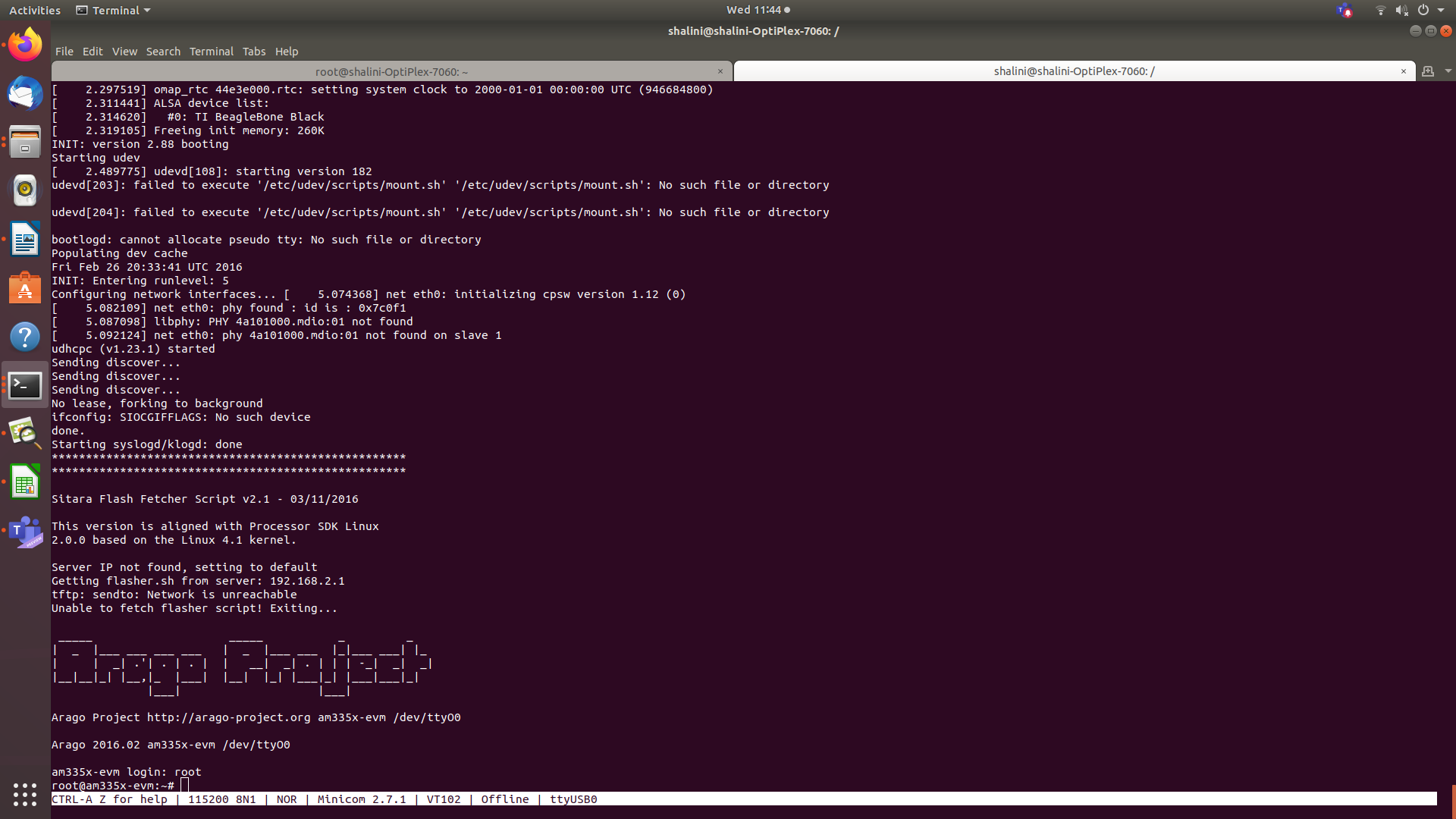
loadx 0x.88080000.

1. Press Ctrl+A and S to select loadx. Select the initfs file and press enter.
2. Type in the following commands:

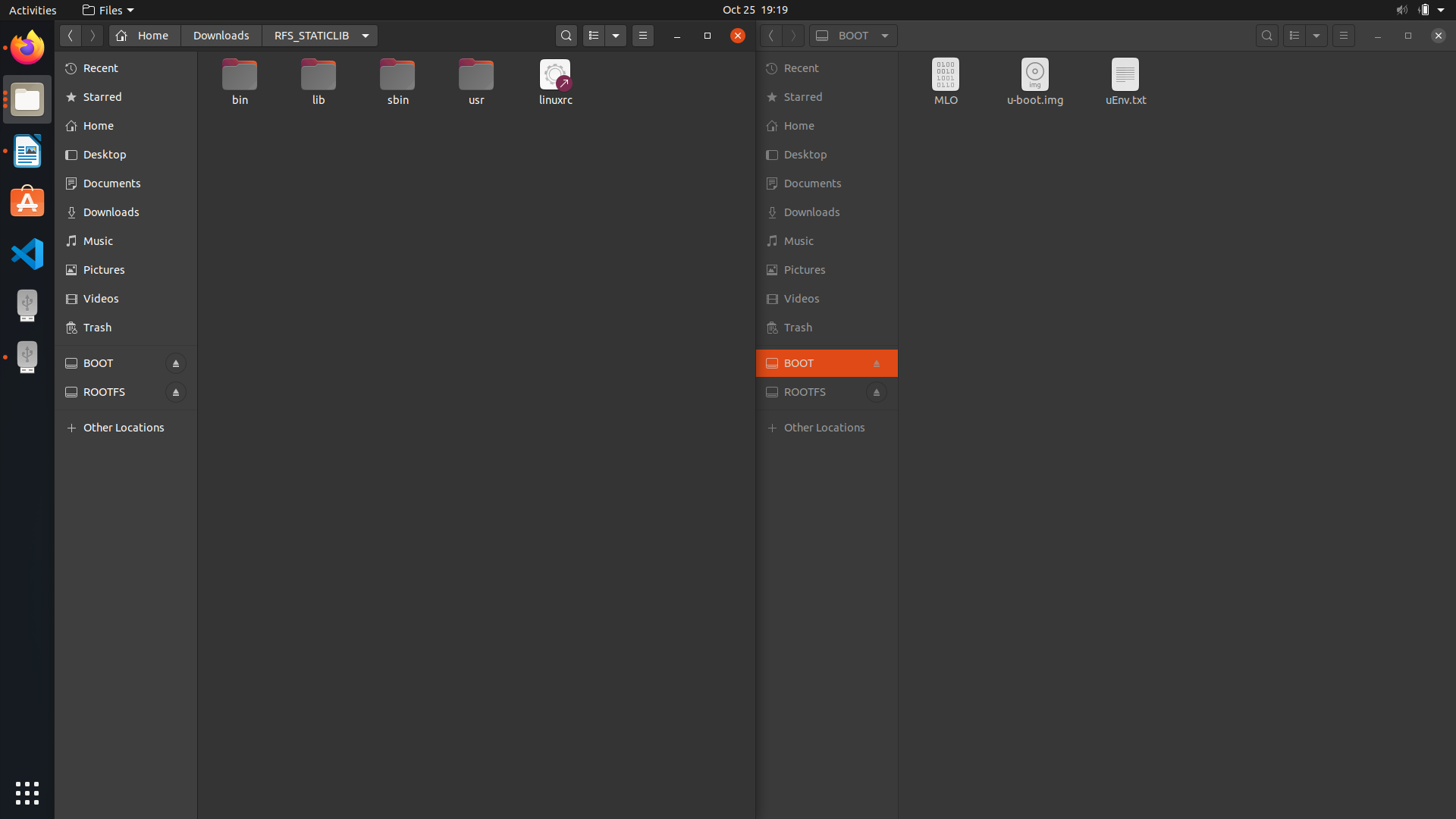
→ setenv bootargs console=ttyo0,115200 root=/dev/ram0 rw initrd=0x88080000.

→ bootm 0x82000000 0x88080000 0x88000000

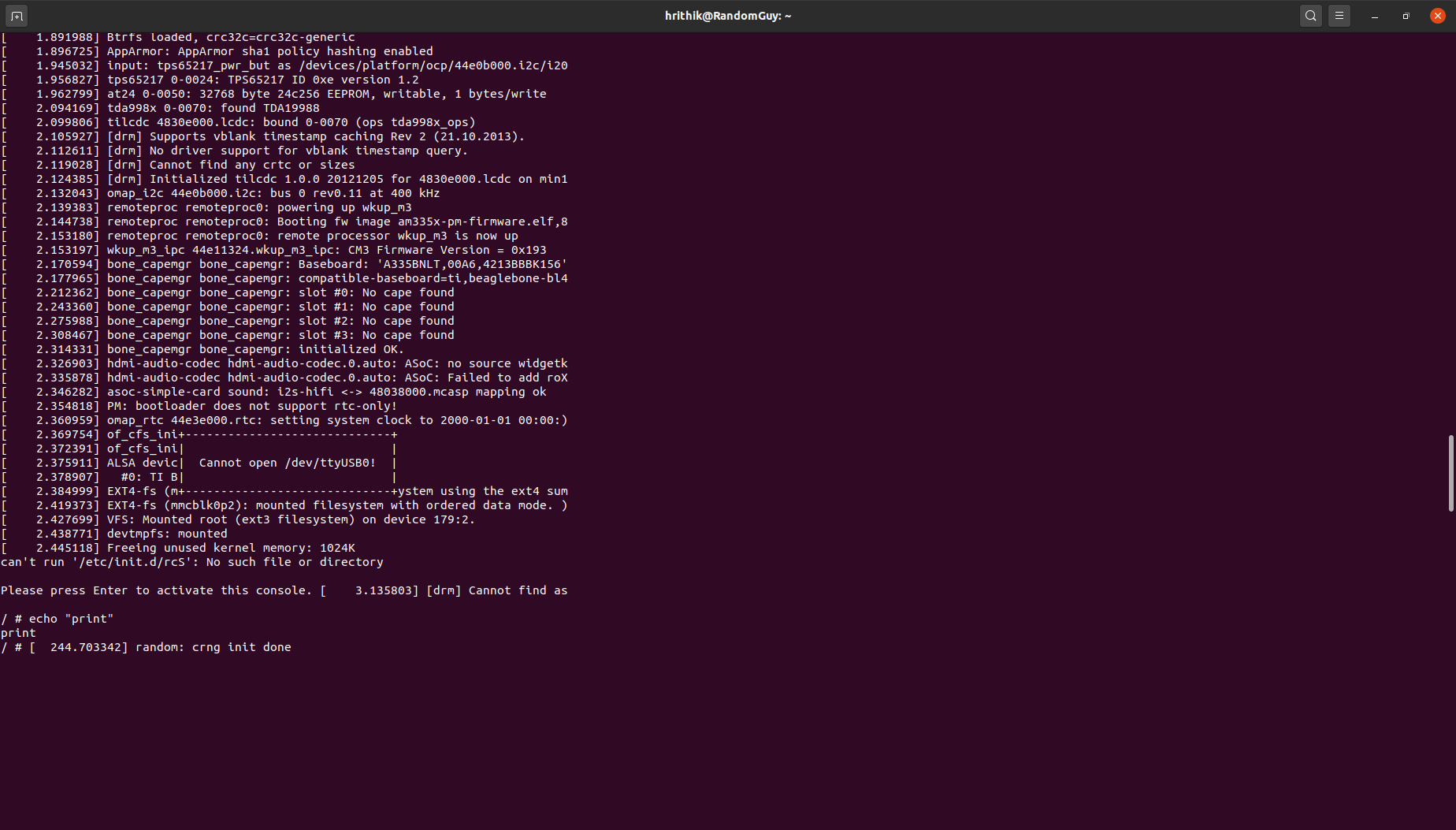
1. Booting is completed.

Figure 17: Serial booting process is completed

# Activity-12: Testing the images and created static library into BBB

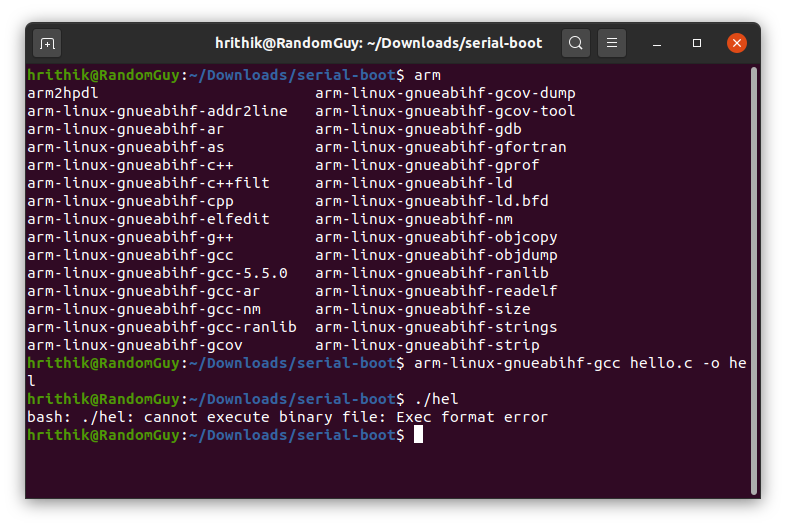
Figure 18: Static ROOTFS and BOOT

* Format the SD card and partition it as BOOT and ROOT FS.
* Copy the created MLO and U-boot image along with uEnv.txt into the BOOT.
* Copy the created static root file system into ROOTFS.
* Copy the uImage and .dtb file of BBB into boot folder inside ROOTFS.
* Find the tty images and copy them into dev directory in ROOTFS.
* Now unmount the SD card and boot up the beaglebone black with SD card.

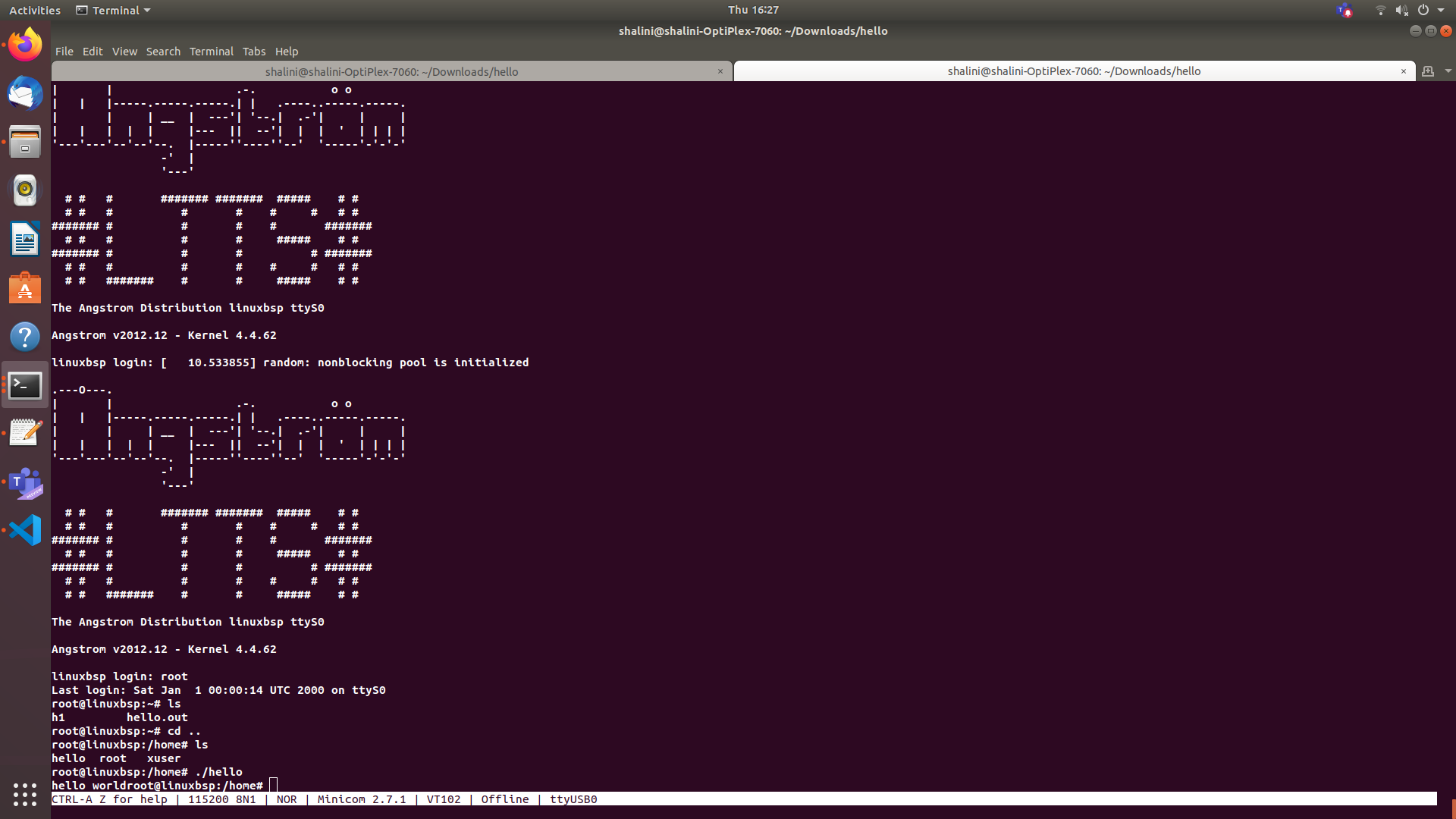
Figure 19: Inside Linux Kernel

# Activity-12: Compile a hello.c using arm compiler.

* First create a simple C program.
* Compile it and generate the object file using arm compiler.

Figure 20: Compilation using ARM compiler

* Copy the generated object file into the etc folder inside ROOTFS of SD card.

Figure 21: Output of the C file shown by BBB

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

Same as activity 9.

# Challenge 2:Increase the AUTOLOAD timings.

To configure autoload timings, we have to create own u-boot image.

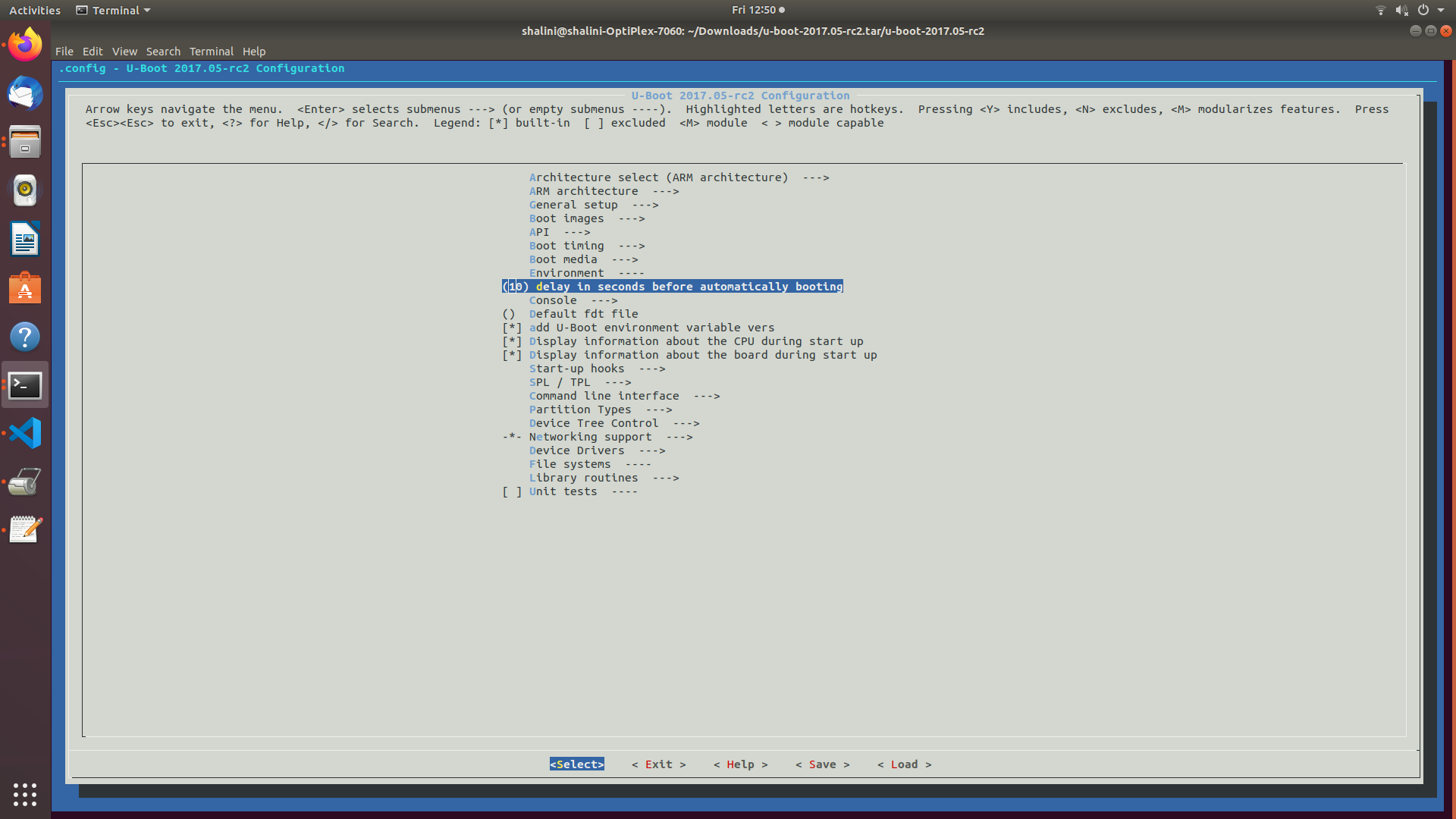
* STEP 1: distclean : deletes all the previously compiled/generated object files.

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- distclean

* STEP 2 : apply board default configuration for uboot

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- am335x\_boneblack\_defconfig

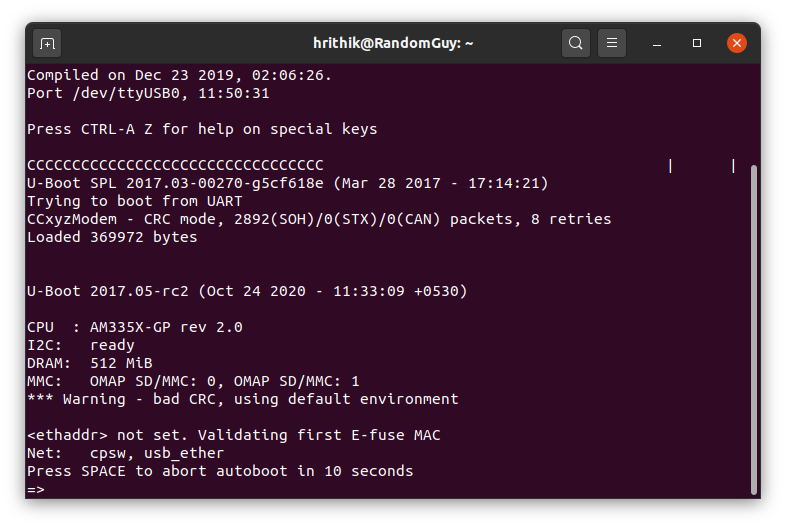
* STEP 3 : run menuconfig, This is where you can change the autoload timing.
  + Select the “delay in seconds before atomatically booting” and press spacebar.
  + Enter 10s as the delay and save it.



* STEP 4 : compile

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- -j4 // -j4(4 core machine) will instruct the make tool to spawn 4 threads.

* After the compilatiion, start booting up BBB using serial booting method.
* Upload the newly created U-boot.img instead of the old one.

Figure 22: Autoload timing changed to 10s