./

Learning Report

Embedded Linux

|  |  |  |
| --- | --- | --- |
| **Name** | **PS Number** | **Email ID** |
| Niranjan Kumar M | 99002516 | niranjankumar.m@ltts.com |

[Contents 3](#_Toc52675972)

[Learning Objectives of the Module 4](#_Toc52675973)

[**Activity 1** – Setup activity](#_Toc52675974) 4

[**Activity 2** – Differences between Raspberry pie , Dragon, imx7 Sabre, BBB](#_Toc52675975) 6

[**Activity 3** – Evolution and Changes of Beagle back Bone Board](#_Toc52675976) 8

[**Activity 4** – Pin expansion header of BBB and locate the various peripherals of Bone.](#_Toc52675977) 10

[**Activity 5** –](#_Toc52675978) [**Testing MLO image on BBB and Testing U-boot image on BBB**](#_heading=h.32hioqz)11

[**Activity 6** –](#_Toc52675979) [**Different booting stages in Beaglebone Black 1**](#_heading=h.1hmsyys)**2**

[1. ROM Bootloader (RBL) 1](#_heading=h.41mghml)2

[2. Secondary Program Loader/ MLO 1](#_heading=h.2grqrue)3

[3. U-Boot](#_heading=h.vx1227)14

[**Activity 7-Learn and quote the Linux boot sequence after the booting.**](#_heading=h.1v1yuxt) **15**

[**Activity 8-Challenge-Make uEnv.txt to Boot from MMC0 and MMC 1 .**](#_heading=h.4f1mdlm) **18**

[**Activity 9-Challenge-Write a uEnvt.txt file to automate TFTP boot.**](#_heading=h.2u6wntf) **19**

[**Activity 10-Challenge-Write a generic uEnv.txt 2**](#_heading=h.19c6y18)**0**

[**Activity 11-Challenge-Increase the AUTOLOAD timings . 2**](#_heading=h.3tbugp1)**1**

[**Activity 12-Challenge-Busybox "Dynamic" Compilation 2**](#_heading=h.28h4qwu)**3**

[**Reference 2**](#_heading=h.nmf14n)**8**

# ACTIVITY 1:SETUP ACTIVITY

**Step by Step Configuration of the boards and set up in Windows OS:**

Step 1: Plug in the Ethernet chord to Beagle Bone Black Ethernet port (to establish communication)

(or) use any other bootalbe options such as SD card, MMC.

Plug in the USB chord to the host machine and the other end to the micro USB port in the

Beagle Bone Black (To supply power to the Beagle Bone Black)

Step 2: Installing Drivers in the Host machine

Visit https://beagleboard.org/getting-started

Depending on the configuration of the host machine download and install the respective USB

Driver Installer. Once installed Reboot the host machine.

Step 3: Connecting to Beagle Bone Black via Ethernet.

Open any Browser (preferably Chrome or Firefox) type the below IP address in the URL

IP address: 192.168.7.2 .

Beagleboadr.org Web page gets loaded which is already present in the Beagle Bone Black

Now the connection to Beagle Bone Black is successful.

Step 4: Obtaining the unique IP address

Click on the Cloud9 IDE a web page loads and select once again Cloud9 IDE this takes to the

Cloud9 IDE which is running on the Beagle Bone Black.

(If any error occurs then make sure don’t use the Internet Explorer Browser)

Open new terminal in the Cloud9 IDE and type the command “ifconfig”.

Note down the IP address in the eth0 section.

Step 5: Connect to the Beagle Bone Black using simple SSH client i.e., PuTTY

Open the URL : https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Download the putty.exe depending on the host machine configuration.

Step 6: Launch the SSH client i.e., PuTTY. Enter the IP address (Noted in the Step 4) in the Host name *IP*

*address text field. Click on Open. This launches a Linux Terminal window in the host machine*

*which asks for the login credential login as: root (which is default).*

Step 7: Remotely connecting to the Beagle Bone Black using TightVNC viewer.

TightVNC server is already installed in the Beagle Bone Black. But TightVNC viewer is

required for the remote PC in order to access remotely. Download the TightVNC viewer using url

link : <https://www.tightvnc.com/download.php>

Download and install the TightVNC viewer based on the configuration on windows.

Step 8: Fire up VNC sever before running the TightVNC viewer in the remote machine.

Launch putty.exe in the Host machine and login. Then install tightvncserver.

Type the “sudo apt-get install tightvncserver” command.

(Note: The below steps are to be performed only once)

Then type command “typevncserver” (Press enter)

Set Password and Verify Password.

Now type command “vncserver :1 -geometry 1280x800 -depth 24 -dpi 96” (Press enter)

Step 9: Launch TightVNC viewer in the remote machine and enter the IP address (Noted in Step 4) along

with :1 Example: 10.1.15.25 : 1

Click on connect and enter VNC password which was set in the Step 8.

A graphical user interface window pops up.

**Step by Step Configuration of the boards and set up in Linux OS:**

Step 1: sudo minicom -s

Step 2: serial port setup (know the TTL cable name)

(In other terminal)

Step 3: dmesg (Search for Prolific Technology) port ttyUSB0

Step 4: Press a and enter /dev/ttyUSB0

Step 5: Press e check for Standard Bod rate : 115200 8N1

8-bit

N-no parity

1-Stop bit

Step 6: Press f, Check for Hardware flow control set it to NO

Step 7: Press g, Check for software flow control set it to NO

Step 8: Save the settings as dfl (default)

Step 9: Exit

Step 10: connect usb

ANGSTROM

Step 11: beaglebone login: root

root@beaglebone:~#

Step 12: Shutdownnow

# Activity 2: Differences between Raspberry pie , Dragon, imx7 Sabre, BBB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **RASPBERRY PIE** | **BEAGLEBONE BLACK** | **Dragon** | **imx7 Sabre** |
| Processor Type | It uses ARM11 processor. | It uses ARM Cortex-A8 processor. | Quad-core ARM® Cortex® A53 at up to 1.2 GHz per core with both 32-bit and 64-bit supporT | Two Arm Cortex-A7 core OS upto 1 GHz, Single Arm Cortex- M4 CORE os |
| 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 533MHz / 8GB eMMC 4.5 / SD 3.0 (UHS-I) | 1 GB DDR3, 533 MHz  eMMC expansion footprint. |
| Processor Speed | It uses 700 MHz for processing. | It uses 1 GHz for its processing. | 1.2 GHz per core with both 32-bit and 64-bit support | 1 GHz :Arm Cortex-A7  200 MHz :Arm Cortex -M4 |
| Min Power | It requires a power supply of 700mA (3.5W). | It requires min power of 210mA (1.05W) for its functioning. | It requires a power supply of 8-18V 2A. | It requires 5V/5A universal power supply. |
| GPIO Pins | It has 12 GPIO pins. | It has 69 GPIO pins. | It has 40 GPIO pins | It has 138 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. | Android 5.1 (Lollipop) on Linux Kernel 3.10 | Vivante Tool v6.2.4.p4.1.7.9 and linux based systems |
| USB Master | It has 2 USB 2.0 on board. | It has 1 USB 2.0 on its board. | one micro USB (device mode only), two USB 2.0 (host mode only) | * 1 USB host connector * 1 micro USB OTG connector |
| Audio Output | Supports HDMI, Analog audio output | It uses Analog output for audio. | It calls for a minimum of single channel audio through two interfaces, BT and HDMI/MHL/DisplayPort | i.MX7 has multiple audio interfaces and one is fully available on the SODIMM connector of the Colibri iMX7. |
| Video Output | It supports HDMI, Composite output for video. | No such specific video output. | 1080p@30fps HD video playback and capture with H.264 (AVC), and 720p playback with H.265 (HEVC) | Its supports HDMI, composite output for video. |
| UART | It uses 1 UART to transmit and receive serial data. | It uses 5 UART to transmit and receive serial data. | support for one SoC UART and an optional second UART both to be routed to the Low Speed Expansion Connector. | UART via USB port |
| No. of I/O pins | It has 8 Digital, 0 Analog pins. | It has 65 Digital, 7 Analog pins. | It has 11 Digital ,0 Analog pins. | It has total 138 pins |

# Activity 3:Evolution and Changes of Beagle back Bone Board

|  |  |
| --- | --- |
| **Revision** | **Additions(differences)** |
| A4 | Preliminary |
| A4A | Incorporated the capacitors to fix the noise issue on the display |
| A4B | Added a 100K pull down resistor between pins 1 and 4 of J1 to fix the serial port issue |
| A5.1 | 1.Added information on Power button and the battery access  points  2.Final production released version. |
| A5.2 | 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. |
| A5.3 | 1. Updated serial number locations.  2. Corrected the feature table for 4 UARTS  3. Corrected eMMC pin table to match other tables in the manual. |
| A5.4 | 1. Corrected revision listed in section 2. Rev A5A is the initial  production release.  2. Added all the locations of the serial numbers  3. Made additions to the compatibility list.  4. Corrected Table 7 for LED GPIO pins.  5. Fixed several typos.  6. Added some additional information about LDOs and Step-Down converters.  7. Added short section on HDMI. |
| A5.5 | 1. Release of the A5B version.  2. The LEDS were dimmed by changing the resistors.  3. The serial termination mode was incorporated into the PCB. |
| A5.6 | 1. Added information on Rev A5C  2. Added PRU/ICSS options to tables for P8 and P9.  3. Added section on USB Host  4. Correct modes on Table 15.  5. Fixed a few typos |
| A5.7 | 1. Updated assembly revision to A6.  2. PCB change to add buffer to the reset line and ground the oscillator GND pin.  3. Added resistor on PCB for connection of OSC\_GND to board GND. |
| A6 | 1. Added changes for rev A6 that covered fixing of the link LED, JTAG Reset, and DHCP issue.  2. Added PRU information and two additional signals for the PRU.  3. Added write protection to EEPROM.  4. Fixed numbering of subsections in Section 7.0  5.Fixed error in Table 9 pin 23Mode 1 should be MMC1\_DAT4.  6. Updated Table 7 to show the revision number in the EEPROM matches the revision of the board.  7. Corrected various typos.  8. Updated Battery Interface section to accurately document the LDO dropout at 200mV.  9. Added SW Support section. |
| A6A | 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. |
| B | 1.Changed the processor to the AM3358BZCZ  2.No changes in features or operation of the board resulted from this change. |
| C | 1.This revision increased the eMMC from 2GB to 4GB |

# Activity 4: Pin expansion header of BBB and locate the various peripherals of Bone.

<https://github.com/L99002516/embedded_linux.git>

the above link has been updated with the Pin expansion header of BeagleBone Black. Which has been P8 and ref excel datasheet.

Types of Pins:

1. 23 – Reconfigurable Digital pins
2. 7 -- Analog Inputs Pins
3. 2 -- Shared I2C Pins
4. 7 -- Pulse width modulation
5. 25 – Digital Pins
6. 32 – Power management Pins

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

1. Step 1: Download gparted using the command
   1. sudo apt install gparted
2. Step 2: Insert card reader and select /dev/sdb(gb) and open gparted and make two partitions.
3. 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.
4. 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.
5. Step 5: Right click on BOOT and select manage flags then select boot option and then close.
6. 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.
7. Step 7: Create a workspace copy MLO file in BOOT and Workspace. Then unmount the SD card.
8. Step 8: Then connect SD Card in board and open the terminal.
9. Step 9: Install minicom by using the command
   1. sudo apt install minicom
   2. sudo minicom -s for setup changes
10. Then select on serial device as /dev/ttyUSB0

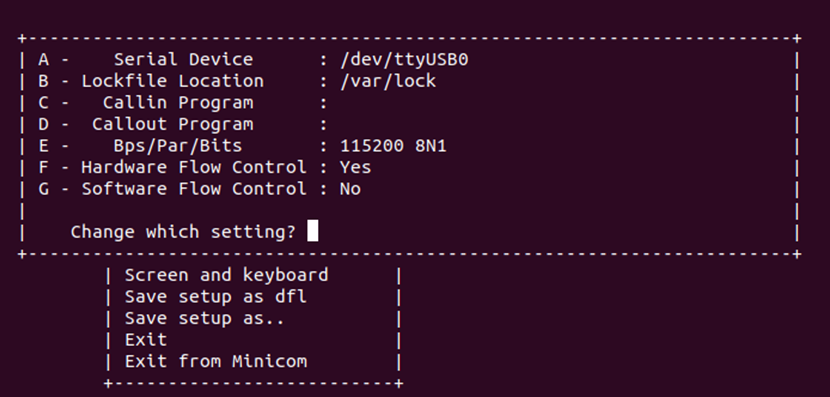


Figure 1: Serial device select

* 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.

# Activity 6: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.

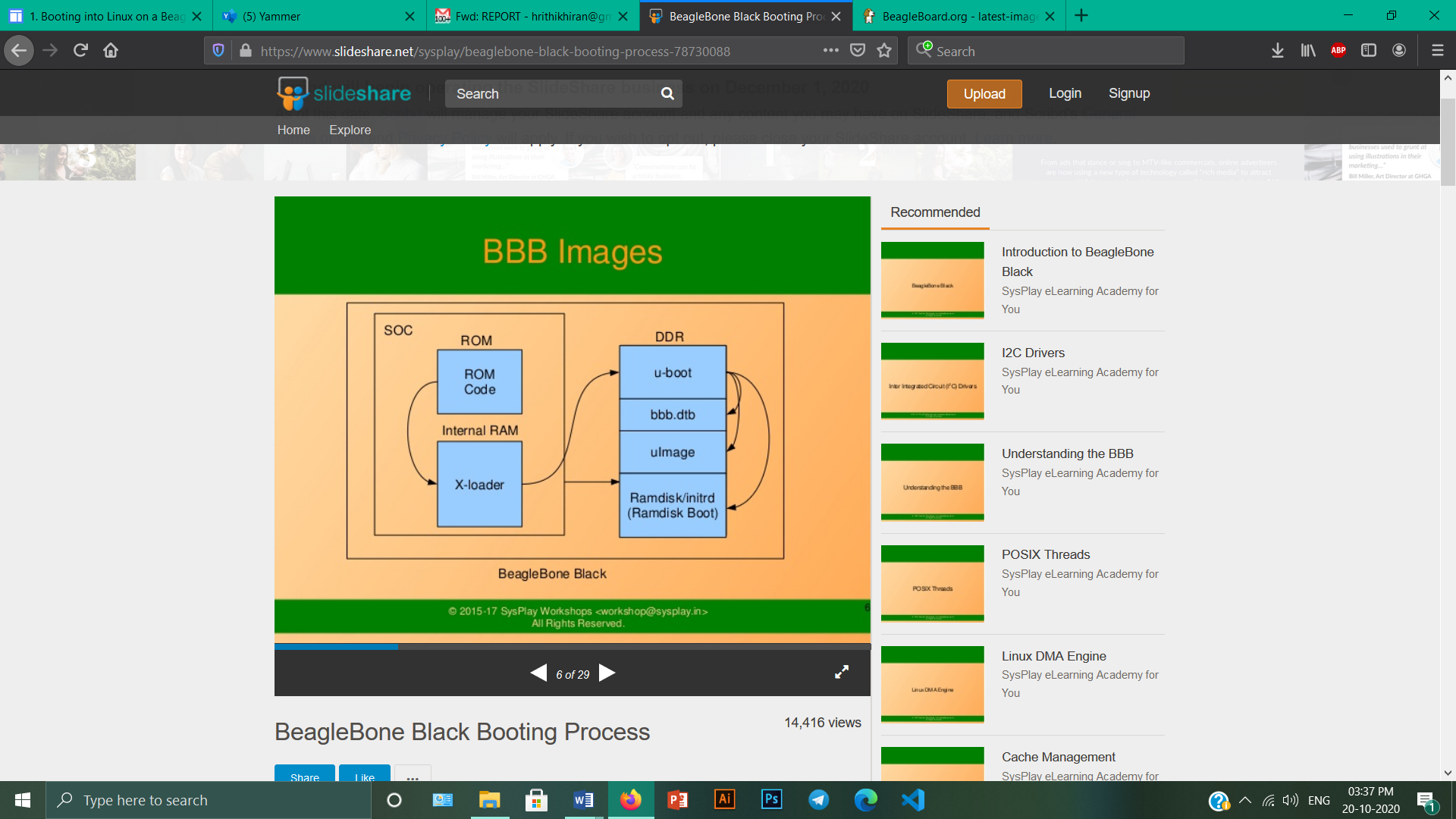


Figure 2: Booting Process

### 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.

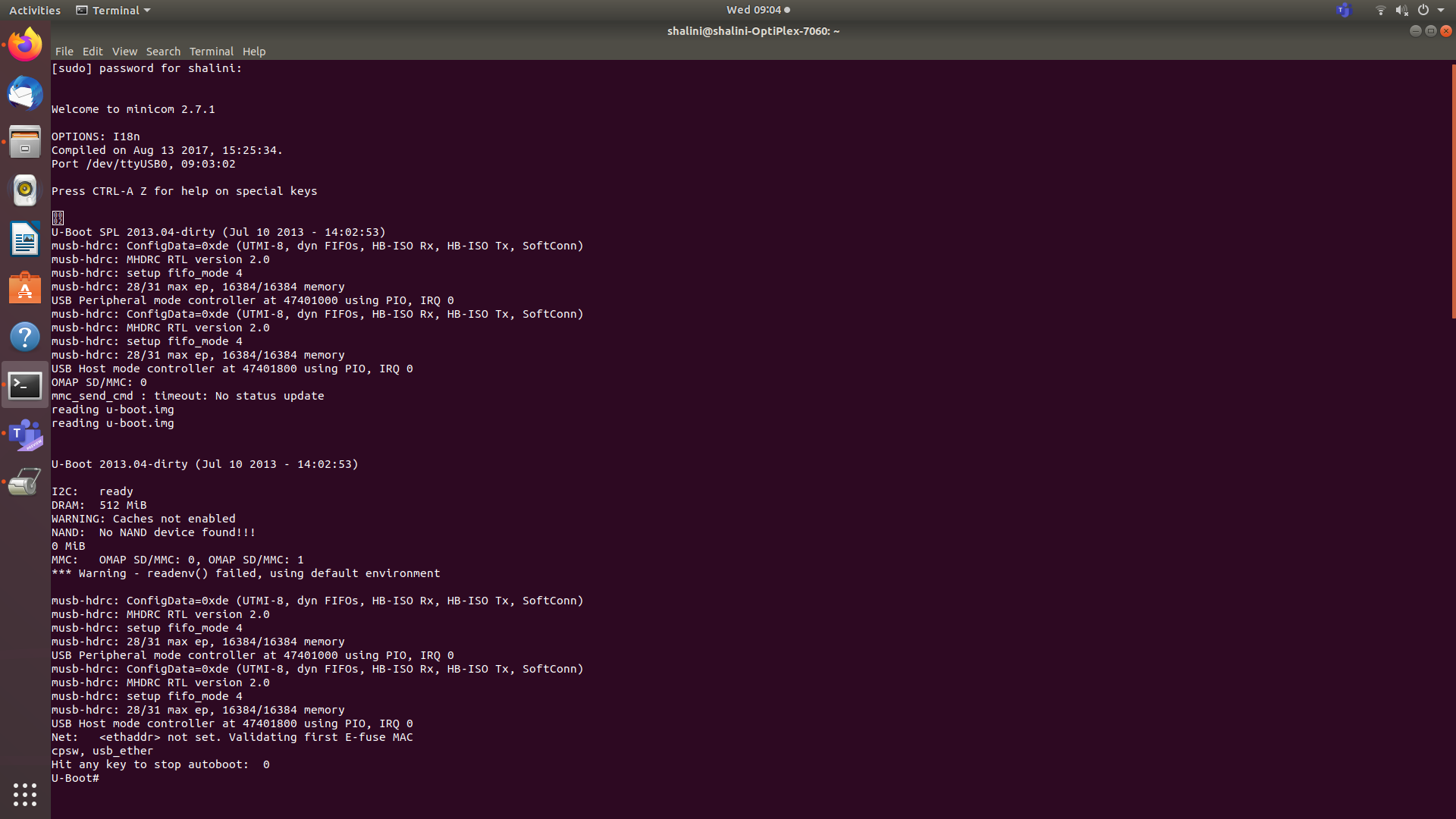


Figure 3: Booting till U-Boot

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-7: 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)

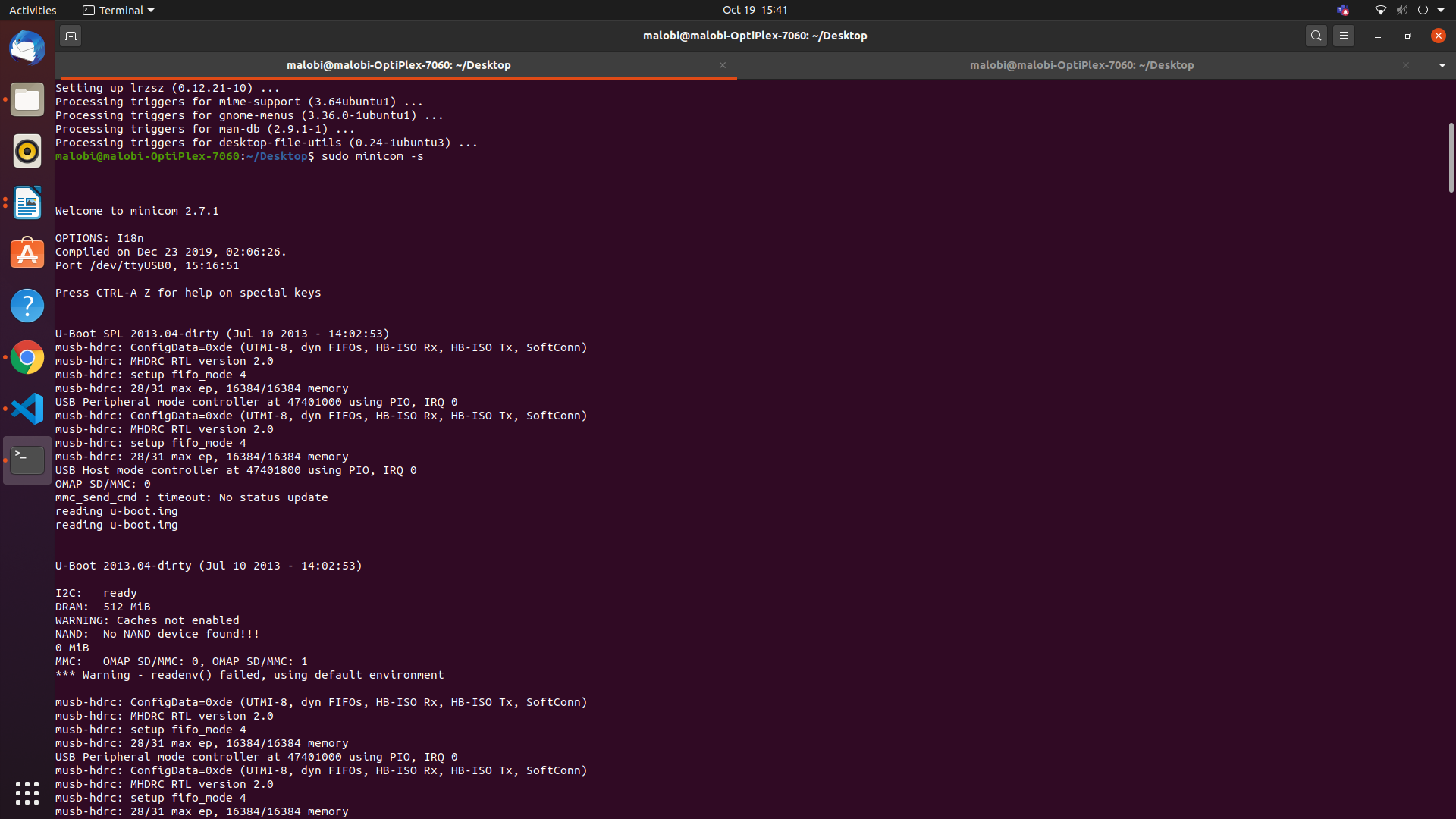


Figure 4: Loading u-boot

* 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"

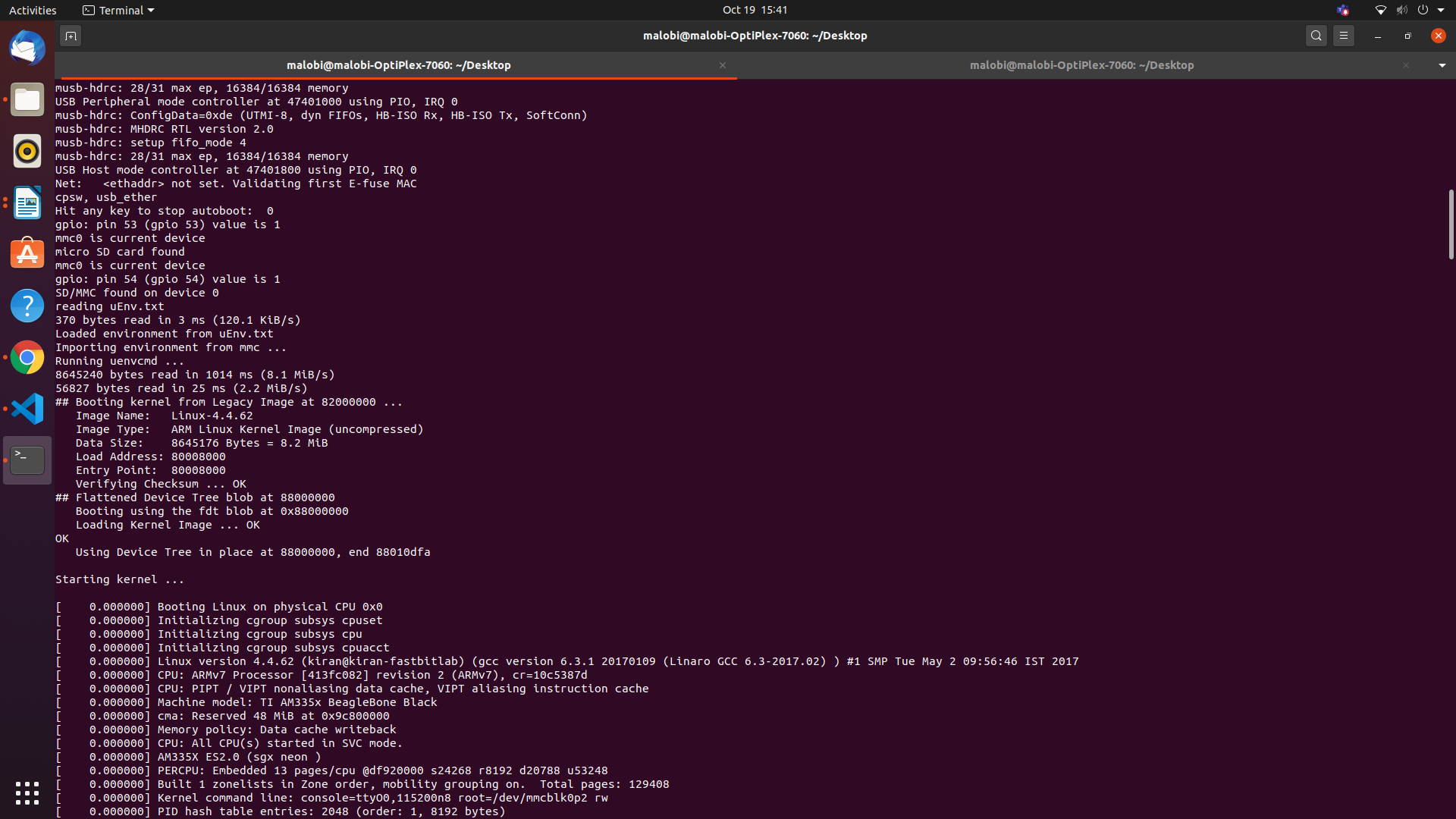


Figure 5: Loading from uEnv.txt

* 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

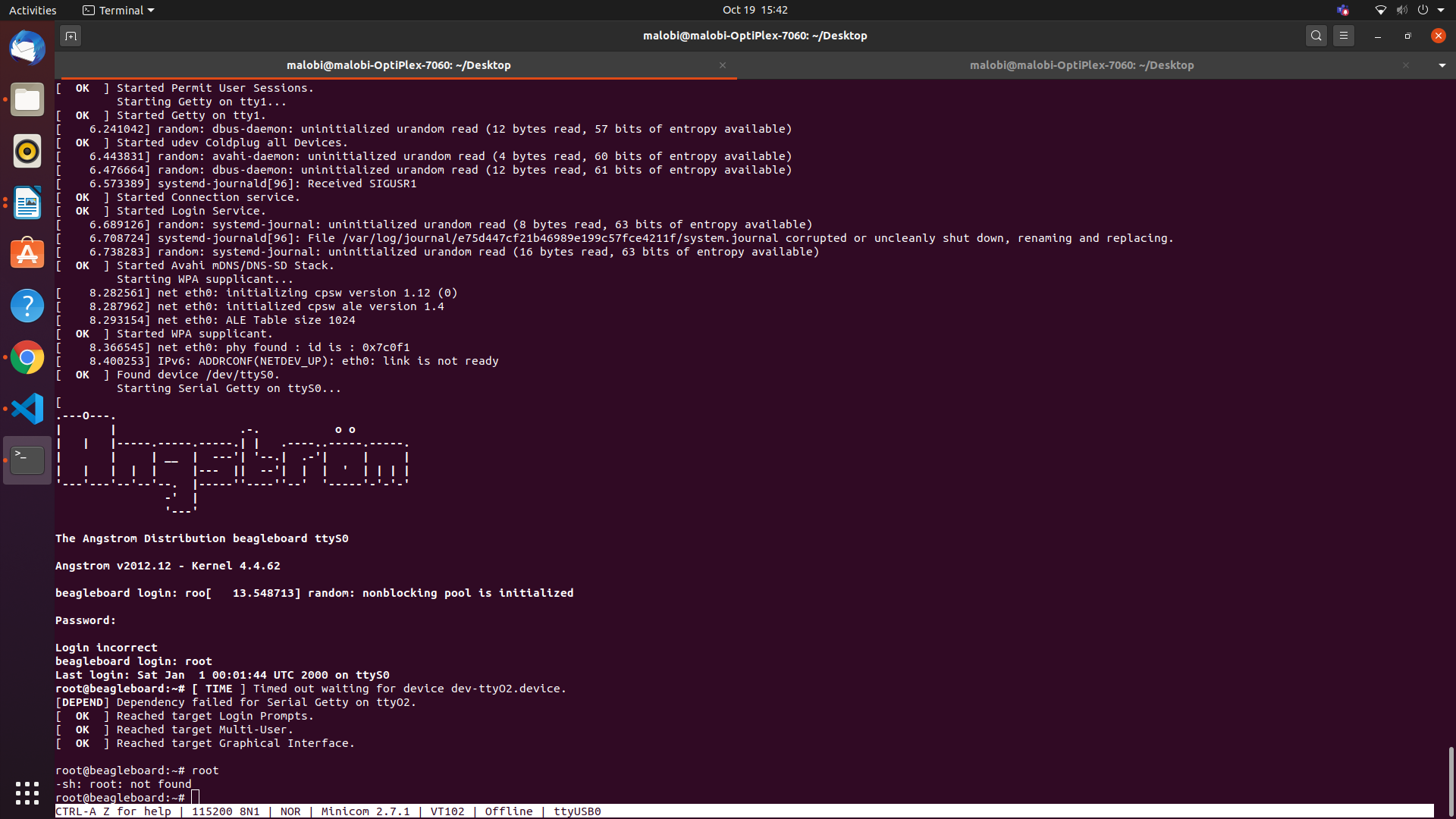


Figure 6: Booting to Kernel Level complete

In this step, the booting process is complete.

# **Activity 8-Challenge-Make uEnv.txt to Boot from MMC0 and MMC 1 .**

* Step 1:

Set IP address of the server with setenv and store inside a variable.

myserverip=setenv serverip=192.168.1.2

* Step 2:

Then set the console and baud rate and also set it the root to read and writable store inside a variable.

Set baud rate as 115200.

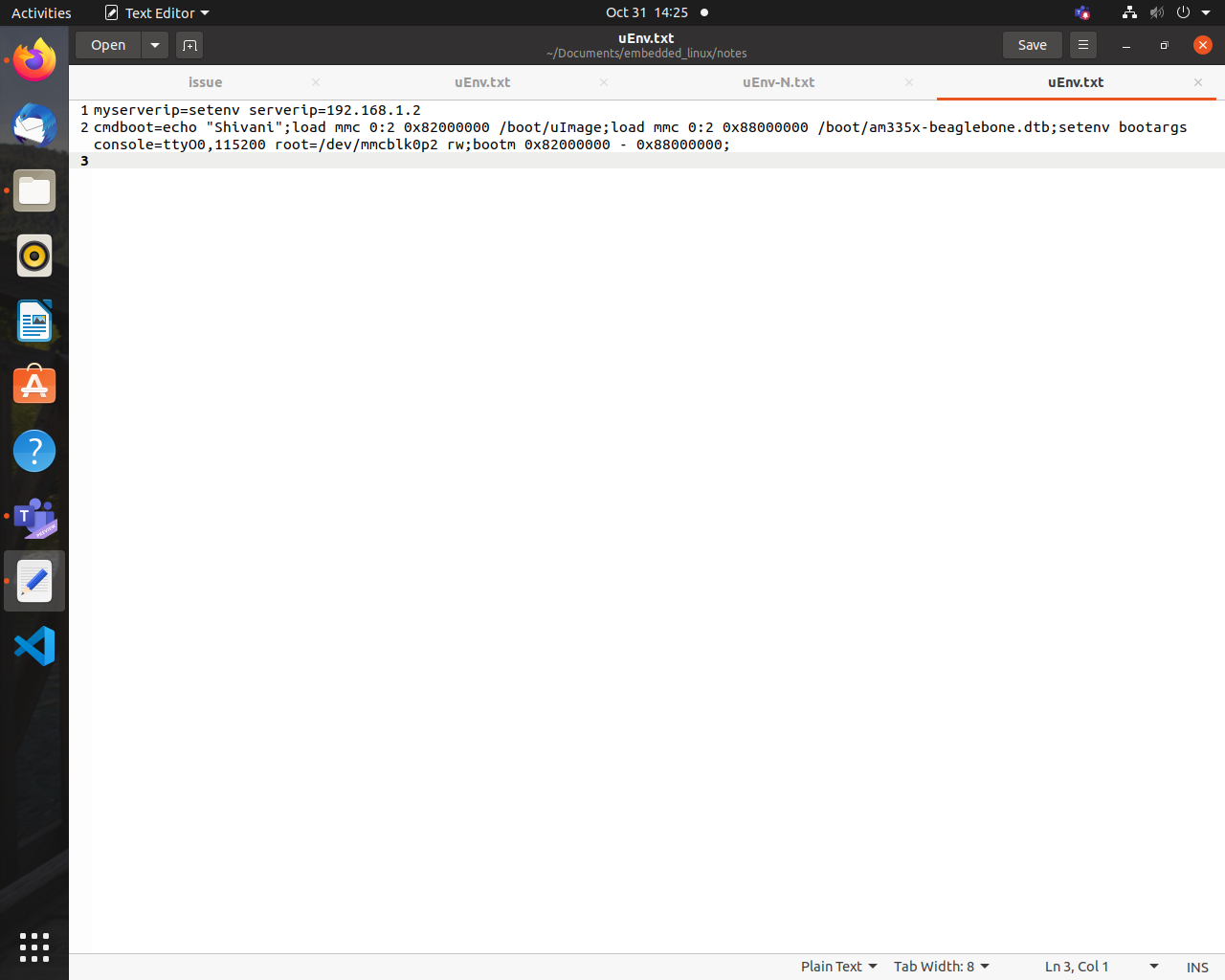
* Step 3:

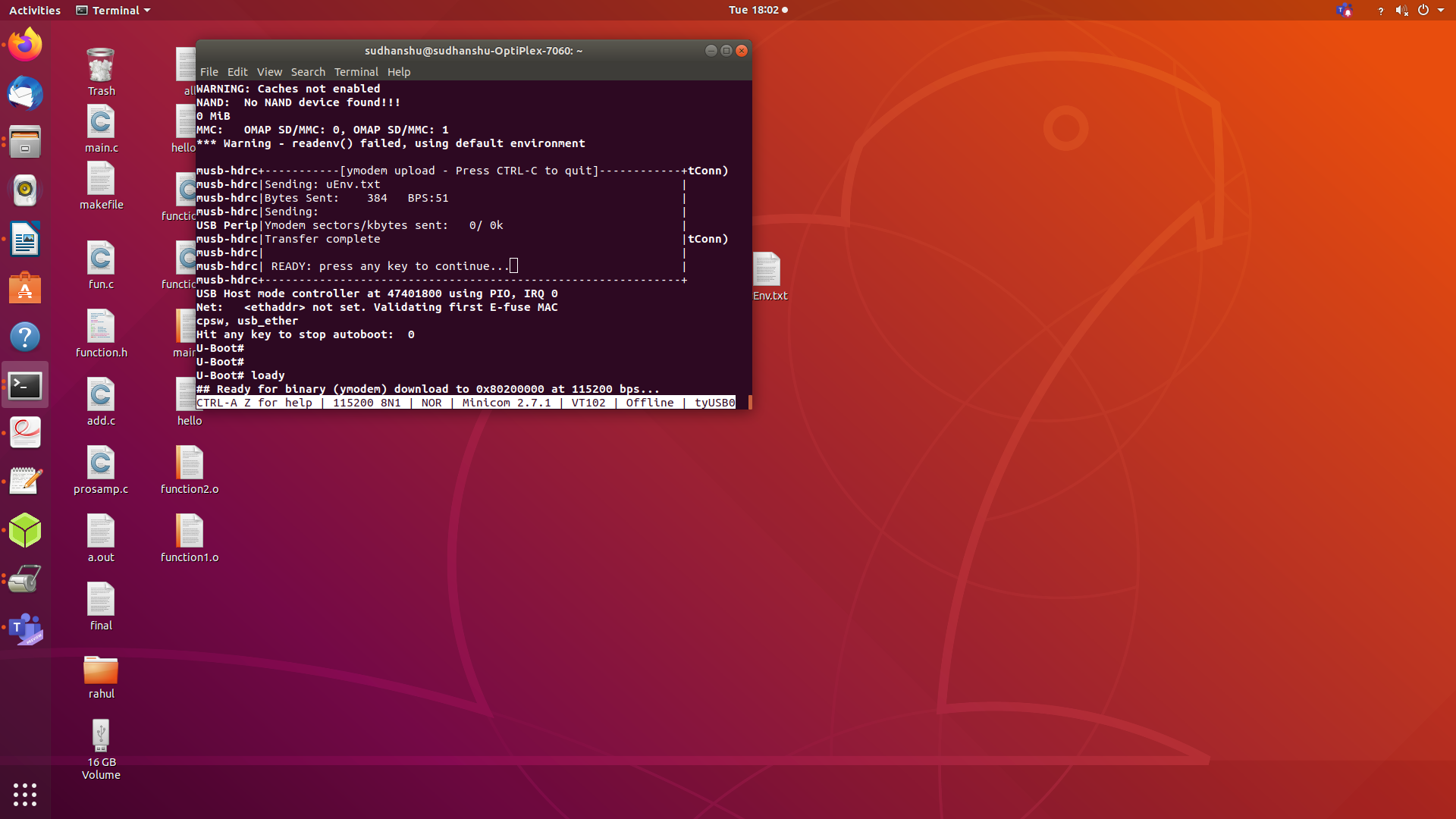
The variable bootm should contain all the booting and loading contents to be taken from the host.

* Step 4:

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.

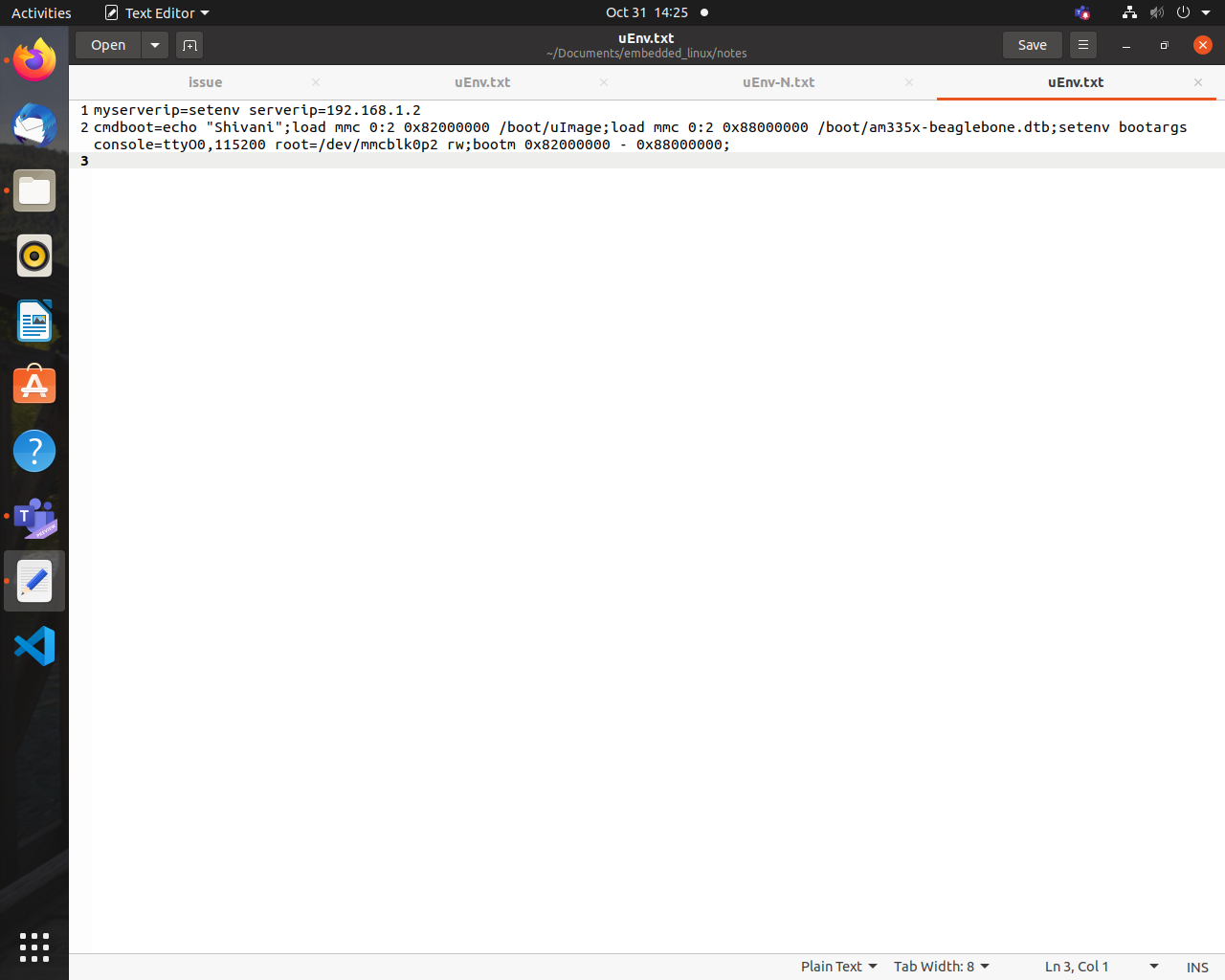
cmdboot=echo”Shivani”;load mmc 0:2 0x82000000 boot/uImage; load mmc 0:2 0x88000000 boot/am335x-beaglebone.dtb.setenv bootargs console=ttyO0,115200 root=/dev/mmcblk0p2 rw;bootm 0x82000000 – 0x88000000;

****



# **Activity 9-Challenge-Write a uEnvt.txt file to automate TFTP boot.**

# **Activity 10-Challenge-Write a generic uEnv.txt**

****

# 

# **Activity 11-Challenge-Increase the AUTOLOAD timings .**

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

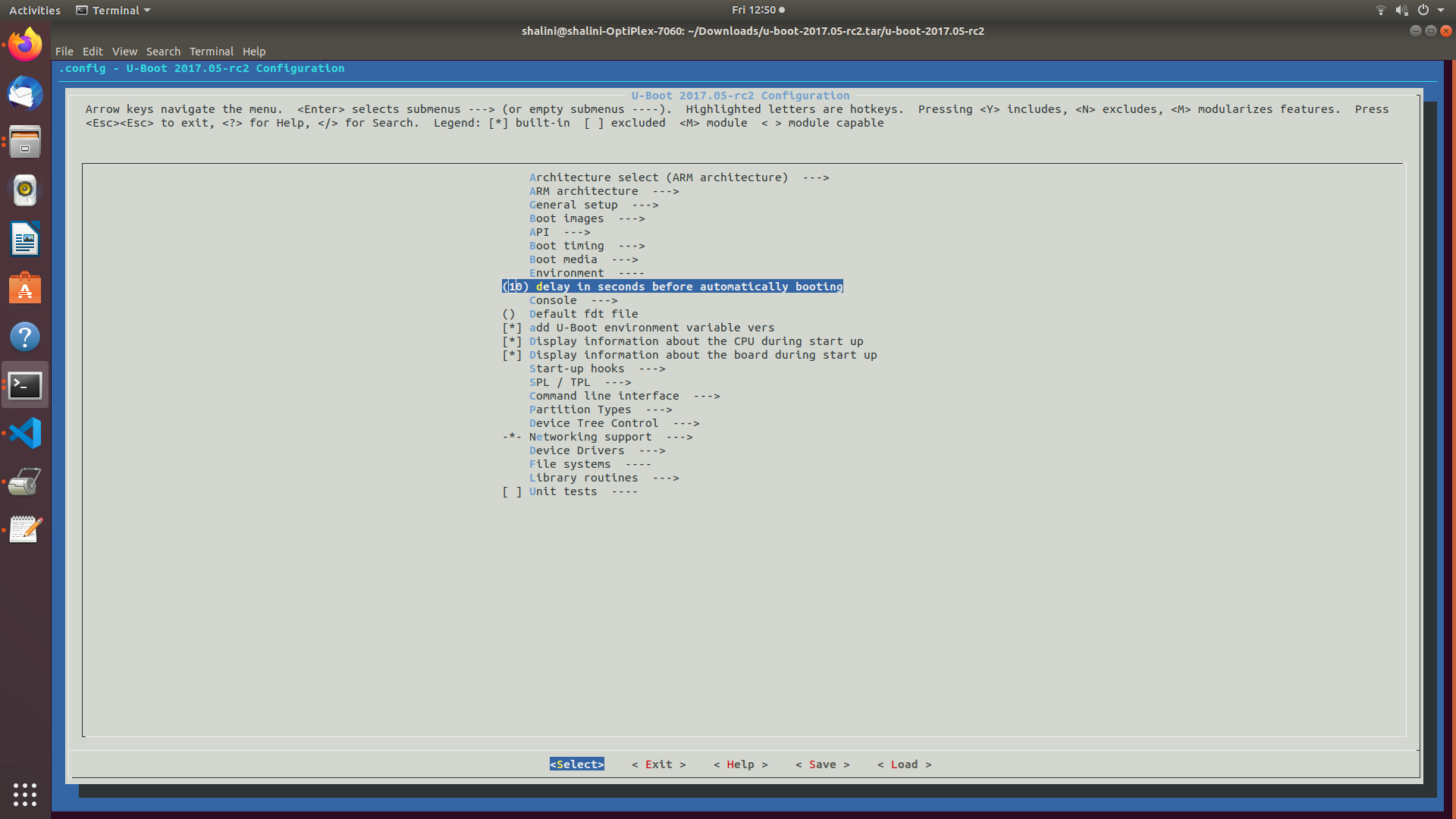
* STEP 1: distclean : deletes all the previously compiled/generated object files.

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.

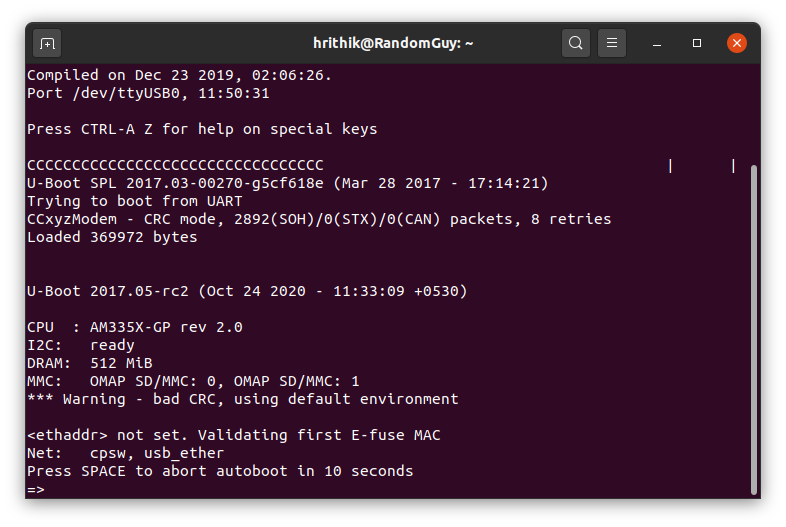


* STEP 4 : compile

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

to spawn 4 threads.

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



# **Activity 12-Challenge-Busybox "Dynamic" Compilation**

Static Linking and Static Libraries is the result of the linker making copy of all used library functions to the executable file. Static Linking creates larger binary files, and need more space on disk and main memory. Examples of static libraries (libraries which are statically linked) are, *.a* files in Linux and *.lib* files in Windows.Static libraries occupy lot of space.They occupy space in kernel.Hence when memory is not a constraint we use static library.

Dynamic Linking doesn’t require the code to be copied, it is done by just placing name of the library in the binary file. The actual linking happens when the program is run, when both the binary file and the library are in memory. Examples of Dynamic libraries (libraries which are linked at run-time) are, .so in Linux and .dll in Windows.Dynamic libraries occupy less space.They do not occupy space in kernel.

Steps for Busybox "Dynamic" Compilation

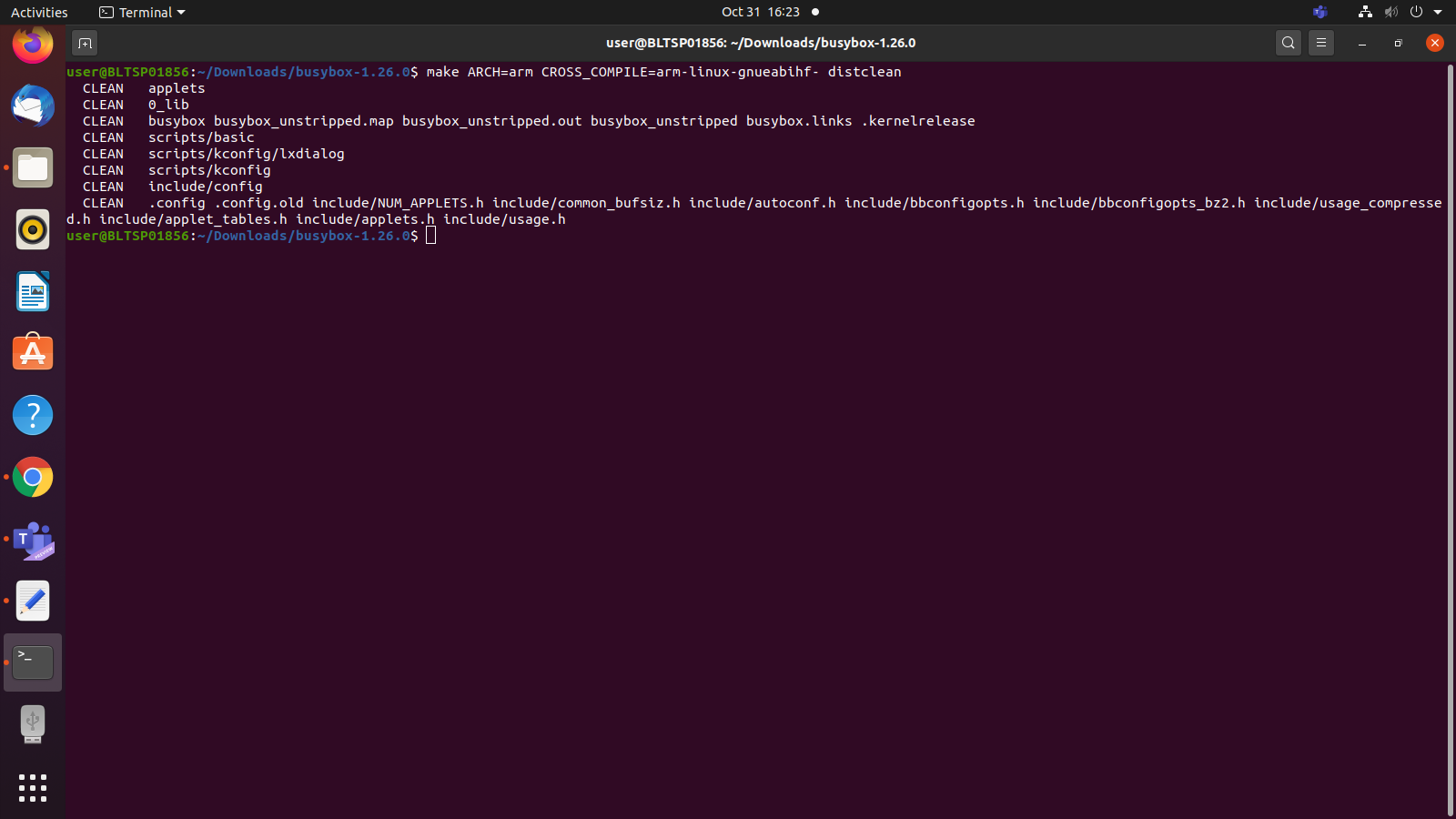
(open terminal)

STEP 1: Download busybox

https://busybox.net/

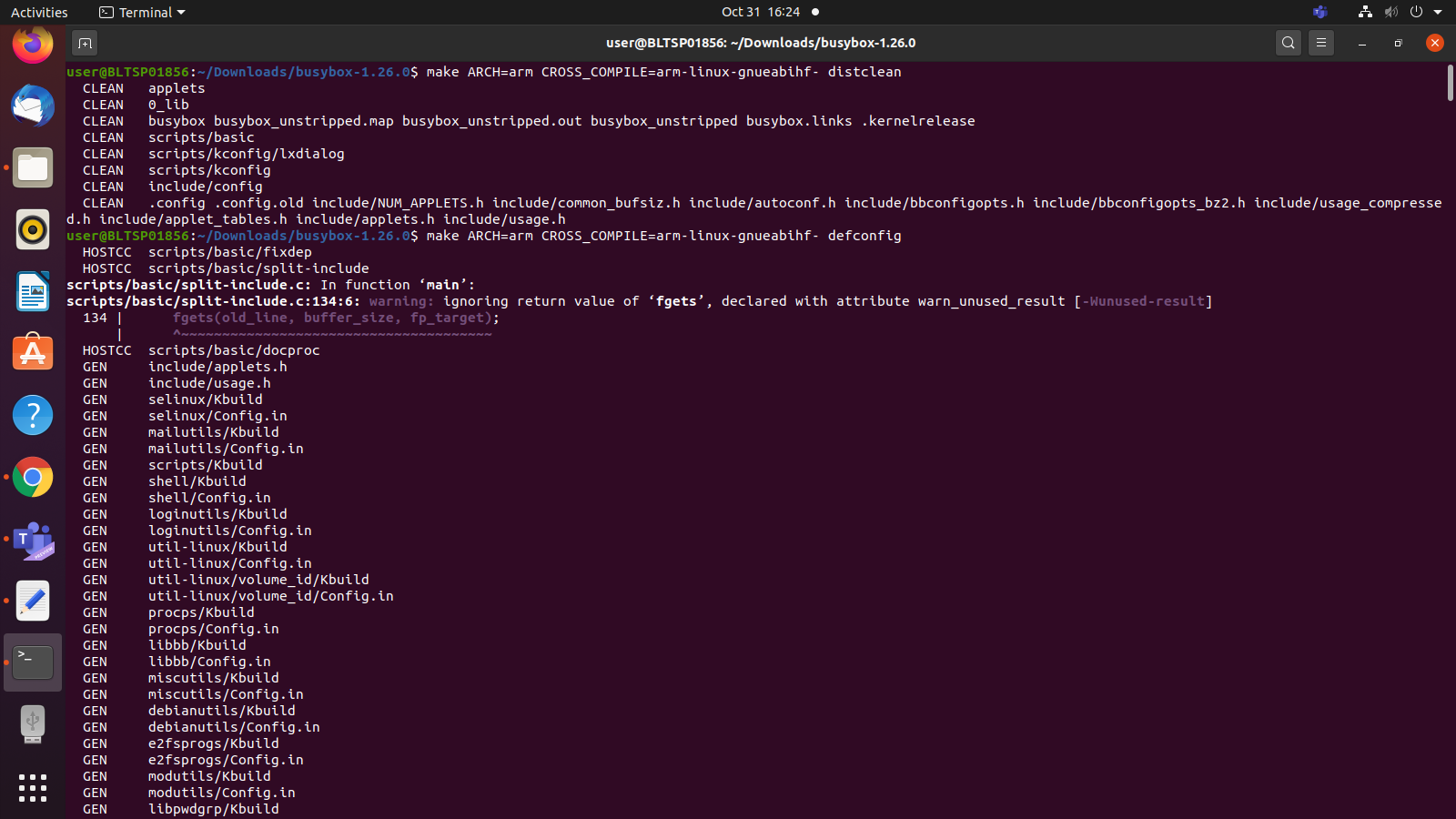
STEP 2 : Clean the earlier files

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



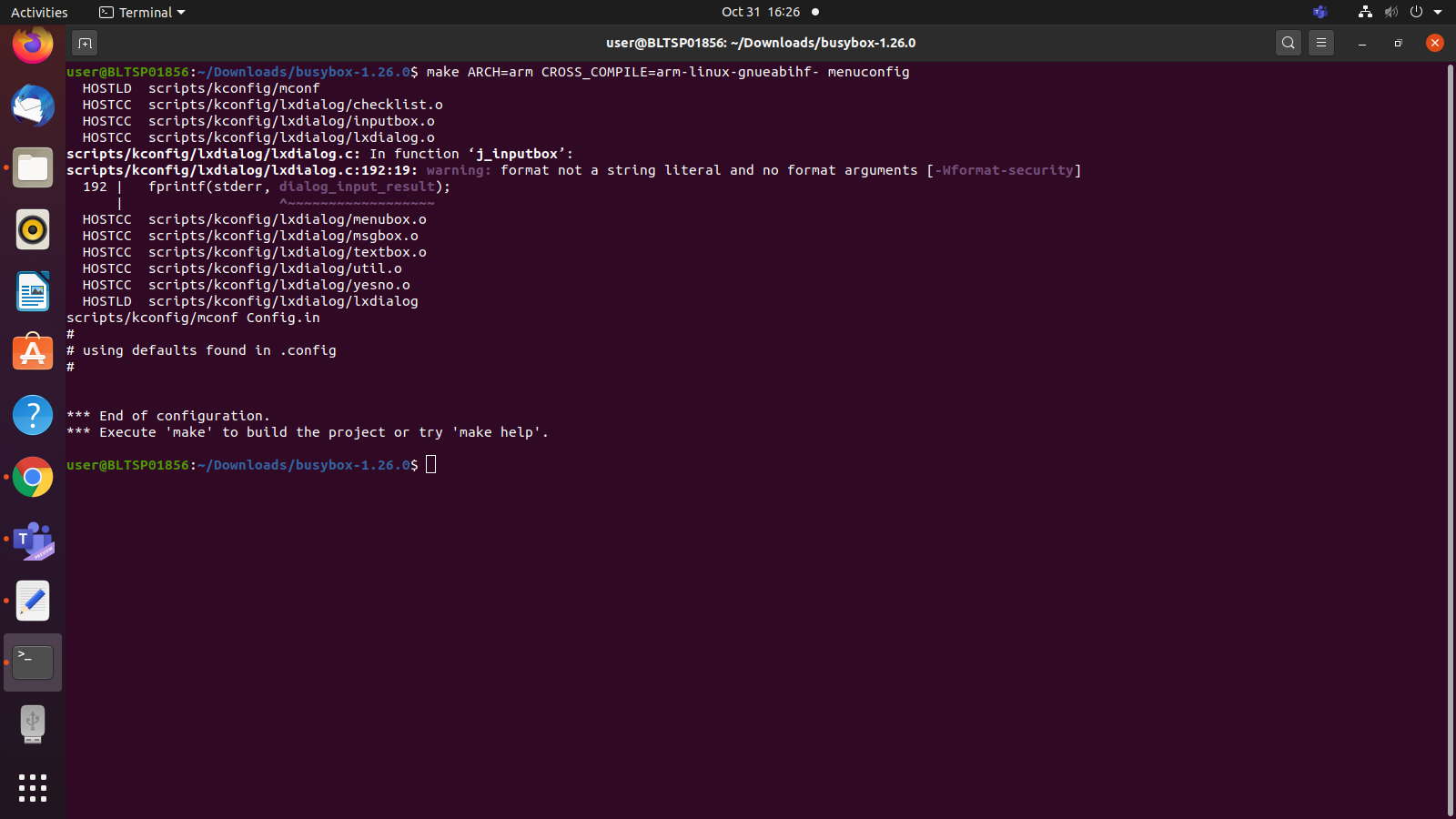
STEP 3 : Apply default configuration

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

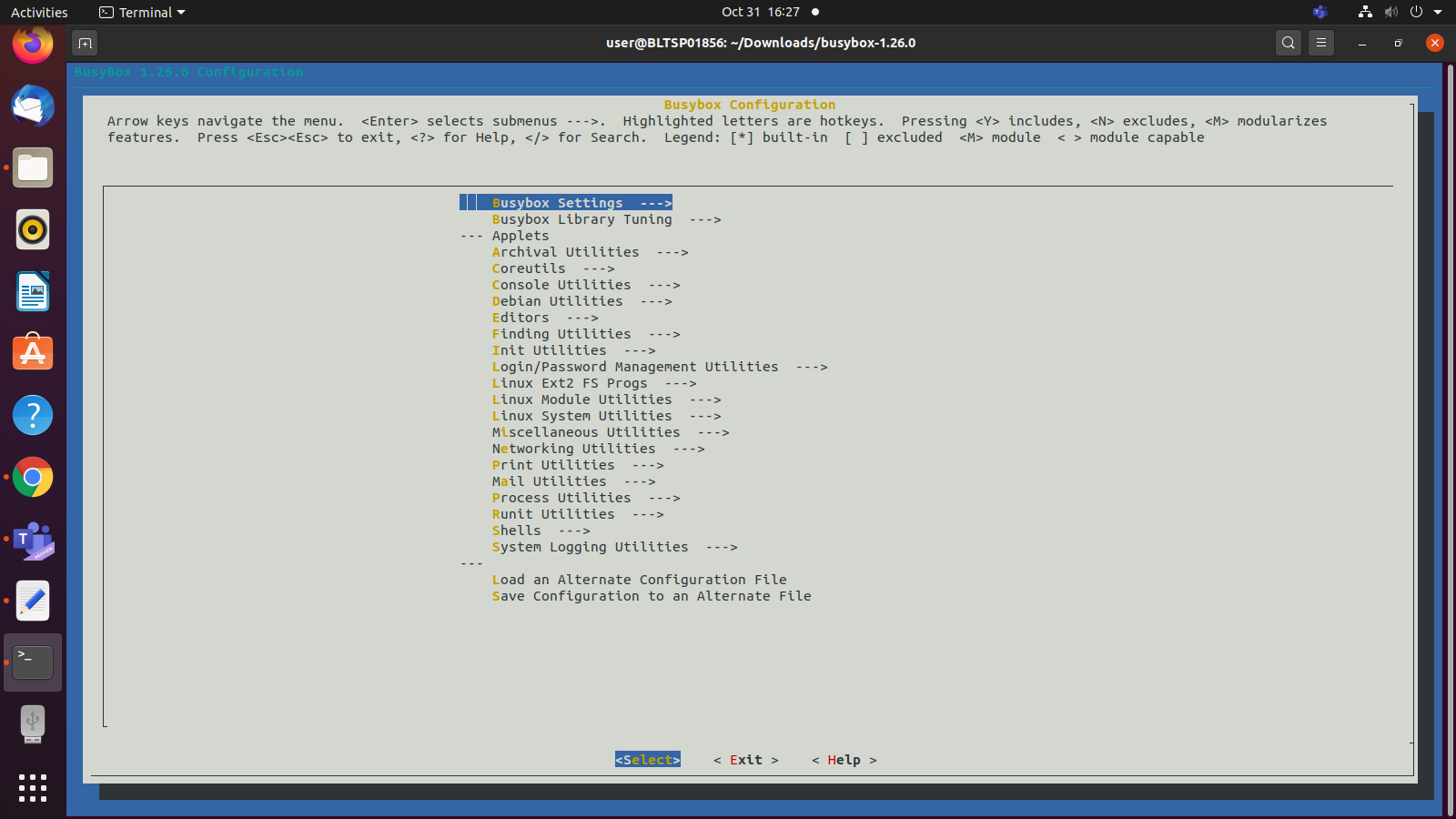


STEP 4 : Change default settings.

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



STEP 5 :Menu Pops up



STEP 6 : Change default settings.

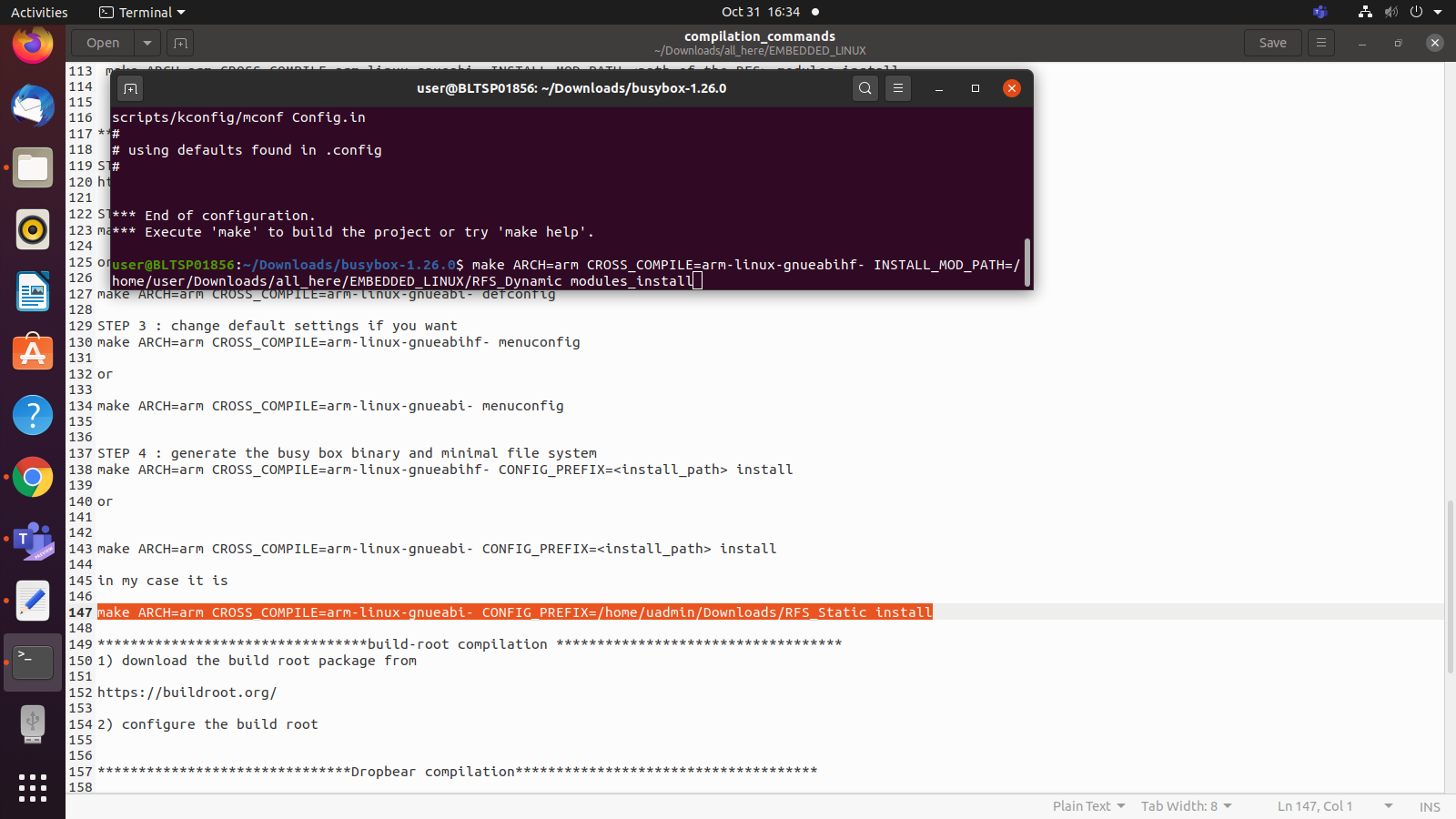
go in Busybox Setting->Build shared libbusybox

press space and save the setting.

# 

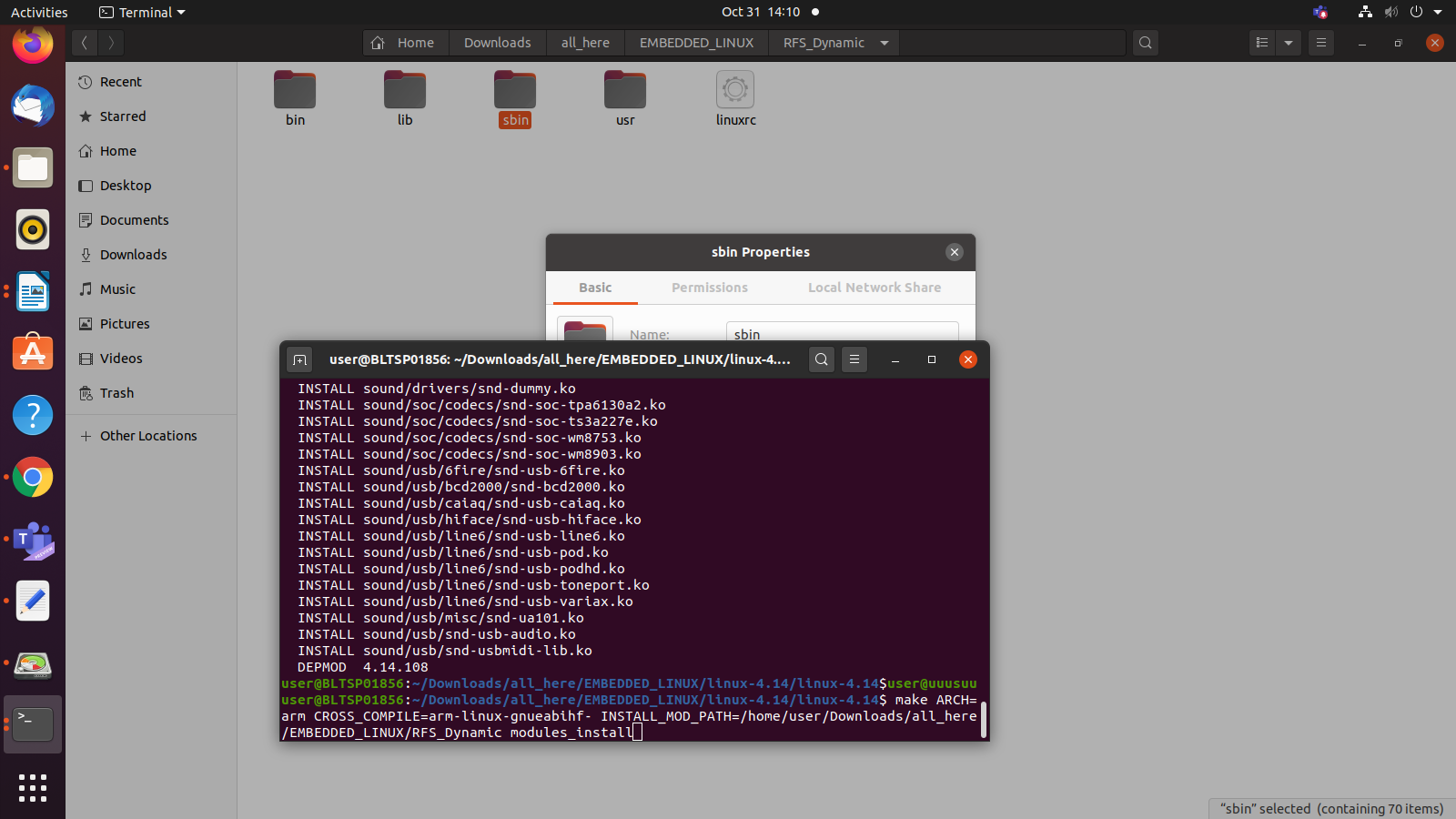
STEP 7 : Generate the busy box binary and minimal file system

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

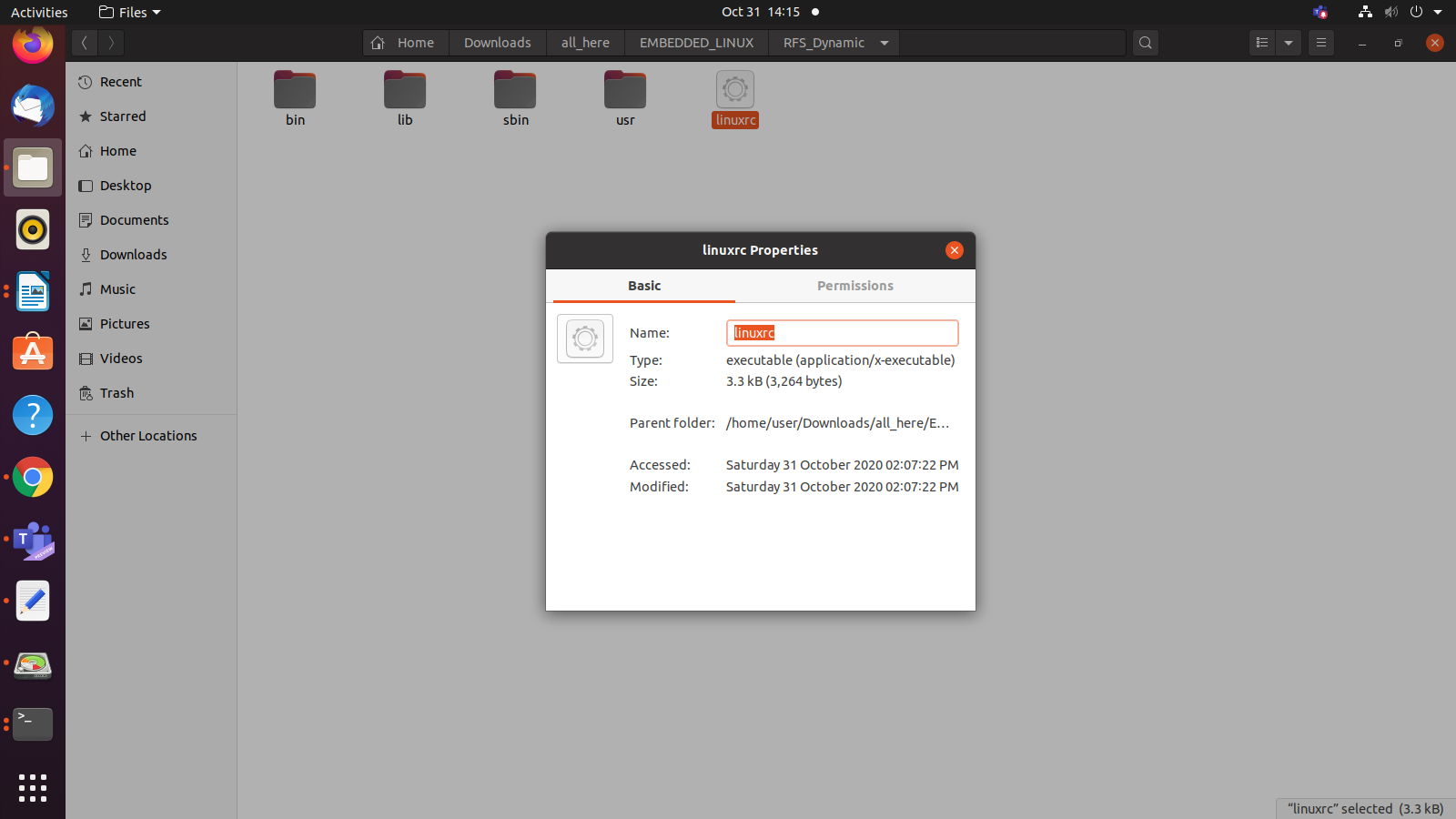


STEP 7 : Open terminal in Linux-4.14

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- INSTALL\_MOD\_PATH=<path of the RFS>

modules\_install

STEP 8 :Dynamic files are generated.We can observe the drastic change in file size



# Reference

[1]Step by step configuration[https://www.youtube.com/watch?v=UMEUo6Wm6u4& list=PLGs0VKk2Di...](https://www.youtube.com/watch?v=UMEUo6Wm6u4&list=PLGs0VKk2DiYyThNvj6VyDFmOnQ8ncXk8b&index=1)

[2]Step by step configuration-[https://www.youtube.com/watch?v=c81tmb7WJxw&list=PLG s0VKk2Di...](https://www.youtube.com/watch?v=c81tmb7WJxw&list=PLGs0VKk2DiYyThNvj6VyDFmOnQ8ncXk8b&index=2)

[3]Beagleboard-https://beagleboard.org/black

[4]Evolution of BBB-https://elinux.org/Beagleboard:BeagleBoneBlack#Revision\_C\_. 28Production\_Version.29

[5]USB to TTL logic-https://www.robotics.org.za/W7965

[6]TI PROCESSOR-https://www.ti.com/lit/ug/swpu270t/swpu270t.pdf

[7]Different versions of BBB-https://en.wikipedia.org/wiki/BeagleBoard