Dpto de Telemática y Electrónica
Universidad Politécnica de Madrid

Embedded Systems

Using Buildroot for cross-compiling and integrating EPICS into Raspberry Pi

Carlos Javier Hernández Lahera-Mariano Ruiz
V 1.0

2017


EPICS
BuildRoot
Making Embedded Linux Easy

Table of contents

Table of figures

1 SCOPE

1.1 Project Overview

The objective of this project is the generation of an embedded Linux distribution for the Raspberry Pi platform, and the inclusion of an operational version of the EPICS System (Experimental Physics and Industrial Control System).

The necessary basics to generate a distribution of Linux have been learned, as the boot loader, kernel, the file system, and the differences between the different C libraries available (Uclibc, glibc).

A cross-compile environment using the Buildroot tool has also been developed and set up, in order to generate an EPICS distribution in the Raspberry Pi platform.

1.2 Method and working stages

- Become familiar with the Raspberry Pi hardware platform and its possibilities.
- Installation of a standard Linux distribution (Raspbian) in the Raspberry Pi.
- Installing and running an EPICS softIOC on the Raspbian distribution.
- Mounting of a virtual machine with VMware, in order to run Linux distribution Ubuntu 14.04.
- Setting up a cross-compile environment with Buildroot in the virtual machine.
- Generation of a minimal distribution of Linux using the previous environment.
- Setting up of this Linux distribution, so that EPICS can be included (use of glibc library, Perl, etc.)
- Use of the cross-compile environment to generate EPICS and to include it in our Linux distribution.
- Running an EPICS softIOC from the Linux distribution.
- Monitoring of the softIOC using different clients (such as Control System Studio)

1.3 Elements used

- Low-cost hardware platform Raspberry Pi 3
- USB to RS232 adapter cable with FTDI chipset
- Laptop with Linux operating system Ubuntu version 16.04 64-bit
- VMware software tool to generate virtual machine (Ubuntu 14.04 32bit)
- Embedded systems generation tool Buildroot version 2016.11
- EPICS base version 3.14.12.6 software and Open Source tools
- Control System Studio tool for EPICS

1.4 Acronyms

rpi	Raspberry Pi
CPU	Central Processing Unit
EABI	Extended Application Binary Interface
I/O	Input and Output
MMC	Multimedia card
OS	Operating system
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
FTDI	Future Technology Devices International
SoC	System on a Chip
GPIO	General Purpose Input/Output
CSS	Control System Studio
EPICS	Experimental Physics and Industrial Control System
IOC	Input/Output Controller
CA	Channel Access
PVs	Process Variables
JRE	Java Runtime Environment

2 REFERENCED DOCUMENTS

2.1 References

[RD1] "Embedded Linux Systems: Using Buildroot for building Embedded Linux Systems V2.1"
Authors: Mariano Ruiz y Francisco Javier Jiménez, 2016

[RD2] "Embedded Linux system development", 2017, Free Electrons

[RD3] "EPICS Application Developer's Guide", 2016

[RD4] Ubuntu 14.04 user's guide

[RD5] Buildroot User Manual

[RD6] Raspberry-Pi User Documentation

3 RASPBIAN OPERATING SYSTEM ON THE RASPBERRY PI

3.1 About Raspberry Pi

Created by the Raspberry Pi Foundation, the Raspberry Pi is an Open Source, small, low-cost, credit card format computer, based on Linux. Throughout its history different models have been developed (see Fig. 1).

	Pi A+	Pi B+	Pi 2 B	Pi 3 B	Compute Module
Dimensions	66 x 56 x 14mm	85 x 56 x 17mm	85 x 56 x 17mm	85 x 56 x 17mm	67.5 x 30mm
SoC	BCM2835	BCM2835	BCM2836	BCM2837	BCM2835
Processor Core	ARM11	ARM11	ARM Cortex-A7	ARM Cortex-A53	ARM11
Processing Power	700 MHz	700 MHz	900 MHz	1.2 GHz	700 MHz
Memory	256 MB	512 MB	1 GB	1GB LPDDR2	512 MB
Ports	1x USB 2.0	4x USB 2.0 1x 10/100 Ethernet	4x USB 2.0 1x 10/100 Ethernet	4x USB 2.0 1x 10/100 Ethernet	N/A
GPIO	40	40	40	40	N/A

Fig. 1: Characteristics of Raspberry Pi different models.

Source: http://uk.rs-online.com/webdocs/14bc/0900766b814bcef0.pdf

Operating system is installed on a microsd card, and starts as soon as the rpi is powered up.
There are different operating systems available, but the most extended is Raspbian, a Linux version based on Debian and specifically adapted for Raspberry Pi.

Fig. 2 shows in detail main components of a Raspberry Pi 3B model. As we can see, it features several USB ports, Ethernet, Wifi, Bluetooth, HDMI connector, video and audio RCA connector, and GPIO port connector.

More information is available at: https://www.raspberrypi.org/ and [RD6]

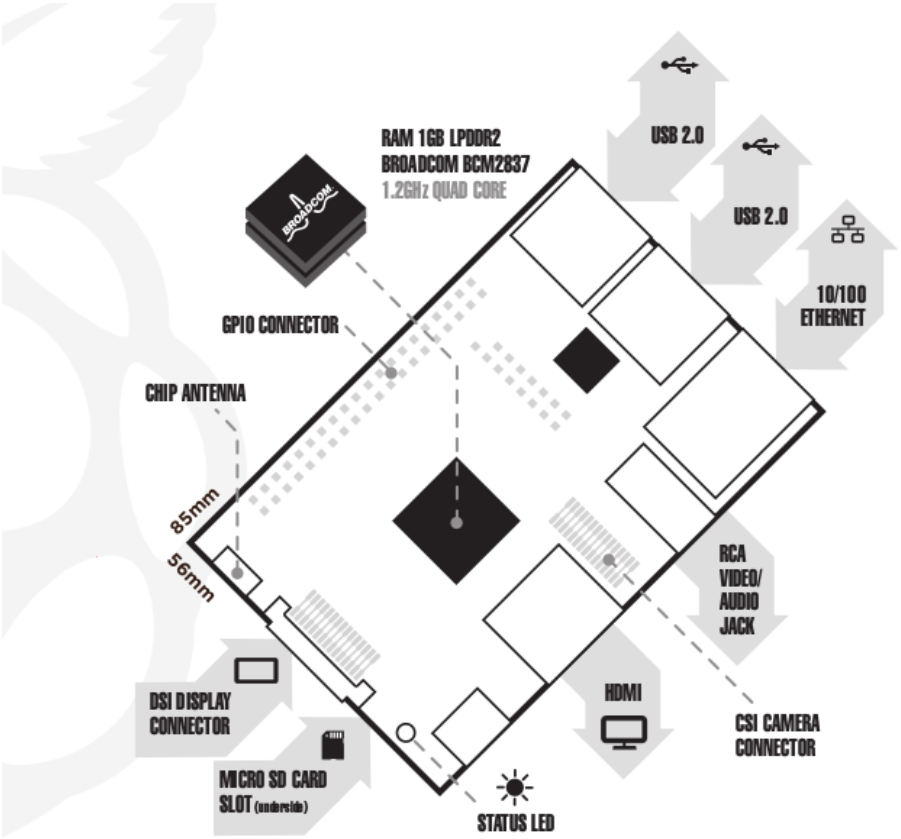


Fig. 2: Raspberry Pi 3B model main components.

3.2 Installing Raspbian

A Raspberry Pi 3B model card (Fig. 3) has been used for all the process outlined in this document. It features integrated wifi and Bluetooth.

Fig. 3: Raspberry Pi 3B.

3.2.1 Setting up process

The following sources have been used in the process of installing the operating system on the rpi:
Raspbian Jessie OS description:

https://www.raspberrypi.org/blog/raspbian-jessie-is-here/
Installation guides:
https://www.raspberrypi.org/documentation/installation/installing-images/README.md
https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

Raspbian "Jessie with Pixel" version can be downloaded from here:
https://www.raspberrypi.org/downloads/raspbian/

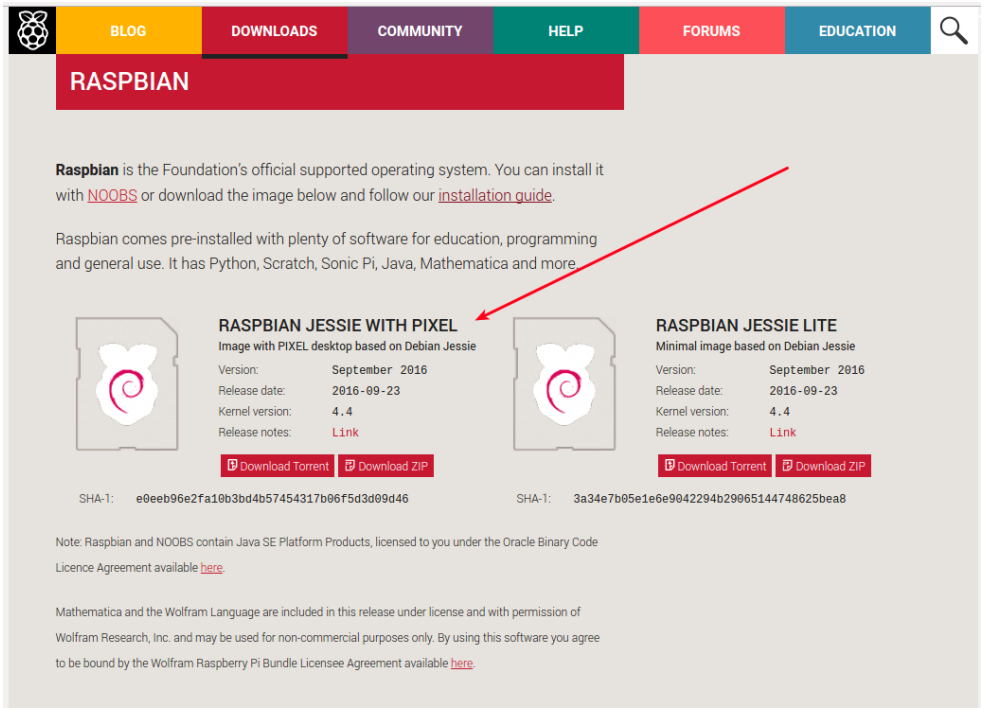


Fig. 4: Raspbian download web page.

Downloaded image file (.zip) is approximately 1.5 Gigabytes in size, with release date 2016-09-23.

Unzip that file will give us and .img file, 4.3 GB in size.
In order to load that image file into a microsd card, we can use Windows or Linux, following one of these guides:
Windows: https://www.raspberrypi.org/documentation/installation/installing-images/windows.md
Linux: https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

Information about suitable microsd cards for this purpose is available here: https://www.raspberrypi.org/documentation/installation/sd-cards.md

Once the image is loaded on the microsd card, we need to put the card in the rpi, connect a keyboard, mouse, HDMI cable to TV, ethernet cable, and power up the rpi.
http://frambuesa-pi.blogspot.com.es/2015/11/raspberrypi-puesta-en-marcha-paso-paso.html

First boot will be under way, and booting screen will be shown.

We have access to the menus by pressing raspberry icon on the upper left. To change the language of the system, we are going to "Preferences -> Raspberry Pi -> Configuration" (Fig. 5).

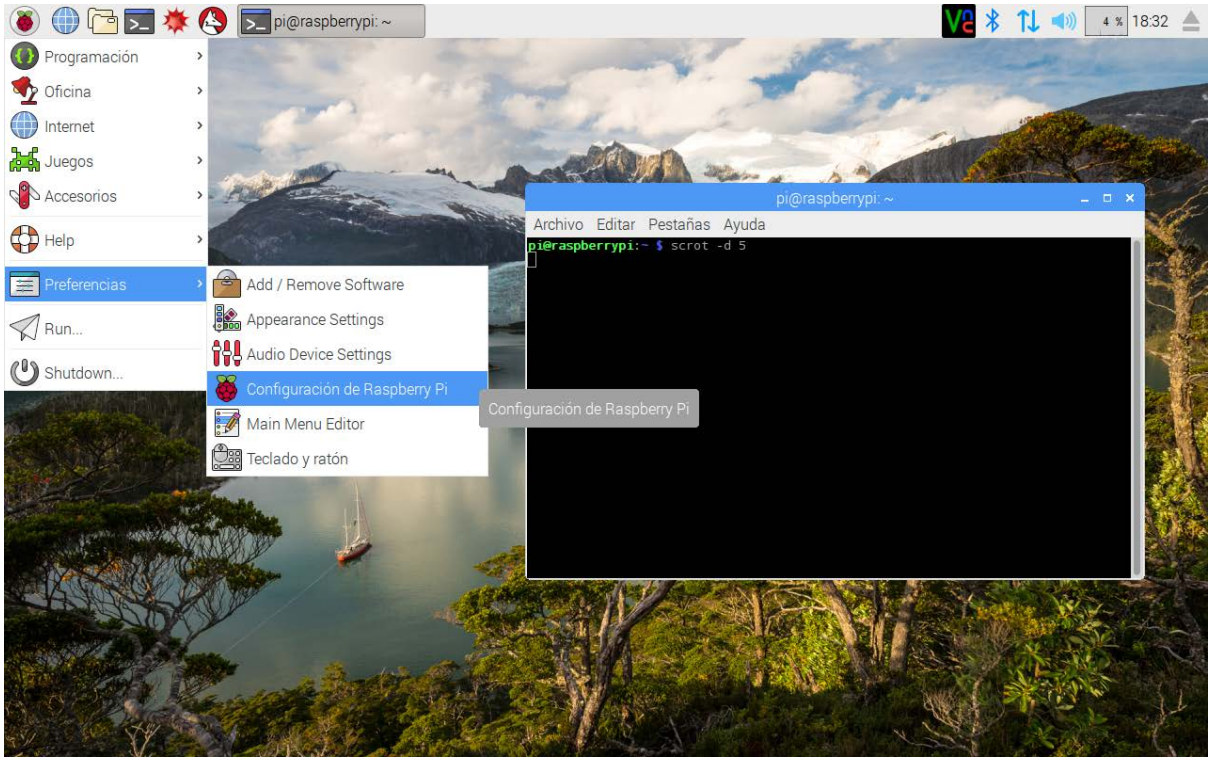


Fig. 5: Configuring Raspbian options.

Four tabs are available: “System”, “Interfaces”, “Performance” and “Localisation”
“System” tab (Fig. 6), allow us to modify rpi interface environment (Desktop or Command Interface CLI), activate auto login, or expand filesystem, to use the complete microsd card capacity for Raspbian.

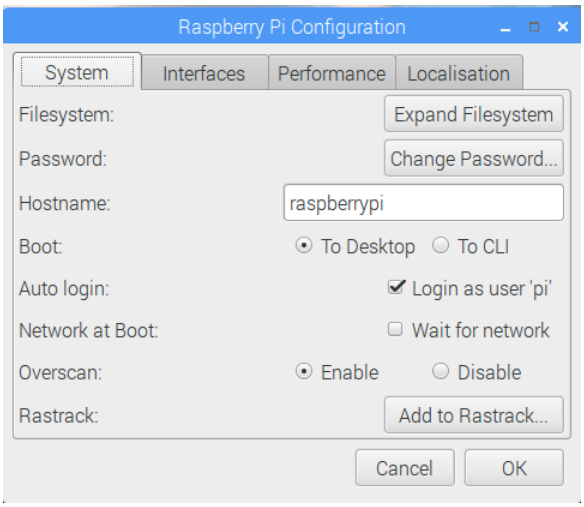


Fig. 6: Raspbian configuration System tab.

“Interfaces” tab (Fig. 7), allow us to define which services will be active after start-up, like camera, SSH, VNC, SPI, I2C, serial port, etc.

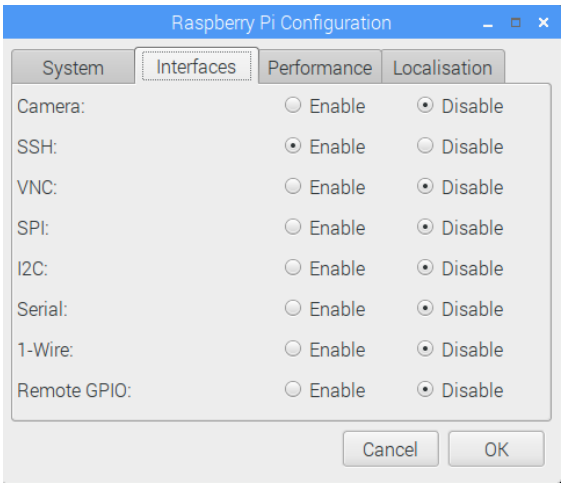


Fig. 7: Raspbian configuration Interface tab.

"Performance" tab (Fig. 8) allows us to change overclock settings and GPU memory.

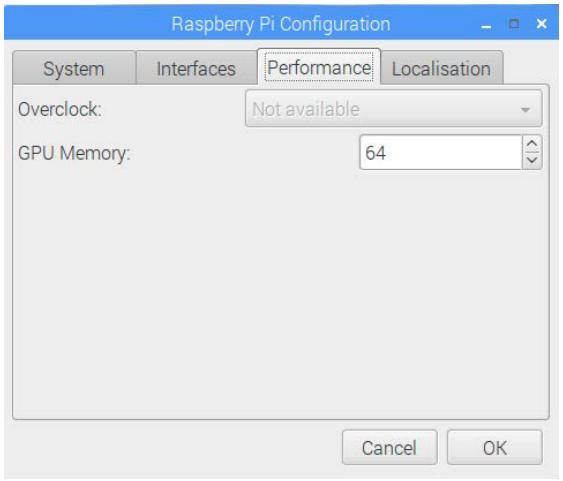


Fig. 8: Raspbian configuration Performance tab.

"Localisation" tab (Fig. 9), allow us to change time zone, keyboard language, and wifi country-related settings.

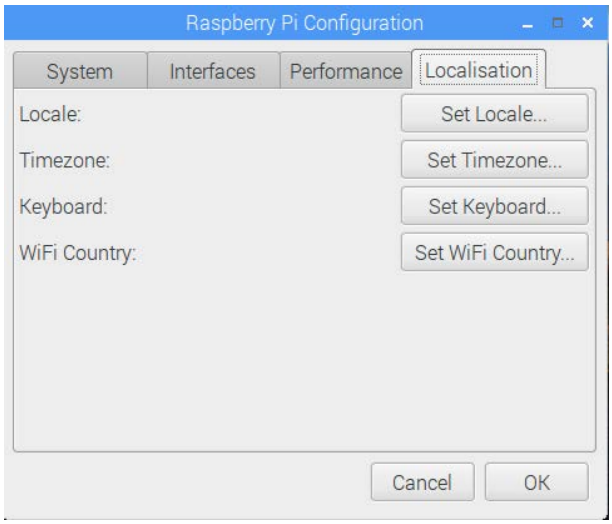


Fig. 9: Raspbian configuration Localisation tab.

3.2.2 Configuring language and keyboard

First of all we are going to change the language, by pressing on "Set Locale", a sub window appears to choose the language, the country and the character set. In "Language" (see Fig. 10) we will choose "es(Spain)", in Country "Es(Spain)" and will choose default character set (UTF-8).

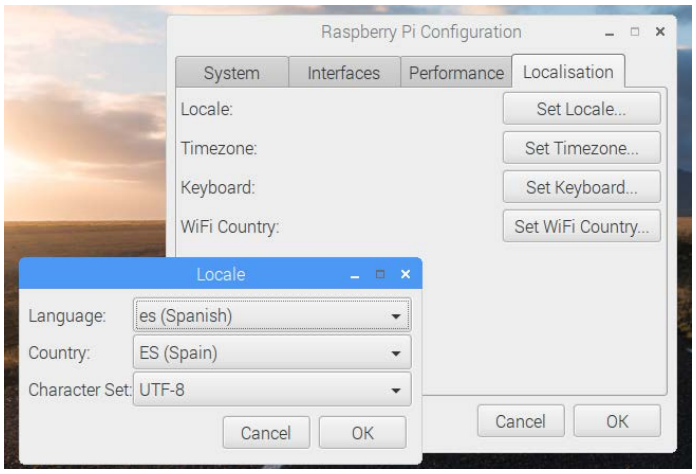


Fig. 10: Configuring country and language in Raspbian.

Then (Fig. 11) we click "Set Timezone", choose "Europe" in "Area" option, and "Madrid" in "Location". Click "OK", and a message will follow: "Setting timezone".

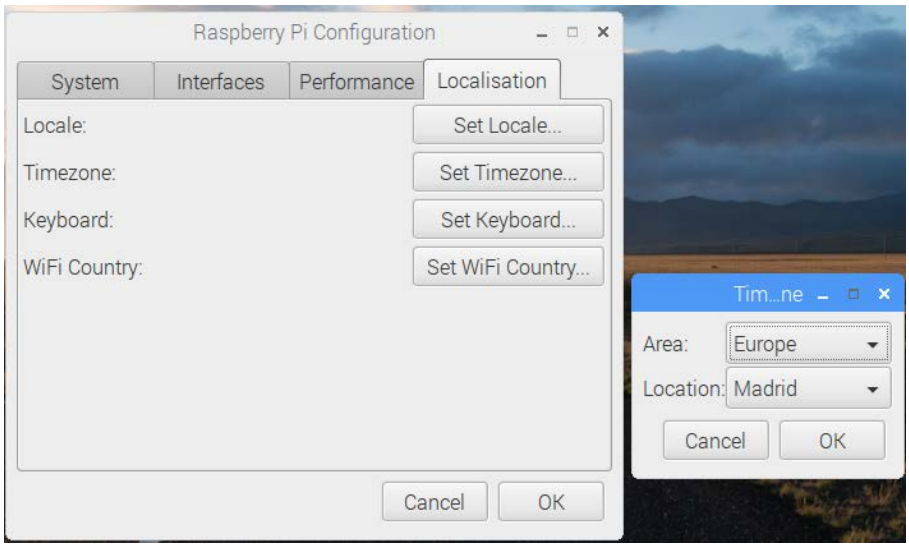


Fig. 11: Configuring time zone in Raspbian.

Then (Fig. 12) click “Set Keyboard”, choose country on the left side: “Spain” and on the right, default variant “Spanish”.

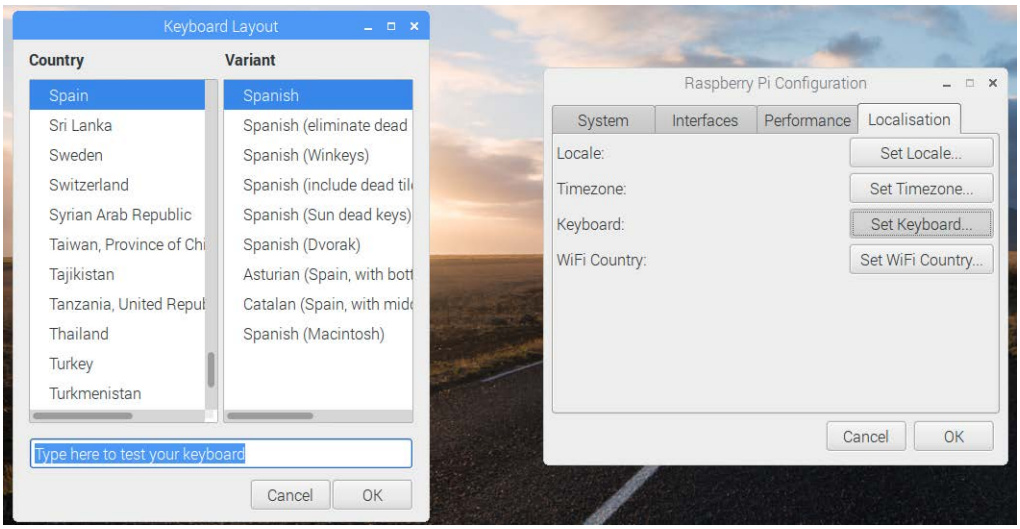


Fig. 12: Configuring keyboard in Raspbian.

You can test the keyboard by pressing keys on the bottom line. Click "OK" to apply the changes.
Then click “Set WiFi country” (Fig. 13), choose Country: “Es Spain” and click “OK”

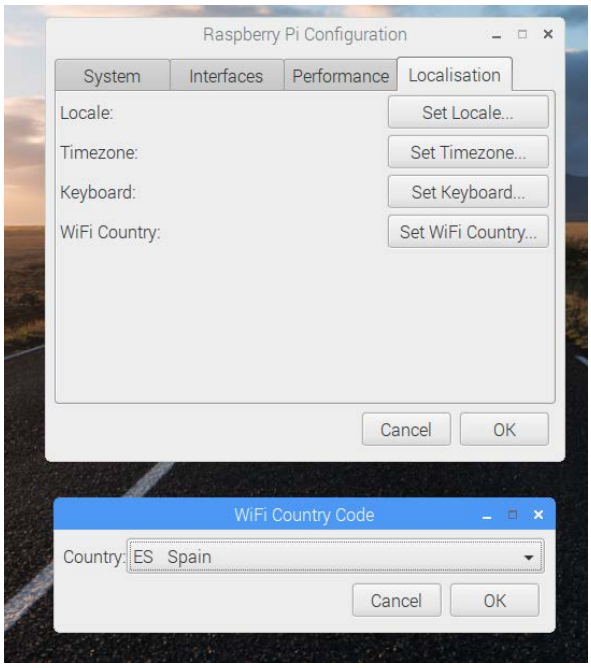


Fig. 13: Choosing wifi country in Raspbian.

Click "OK" again, and a message indicates that for the changes to take effect, you must restart the rpi.

3.2.3 Screenshot capture

"Scrot" is a screen capture utility, which is already installed on the system.
Screenshots are saved by default into "/home/pi" folder, and a screenshot will be taken just by pressing PrtScr key.
We can also open a command console, and type:

```
scrot -d 5
```

"5" meaning 5 seconds delay before capturing the screen.

3.2.4 Checking rpi hardware and software versions

We can find out our Raspberry Pi hardware version, just by typing: "*cat /proc/cpuinfo*"

```
Pi@raspberrypi:~ $ cat /proc/cpuinfo
Hardware        : BCM2709
Revision        : a02082
Serial          : 00000000d9a252d6
```

Possible results can be found at:
http://www.raspberrypi-spy.co.uk/2012/09/checking-your-raspberry-pi-board-version/

We can find out our software version, typing:

```
cat /proc/version
```

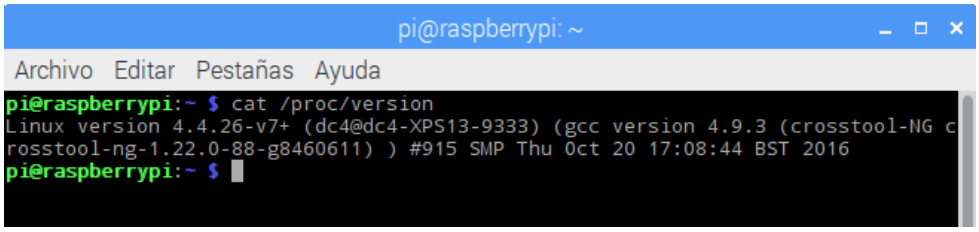


Fig. 14: Checking software version in Raspbian.

3.2.5 Expanding the file system

In our first installation, we have just copied the operating system image to the microsd card, so we are using only a part of its capacity. If we want to use full capacity of microsd card, we need to go to the Raspberry icon at upper left corner, then "Preferences" and "Configuring Raspberry pi". Select the tab "System" and click "Expand File System" option (Fig. 15).

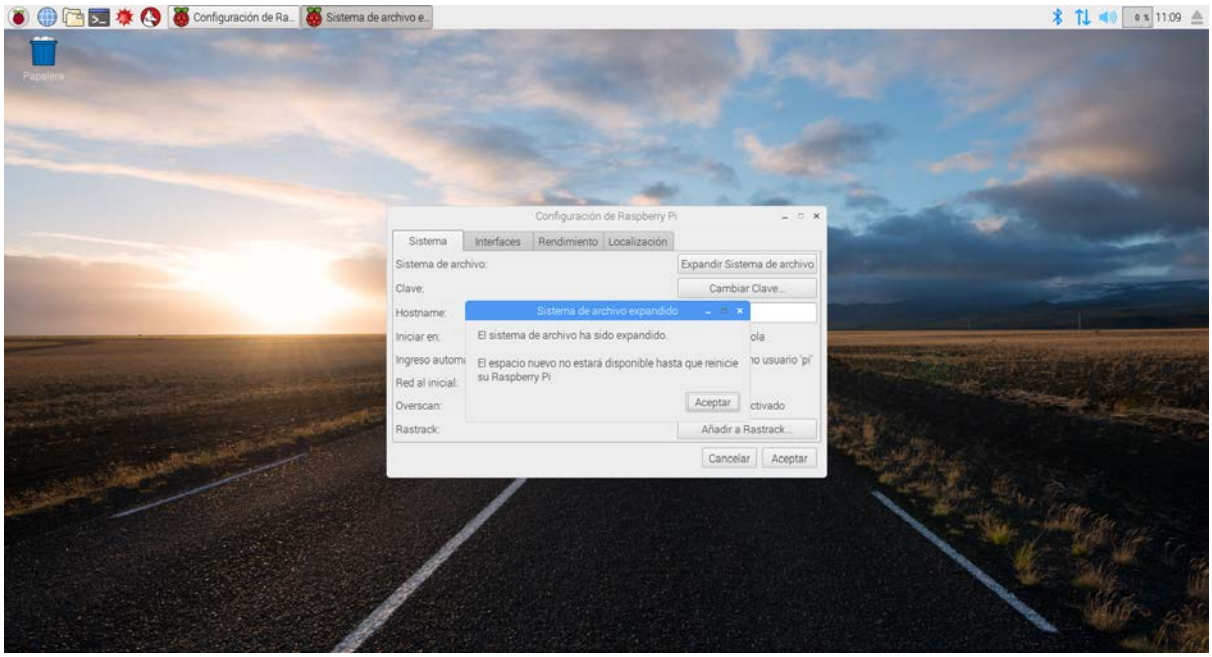


Fig. 15: Expanding file system in Raspbian.

After rebooting, we will be able to check with the file browser or with the command "*df -h*" that we have all available space on the microsd card at our disposal.

3.2.6 System update

Open a terminal window by clicking on the icon shown in Fig. 16, or by pressing keys Ctrl + Alt + T.

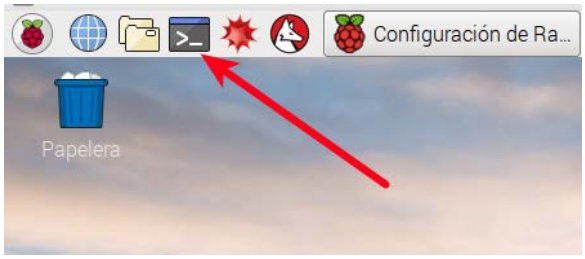


Fig. 16: Opening terminal window in Raspbian.

And execute the following commands:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

This way the operating system will be updated. See Fig. 17 and Fig. 18 for screenshots of the process.

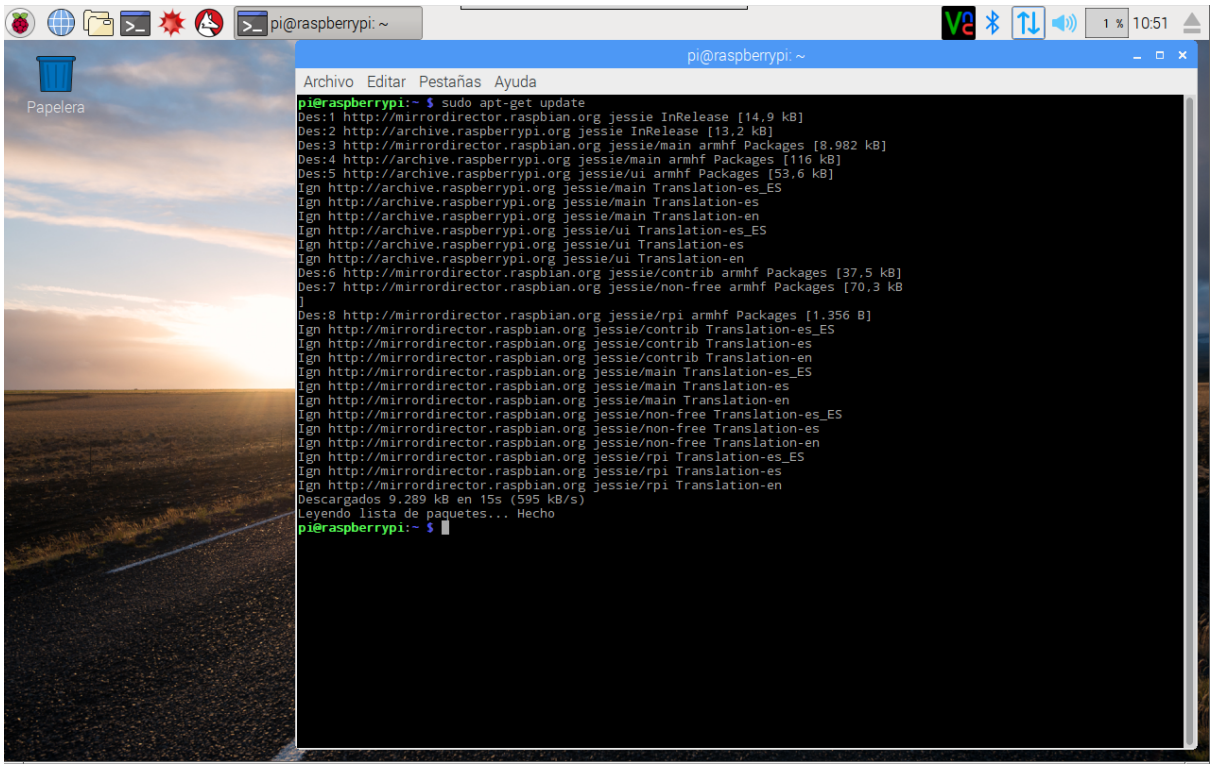


Fig. 17: Updating Raspbian OS.

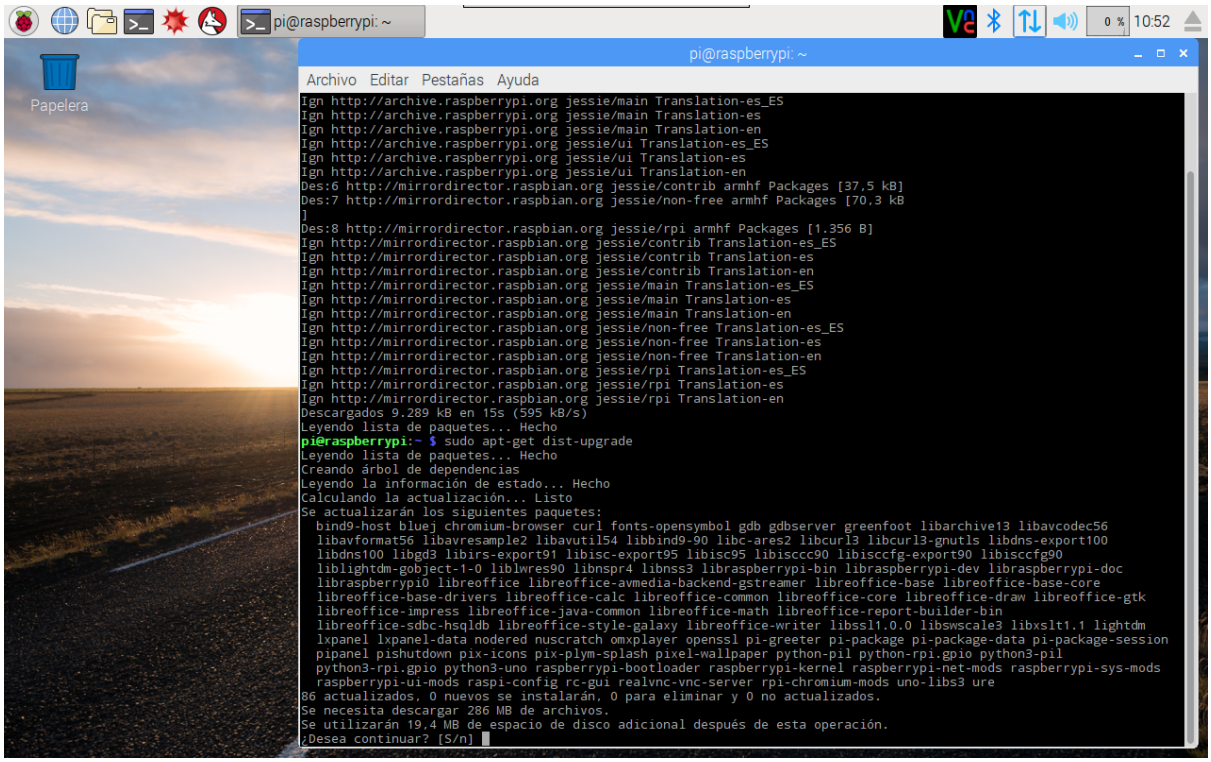


Fig. 18: Raspbian updating process.

The system will be updated with the latest patches and upgrades. If you obtain errors, you should execute this command:

```
sudo dpkg --configure -a
```

Flash Player install warnings may appear, or even some mods applications.
You will notice that a lot of programs and OS kernel will be updated in this process.

3.2.7 Connecting Raspberry Pi to a wifi network

Click on the connection icon in the top right, choose the wifi network to which you want to connect (Fig. 19), and you will be asked for the wifi password.

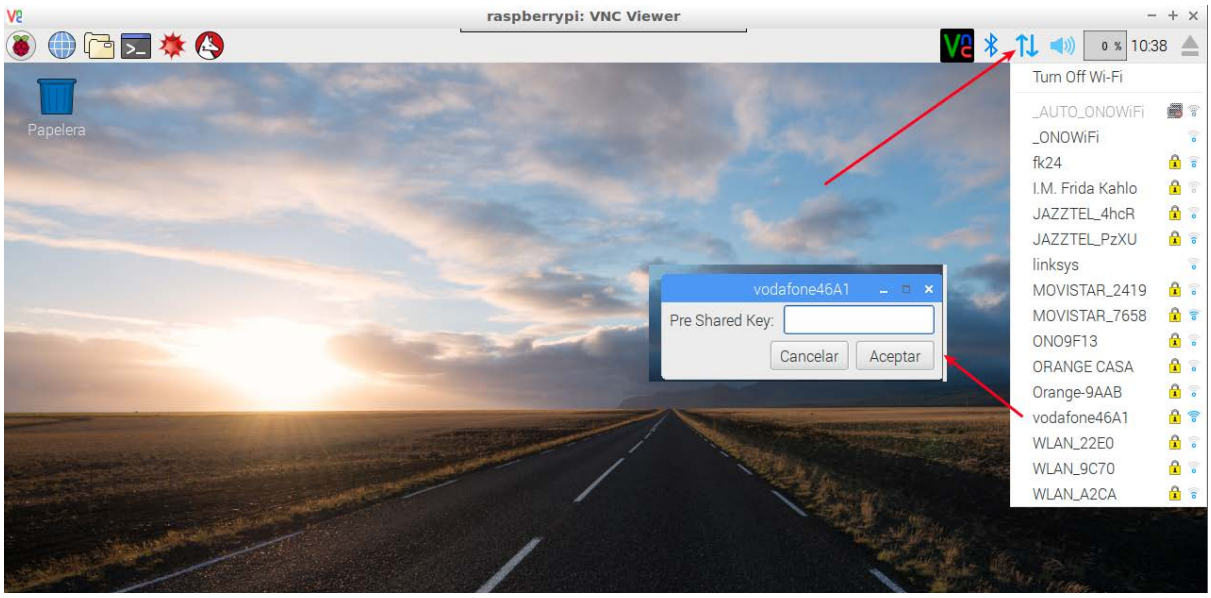


Fig. 19: Connecting to a wifi network in Raspbian.

A problem appeared the next day: the system had been upgraded automatically, and remote connection failed. The following message showed after boot-up:

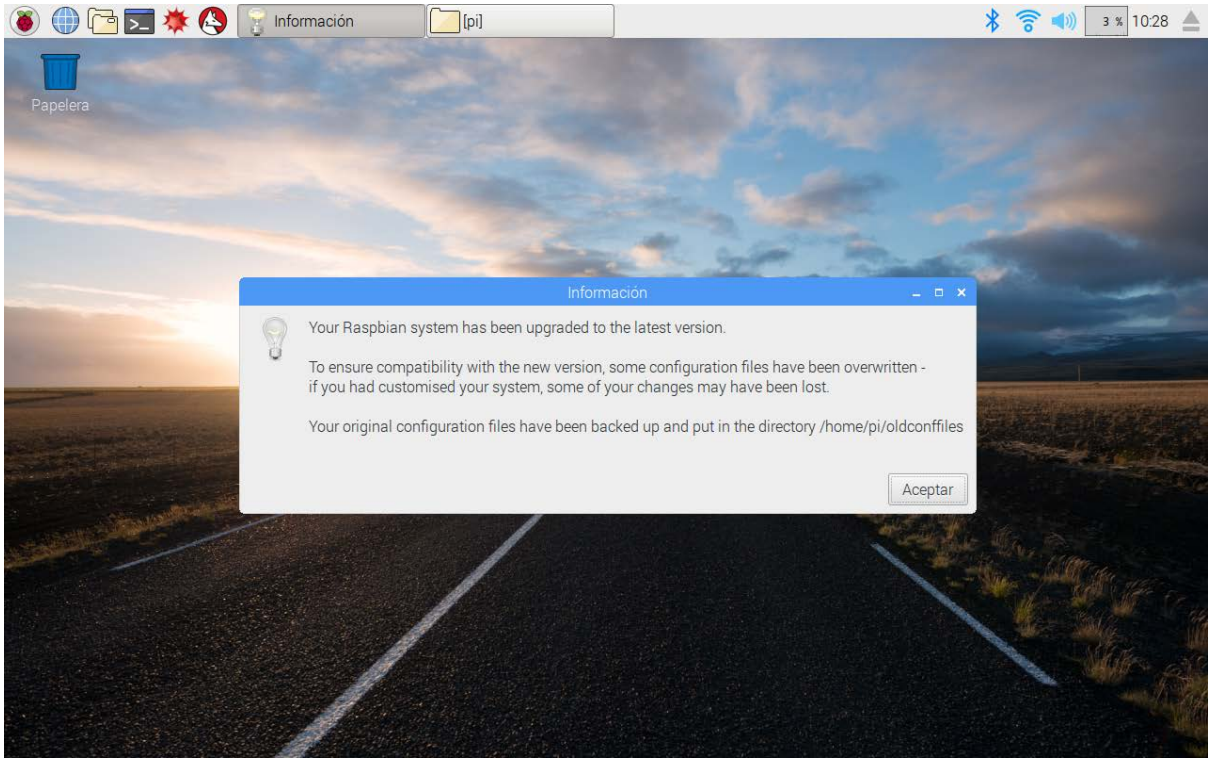
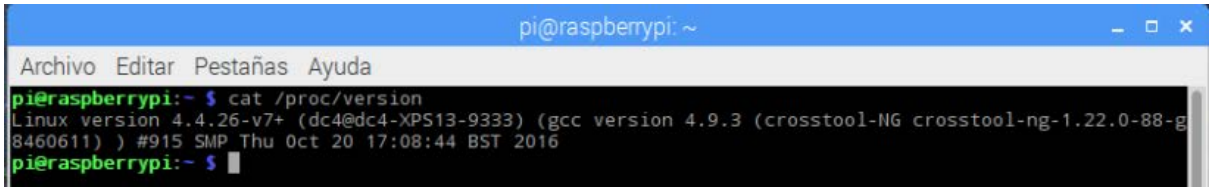


Fig. 20: Raspbian automatic update result.

Keyboard language and VNC server had to be reconfigured, but WiFi settings remained.
WiFi IP address is the same as Ethernet IP, plus one.
To find out differences after the upgrade, we executed command "*cat /proc/version":*



No changes are shown.

3.2.8 Setting static IP issue

In order to arrange a static IP for the rpi, we have tried to define into router settings both WiFi and ethernet addresses. WiFi IP address is 192.168.0.199, and ethernet 192.168.0.198. It is mandatory for these addresses to be into router DHCP range. You can set these values into router webpage, in this case "Settings / LAN" (Fig. 21):

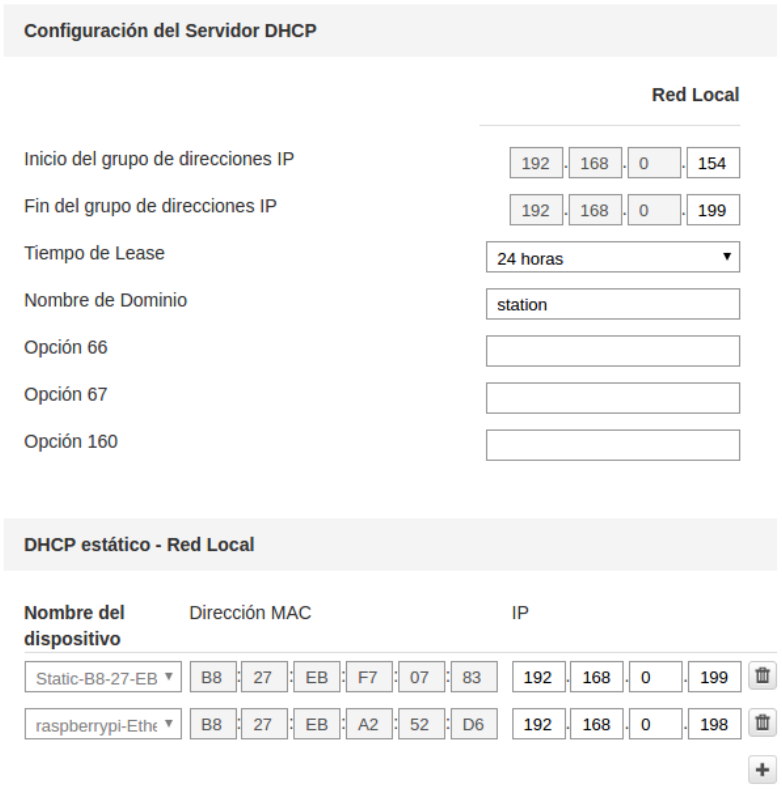


Fig. 21: Configuring rpi static IP into router web page.

In Raspbian, WiFi settings are stored into "***/etc/wpa_supplicant/wpa_supplicant.conf***" file:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=ES
network={
        ssid="vodafone46A1"
        psk="xxxx"
        key_mgmt=WPA-PSK
        disabled=1
}
```

If we connect to the wifi network generated by an Android mobile phone brand Cubot ("Tethering" process), the contents of the ".conf" file will change like this (WiFi passwords hidden for security reasons):

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=ES
network={
        ssid="vodafone46A1"
        psk="xxxxx"
        key_mgmt=WPA-PSK
        disabled=1
}
network={
        ssid="CubotX17CJ"
        psk="xxxxx"
        key_mgmt=WPA-PSK
}
```

At startup, the issue will rise: rpi did not connect to router (SSID "Vodafone46A1"). It did not appear "Associated with Vodafone46A1" as in Fig. 22 (an example of correct connection). If we check rpi IP address, **169.254.149.192** was shown**.** So, setting a static IP for rpi on router was not working. After changing the settings to DHCP, everything was back to normal.

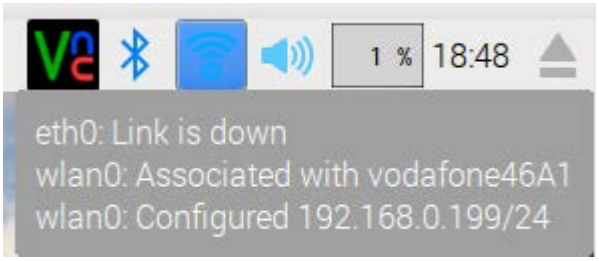


Fig. 22: WiFi state connection in Raspbian.

3.3 Remote control of Raspberry Pi from the pc using RealVNC

Now we are going to configure a program called VNC in order to remotely control the rpi, therefore there will be no need to connect a monitor, keyboard, etc.

Instructions provided here:
http://frambuesa-pi.blogspot.com.es/2016/10/raspberry-pi-with-pixel-control-remoto.html

First we need to activate RealVNC server in Raspbian. As shown in Fig. 23, we have to press raspberry icon in the upper left corner, then "Preferences" -> "Raspberry Pi Configuration", "Interfaces" tab, and activate "VNC". Then we press "OK" and VNC icon will show up (3).

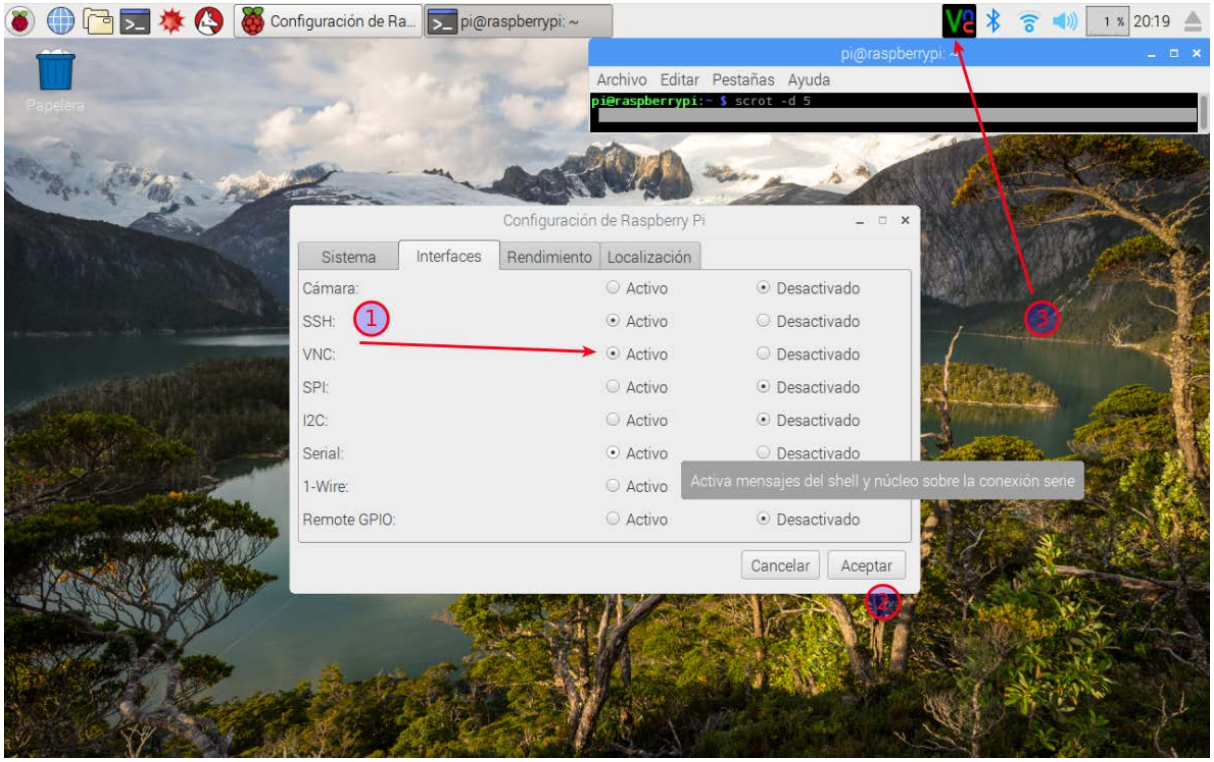


Fig. 23: Activating VNC remote control in Raspbian.

VNC server will be automatically loaded in every reboot.

Now we need to install RealVnc Viewer in the computer from which we want to control the Raspberry. The program can be downloaded from https://www.realvnc.com/download/viewer/
In my computer, "VNC-Viewer-6.0.0-Linux-x64" was the correct file, because my Linux system is 64bits.
After some testing, it was clear that VNCviewer version 6 was not working, so an older version was downloaded (version 5) from:
https://www.realvnc.com/download/vnc/linux/
At the bottom of the page appears "Can I still download VNC 5.x?", press "Show the table" (Fig. 24) and you can choose proper version for your system. I chose https://www.realvnc.com/download/file/vnc.files/VNC-5.3.2-Linux-x64-DEB.tar.gz

Can I still download VNC 5.x?

Yes. If you have one of our auxiliary platforms, this may be necessary. Show the table.

How do I connect?

Start with the FAQ.

Version	Software download						Policy template files	VNC Permissions Creator	
5.3.2	DEB x86	DEB x64	RPM x86	RPM x64	Generic x86	Generic x64	Download	x86	x64
5.3.1	DEB x86	DEB x64	RPM x86	RPM x64	Generic x86	Generic x64	Download	x86	x64
5.3.0	DEB x86	DEB x64	RPM x86	RPM x64	Generic x86	Generic x64	Download	x86	x64
5.2.3	DEB x86	DEB x64	RPM x86	RPM x64	Generic x86	Generic x64	Download	x86	x64
5.2.2	DEB x86	DEB x64	RPM x86	RPM x64	Generic x86	Generic x64	Download	x86	x64
5.2.1	DEB x86	DEB x64	RPM x86	RPM x64	Generic x86	Generic x64	Download	x86	x64

Fig. 24: VNC remote control program download web page.

For Windows systems, version 5 is available from:
https://www.realvnc.com/download/file/vnc.files/VNC-5.3.2-Windows.exe

NOTE: This version works well, without connecting keyboard nor hdmi cable.
Installing package "VNC-Viewer-5.3.2-Linux-x64.deb" version 5.3.2.19179 some warning messages will appear (Fig. 25).

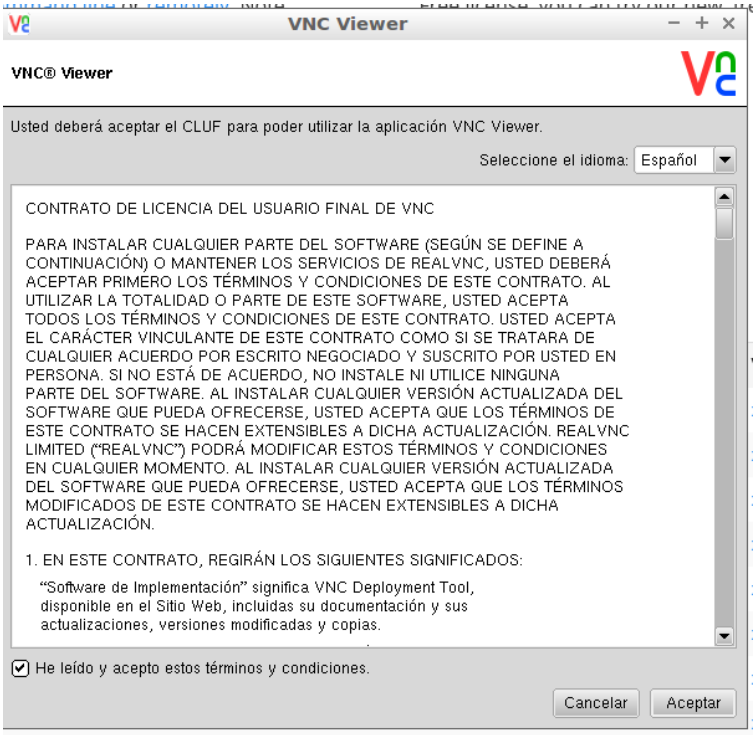


Fig. 25: VNC Viewer license agreement screenshot.

When we execute the program, VNC Viewer main screen shows up (Fig. 26), we need to put rpi IP address (MAC address is B8:27:EB:A2:52:D6, IP address 192.168.0.159)

Fig. 26: VNC Viewer connection screen.

Then click "Connect".
Connection is established OK, unlike version 6, which did not work at all.

[Issue]: Resolution of VNC Viewer in this mode is very low, 600x400 pixels. Thus not all desktop icons are shown and a lot of desktop space is not reachable.

To solve this, we need to modify "*/boot/config.txt"* file, so we will choose a more appropriate graphical mode resolution for this "headless" mode (booting the rpi without a monitor).
Info was extracted from: https://www.raspberrypi.org/documentation/configuration/config-txt.md

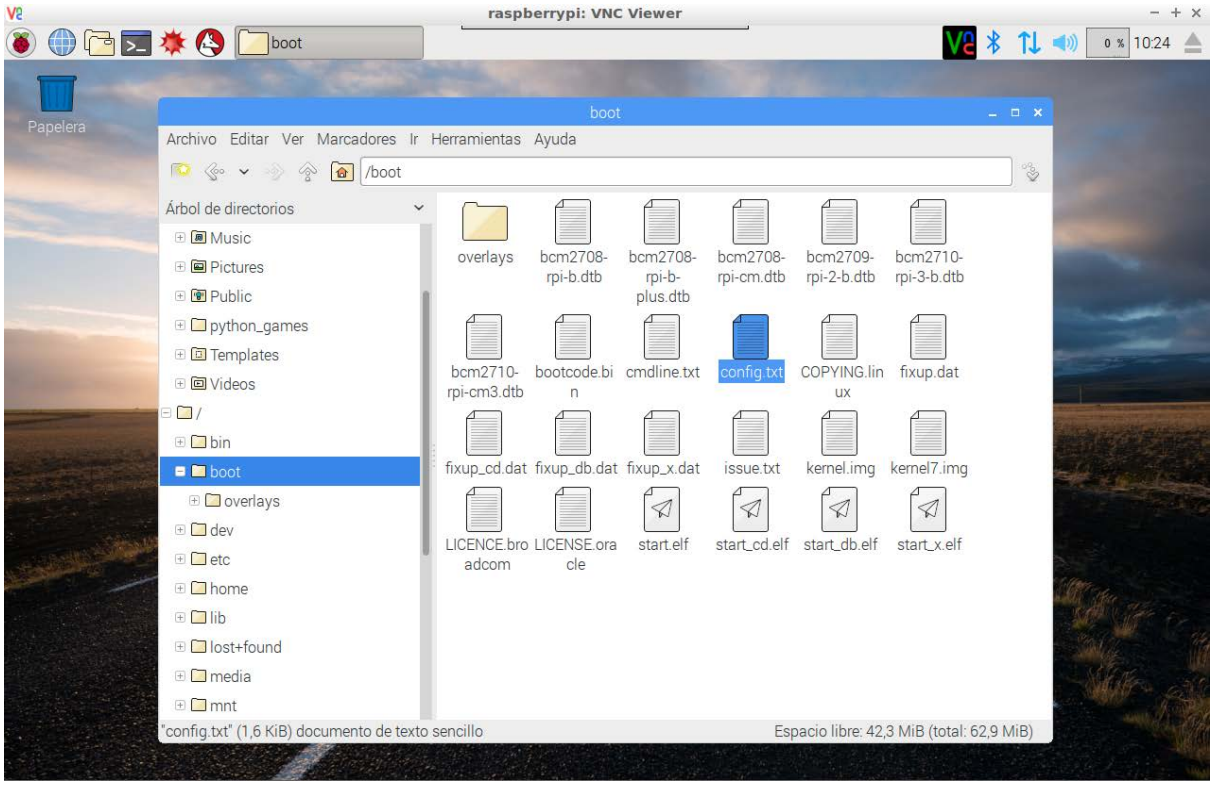


Fig. 27: Config.txt file location en Raspbian.

From a terminal windows, we execute this command:

```
sudo nano /boot/config.txt
```

We need to uncomment (see Fig. 28) "*hdmi_force_hotplug=1"* line, and also set parameters "*hdmi_group"* to 2 and "*hdmi_mode"* to 28.
hdmi_group=2 sets HDMI output to DMT mode (monitors) and *hdmi_mode=28* sets screen resolution to 1280x800 pixels at 60Hz.

```
hdmi_force_hotplug=1
hdmi_group=2
hdmi_mode=28
```

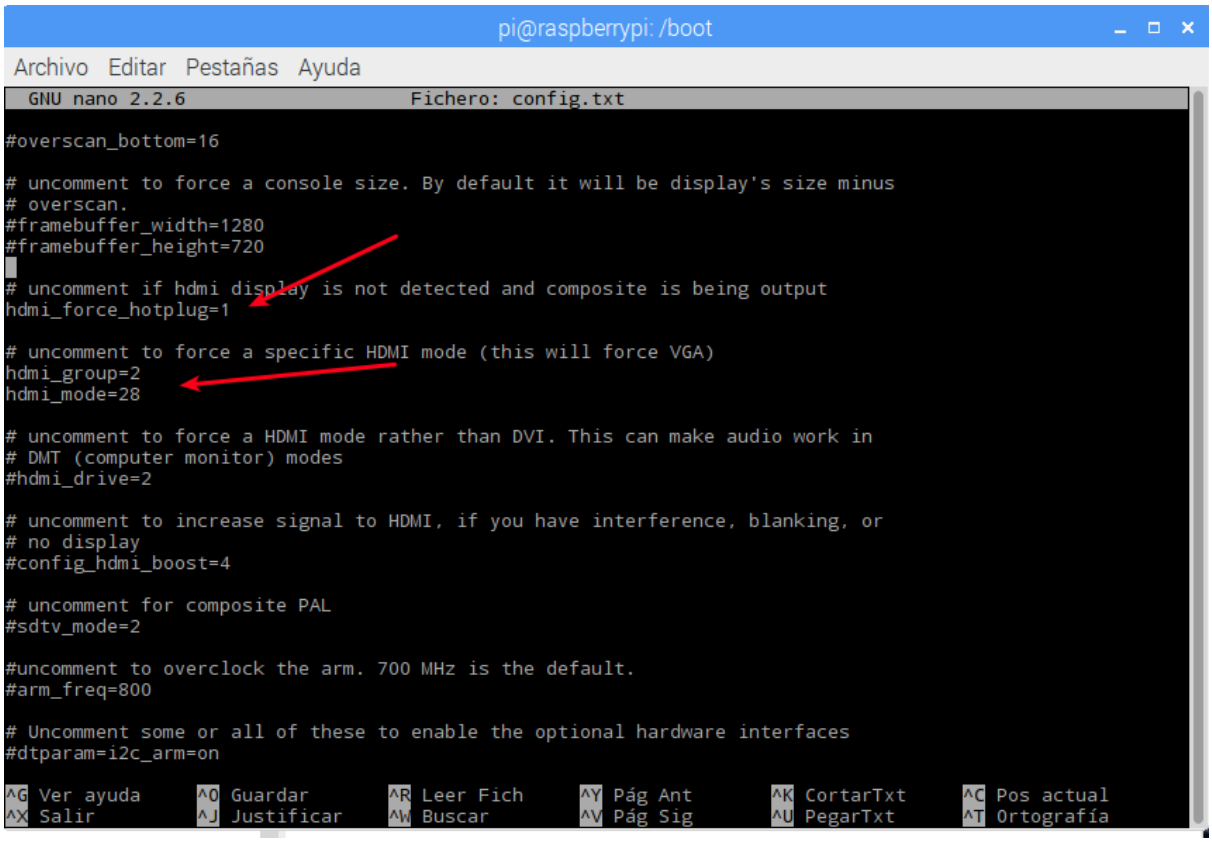


Fig. 28: Modified file "config.txt" for using rpi in headless mode.

Save the file by pressing Ctrl + O and then Ctrl + X, next execute *"shutdown"* of the rpi, and you can disconnect HDMI cables, mouse, keyboard, etc.

Next time rpi boot up, we will be able to control it from another computer (Fig. 29) using VNC Viewer and using a 1280x800 pixels screen resolution.

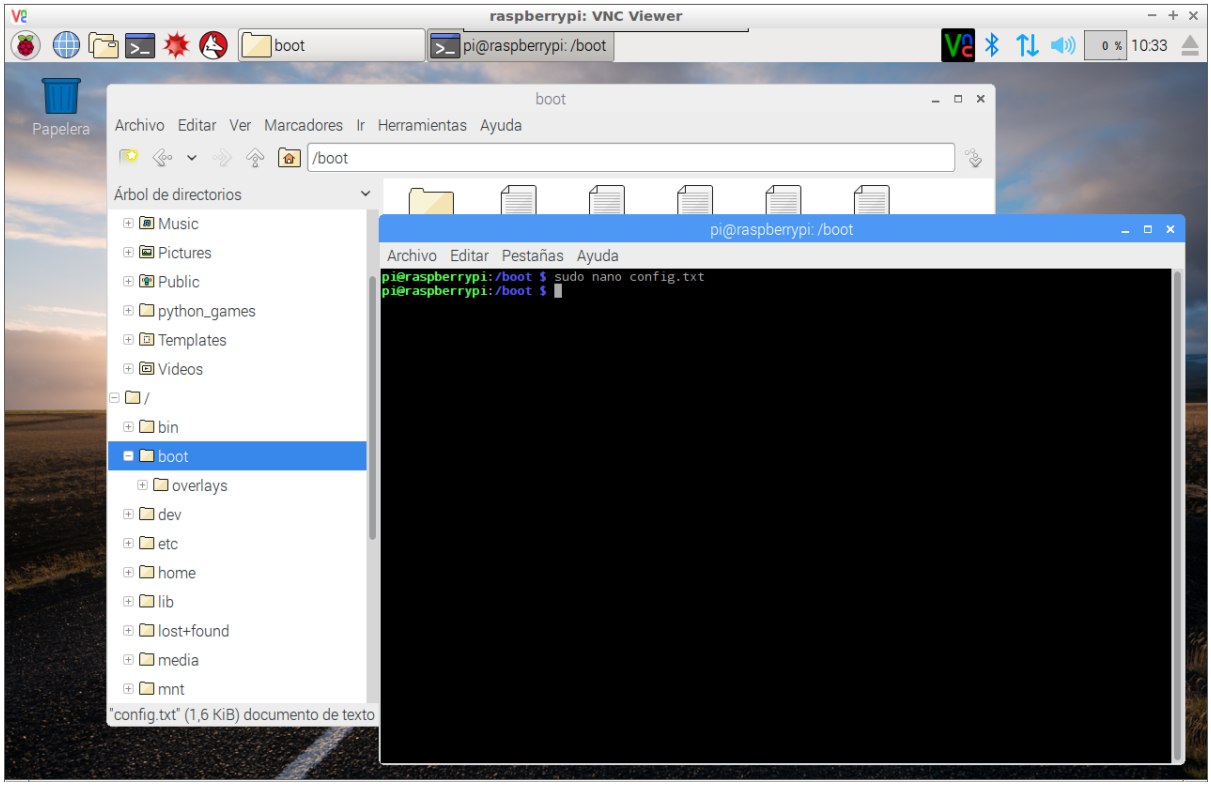


Fig. 29: Remotely controlling rpi in headless mode using VNC Viewer.

3.3.1 Transferring files from/to rpi/pc using RealVNC

To do this (Fig. 30), you need to **right click** Real VNC Server icon in Raspbian. Then we click on (1) "File Transfer", then on (2) "Send Files", select file to transfer (3) click "OK", and a window will show up (4) indicating that file transfer has been completed.

Transferred files are located in the desktop of target pc, although this behavior can be modified.

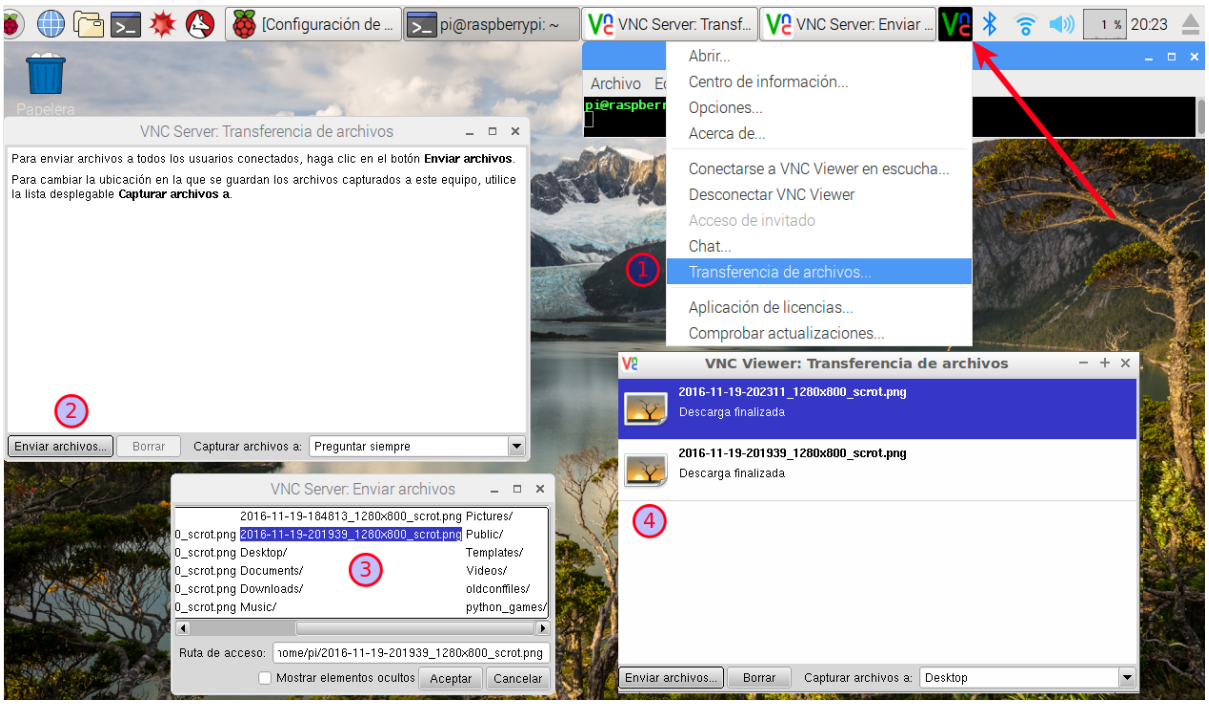


Fig. 30: Transferring files between a PC and rpi using RealVNC.

3.4 Connecting FTDI cable to Raspberry Pi

Previously, we have shown how to connect a computer to our rpi using SSH, but there are also other options. One of them is to connect the rpi to the computer using a **USB to RS232 (FTDI) adapter cable**, which will provide us with a serial interface that we will use to access Linux operating System console.
This way, we will be able to see Kernel messages during OS booting process, access OS prompt, etc.
Fig. 31 shows how to connect all the elements. Connect USB cable to computer, and the other end (three connectors, because the forth is not needed) to rpi GPIO port connector (Fig. 33).

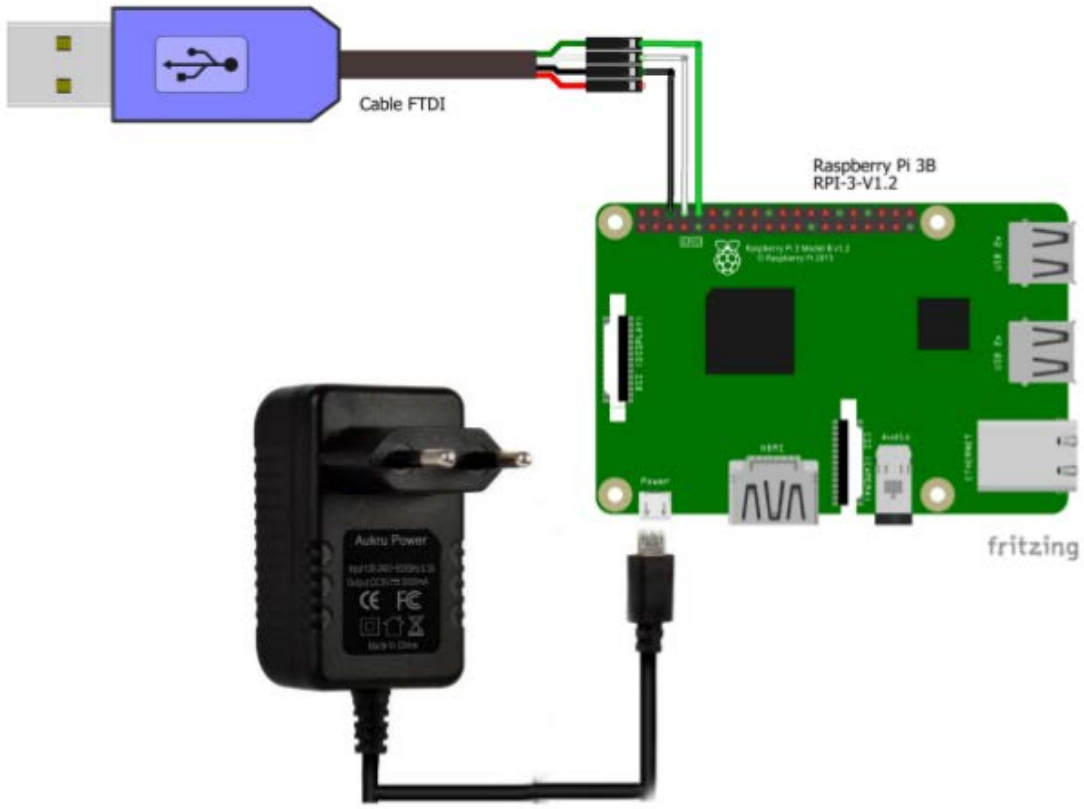


Fig. 31: Raspberry Pi 3 connection scheme using FTDI cable.

3.4.1 Elements needed

We are going to use a "FTDI USB to serial TTL console/debug cable for Raspberry Pi" (Fig. 32), purchased at: https://shop.technofix.uk/ftdi-usb-to-serial-ttl-console-debug-cable-for-raspberry-pi?search=ftdi

Fig. 32: FTDI cable for Raspberry Pi.

Installation guides and driver download links are available at the following web addresses:
http://www.ftdichip.com/Drivers/VCP.htm
http://www.ftdichip.com/Support/Documents/InstallGuides.htm

Some good information about how to use these cables at:
https://blog.christophersmart.com/2016/10/27/building-and-booting-upstream-linux-and-u-boot-for-raspberry-pi-23-arm-boards/

Raspberry Pi GPIO pinout (from: http://pinout.xyz/)

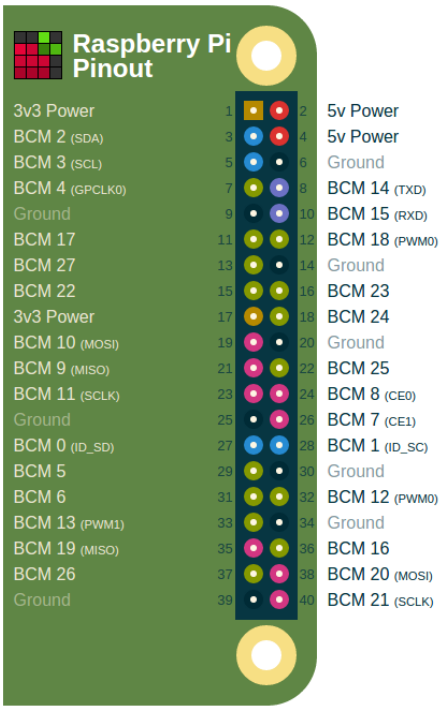


Fig. 33: Raspberry Pi 3 GPIO pinout.

There is also good information about it at: http://elinux.org/RPi_Serial_Connection

3.4.2 Installing Linux

Once the USB cable is connected to the computer (Lubuntu or Ubuntu version), using the following command allow us to find out if the cable has been recognized by the system:

```
dmesg
```

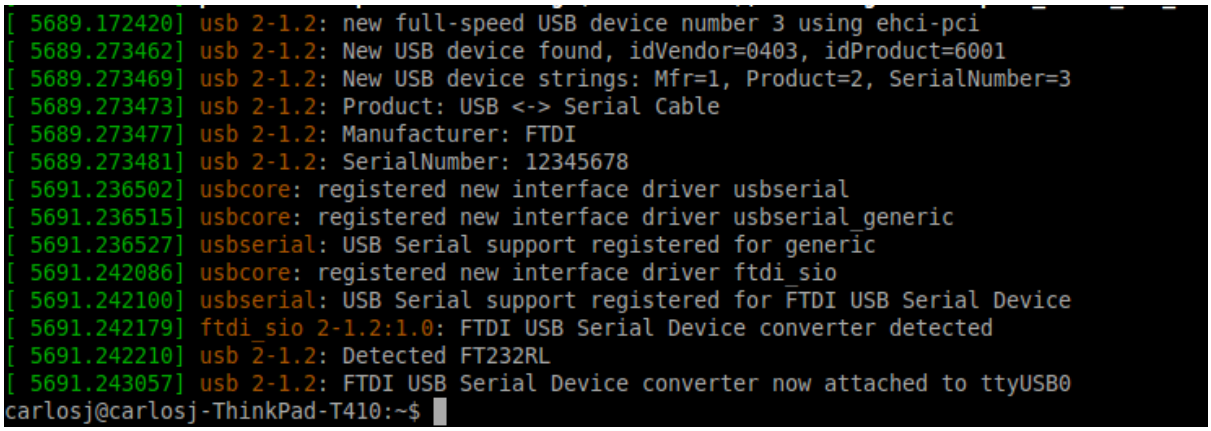


Fig. 34: Dmesg command result after connecting FTDI cable to pc.

[Serial interface identification in Linux]: In Linux serial devices are identified with names /dev/ttyS0, /dev/ttyS1, etc. If we are using a USB-RS232 converter, it will show up as /dev/ttyUSB0, /dev/ttyUSB1, etc.

In this case the FTDI cable is recognized as "*ttyUSB0*" (Fig. 34).

Using FTDI cables with Raspberry Pi 3 has been reported a little tricky, because in this model, Bluetooth and serial port share the same UART.
Those issues related to serial port, UART and Bluetooth, are thoroughly described at (March 2016):
http://www.slideshare.net/yeokm1/raspberry-pi-3-uartbluetooth-issues

However, as we shall see later, we have not suffered from such problems.

Next we are going to install in our computer running Linux, the "screen" program (Fig. 35) in order to connect to the rpi.

```
sudo apt-get install screen
```

```
carlosj@carlosj-ThinkPad-T410:~$ sudo apt-get install screen
[sudo] password for carlosj:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  libgee-0.8-2 liblua5.1-0 linux-headers-4.2.0-16 linux-headers-4.2.0-16-generic linux-headers-4.2.0-34
  linux-headers-4.2.0-34-generic linux-headers-4.2.0-35 linux-headers-4.2.0-35-generic linux-headers-4.2.0-36
  linux-headers-4.2.0-36-generic linux-headers-4.2.0-38 linux-headers-4.2.0-38-generic linux-image-4.2.0-16-generic
  linux-image-4.2.0-34-generic linux-image-4.2.0-35-generic linux-image-4.2.0-36-generic linux-image-4.2.0-38-generic
  linux-image-extra-4.2.0-16-generic linux-image-extra-4.2.0-34-generic linux-image-extra-4.2.0-35-generic
  linux-image-extra-4.2.0-36-generic linux-image-extra-4.2.0-38-generic linux-signed-image-4.2.0-34-generic realpath
Utilice «apt-get autoremove» para eliminarlos.
Paquetes sugeridos:
  iselect screenie byobu ncurses-term
Se instalarán los siguientes paquetes NUEVOS:
  screen
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 2 no actualizados.
Se necesita descargar 567 kB de archivos.
Se utilizarán 971 kB de espacio de disco adicional después de esta operación.
Des:1 http://es.archive.ubuntu.com/ubuntu/ wily/main screen amd64 4.3.1-2 [567 kB]
Descargados 567 kB en 0s (1.090 kB/s)
Seleccionando el paquete screen previamente no seleccionado.
(Leyendo la base de datos ... 354891 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../screen_4.3.1-2_amd64.deb ...
Desempaquetando screen (4.3.1-2) ...
Procesando disparadores para install-info (6.0.0.dfsg.1-3) ...
Procesando disparadores para man-db (2.7.4-1) ...
Procesando disparadores para systemd (225-1ubuntu9.1) ...
Procesando disparadores para ureadahead (0.100.0-19) ...
ureadahead will be reprofiled on next reboot
Configurando screen (4.3.1-2) ...
Procesando disparadores para systemd (225-1ubuntu9.1) ...
Procesando disparadores para ureadahead (0.100.0-19) ...
carlosj@carlosj-ThinkPad-T410:~$
```

Fig. 35: Installing Screen program in Linux.

Without connecting FTDI cable to the computer, and with the Raspberry Pi turned off, we have to connect the cables as shown in Fig. 36, to 6, 8 and 10 pins of rpi GPIO connector:

Black: GND.
White: RXD, connects to rpi TXD.
Green: TXD, connects to rpi RXD.
Red: Vcc (5 volts), we are not going to use it because we are going to power the rpi through its microUSB connector with an external charger.

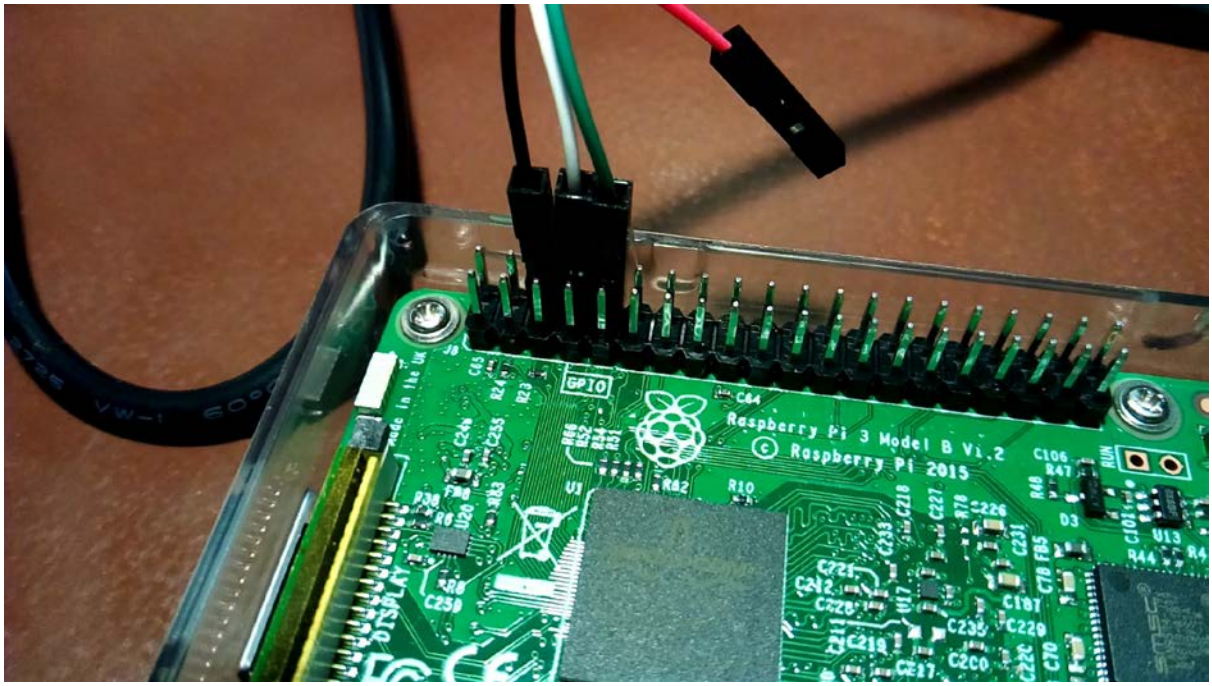


Fig. 36: FTDI cable connections at rpi GPIO interface.

Connect the USB cable to your computer and check with "dmesg" command that it has been recognized as ttyUSB0:

```
13324.615885] ftdi_sio 2-1.2:1.0: FTDI USB Serial Device converter detected
13324.615952] usb 2-1.2: Detected FT232RL
13324.616666] usb 2-1.2: FTDI USB Serial Device converter now attached to ttyUSB0
```

Fig. 37: Finding out how Linux recognizes FTDI cable.

Now we execute the following command:

```
sudo screen /dev/ttyUSB0 115200
```

If we boot the rpi up, we should see kernel messages, but nothing appears.

We run again "*sudo screen /dev/ttyUSB0 115200*", but to no avail, still nothing shows up.

SOLUTION: We need to go to Raspberry Pi Configuration in Raspbian, "**Interfaces**" tab, activate serial port option ("**Serial**") and click "OK" (Fig. 38).

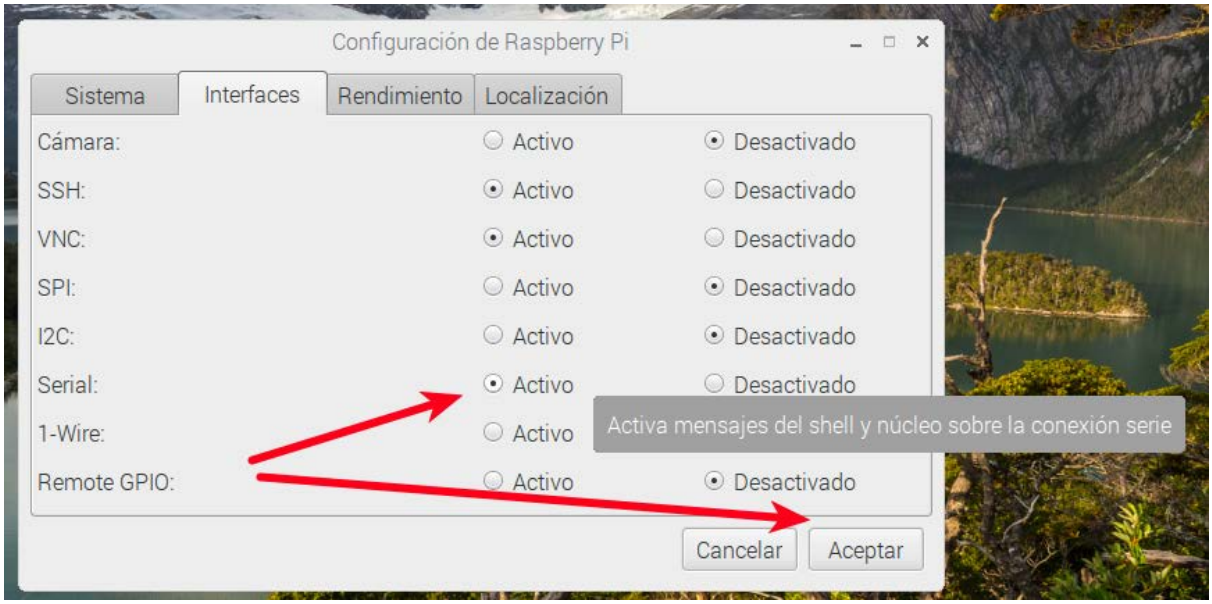
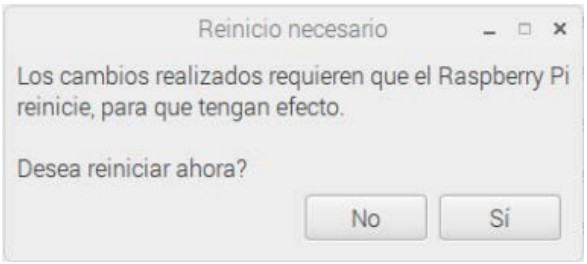


Fig. 38: Activating serial port support in Raspbian.

A restart is needed to apply the changes.



Another option to activate serial port support in Raspbian is to modify "*/boot/**config.txt***" file, adding a line: "*enable_uart=1*"

The same result can be achieved using the console instead of the graphical interface, through:

```
sudo rpi-config
```

After restarting the rpi, we see (Fig. 39) in "*screen*" console that a login is required:

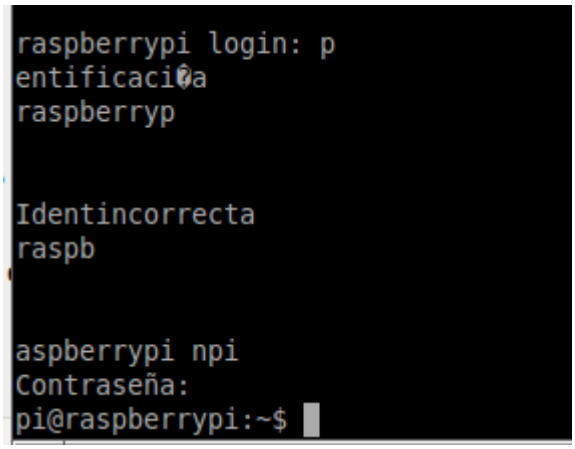


Fig. 39: rpi login using Screen.

3.4.3 Errors detected

Fig. 39 shows an error: characters are not displayed in same row, Enter must be pressed on several occasions for the lines to be displayed, and some of the lines overlap, even if user and password ("pi" and "Raspberry") are accepted.

Typing "*Exit*" login will be asked again.

We install "Putty" program to find out if it is a "screen" program fault.

```
sudo apt-get install putty
```

Execute it with this command:

```
sudo putty
```

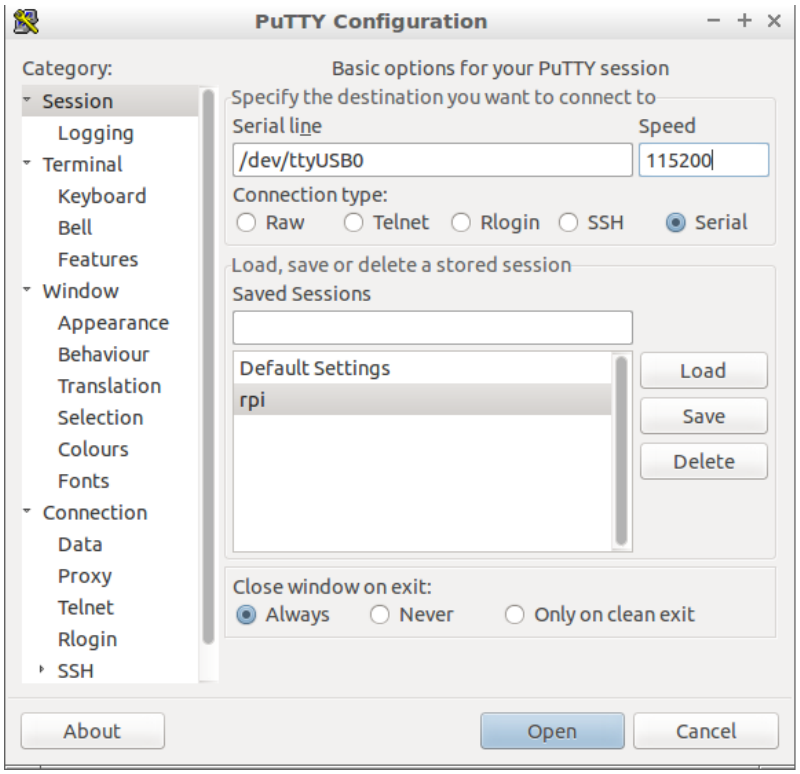


Fig. 40: Putty program.

Configure Putty by pressing "**Serial**" connection type, Serial Line "**/dev/ttyUSB0"**, Speed **"115200"**, and use values shown in Fig. 41 in left column category "Serial".

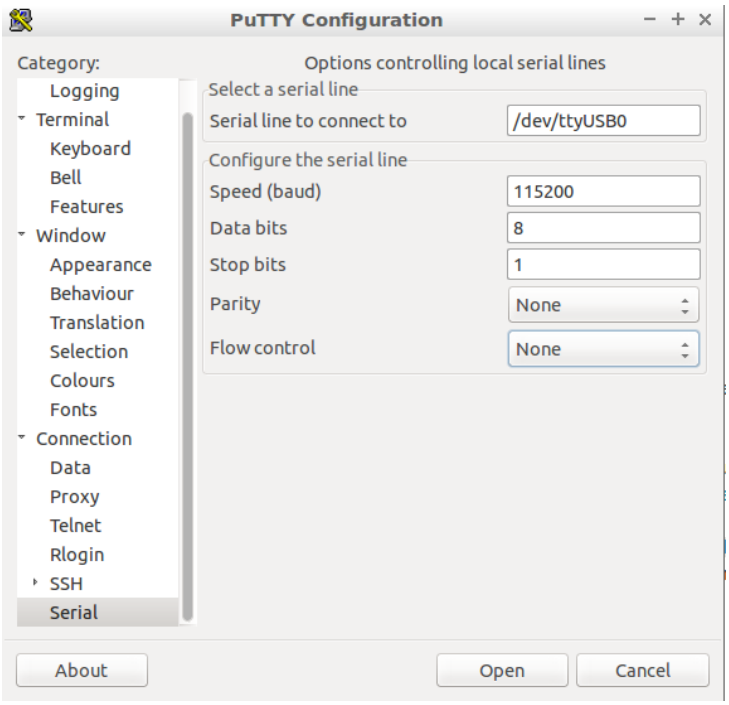


Fig. 41: Putty program configuration parameters for rpi.

Result is the same (Fig. 42), lines overlap and crushed with each other, but login is still possible by typing user and password.

Fig. 42: Connection result using Putty.

In order to test the connection, cables are rearranged, but without success.

3.4.4 Installing FTDI cable in Windows

To rule out a Linux related issue, we are going to test the FTDI cable in Windows OS.
We need to download drivers from:
http://www.ftdichip.com/Drivers/VCP.htm
And unzip file into a folder.

When cable is connected to the pc, a ***USB serial converter*** device will be installed, but we will need to browse manually the exact location where the drivers have been saved to. Repeat same process with a ***usb serial port*** device.
Fig. 43 shows the device ready for use (COM9) in Windows Device Manager.

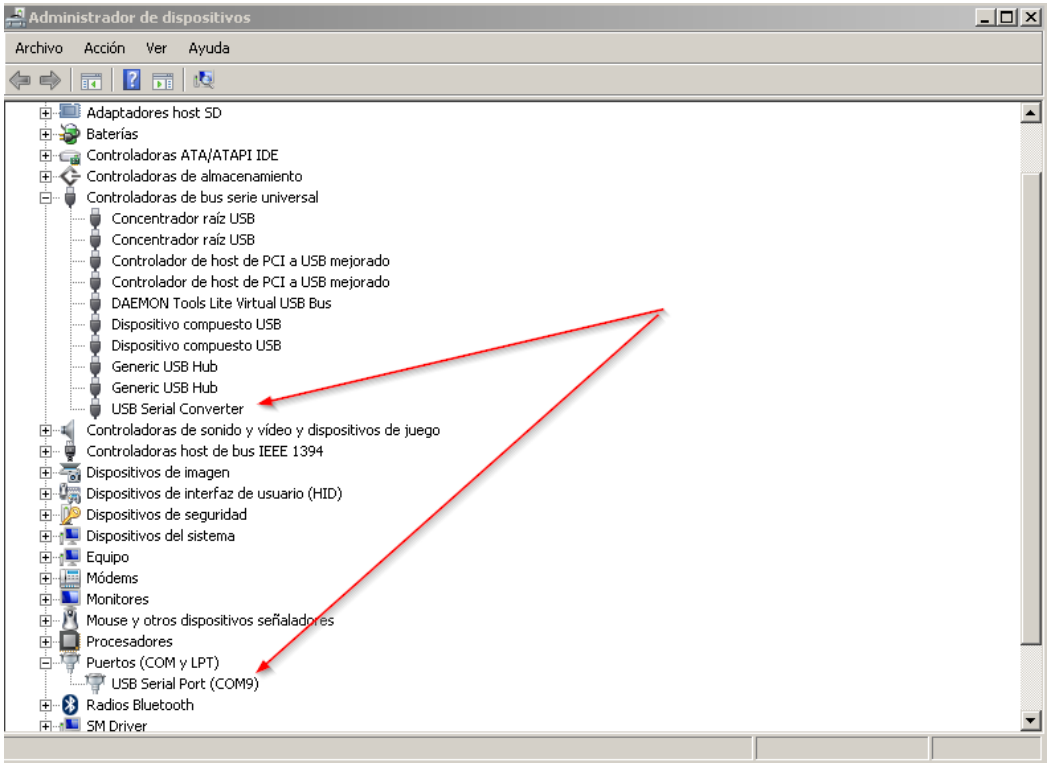


Fig. 43: FTDI cable entry in Windows Device Manager.

Download and install **Putty**, and set it up with the values shown in Fig. 44 y Fig. 45.

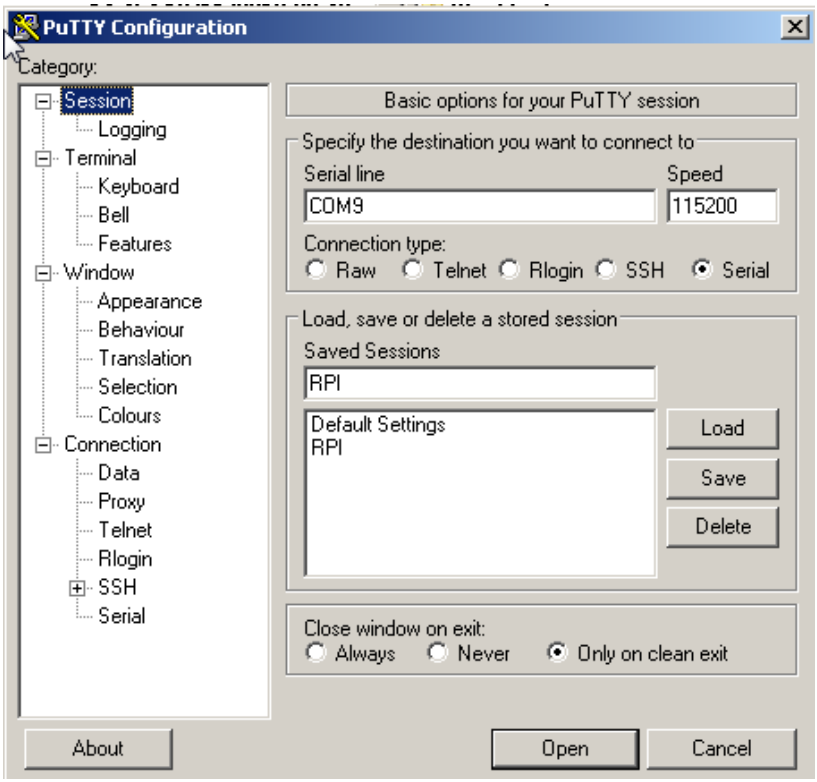


Fig. 44: Putty program in Windows OS.

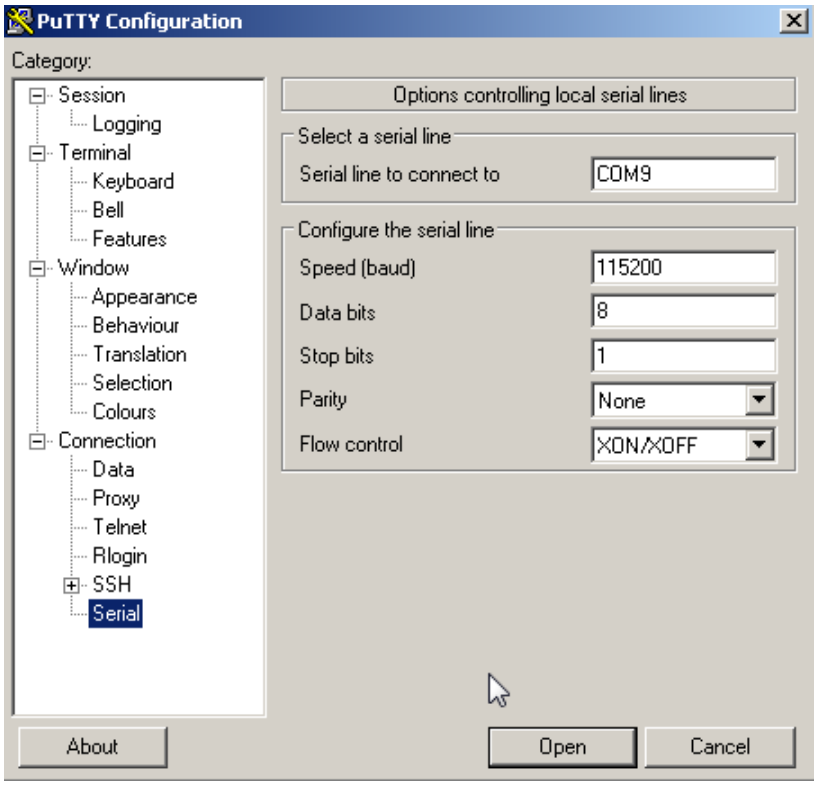


Fig. 45: Configuring Putty program in Windows OS.

Click “Open”, and connection will be established, asking for username and password (Fig. 46).

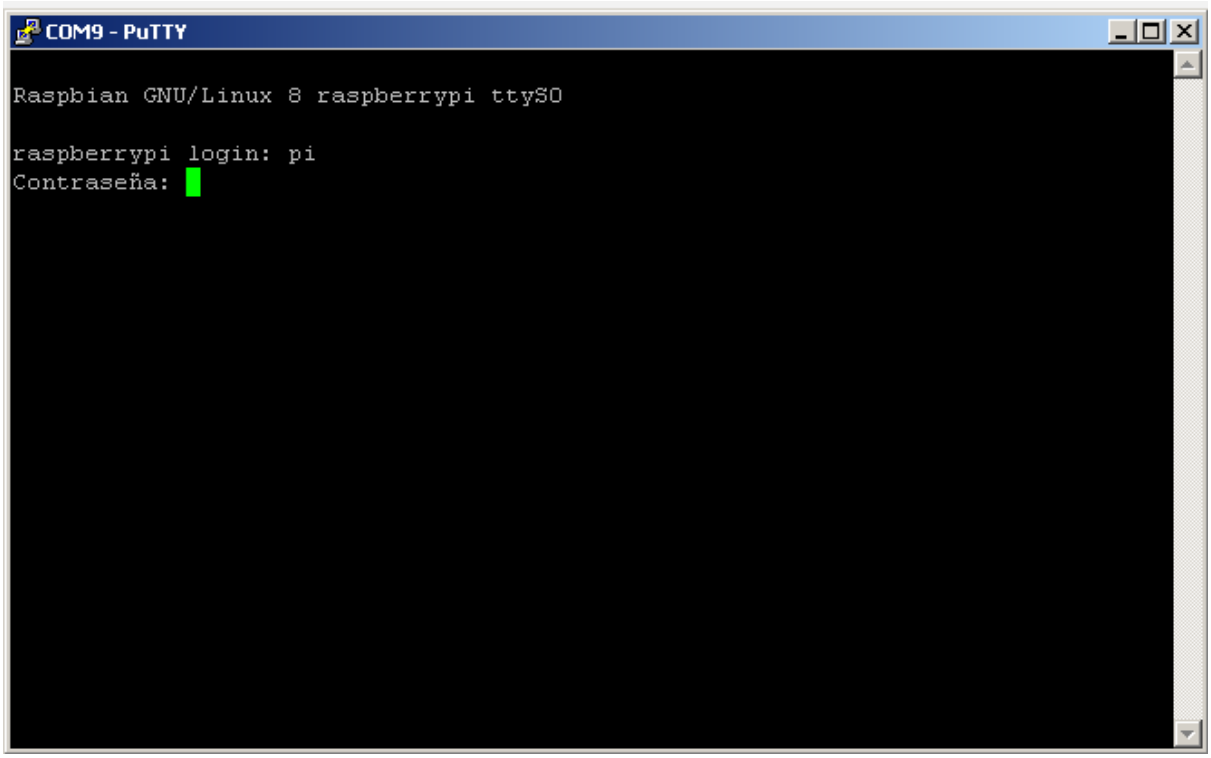


Fig. 46: Putty window showing successful connection to rpi.

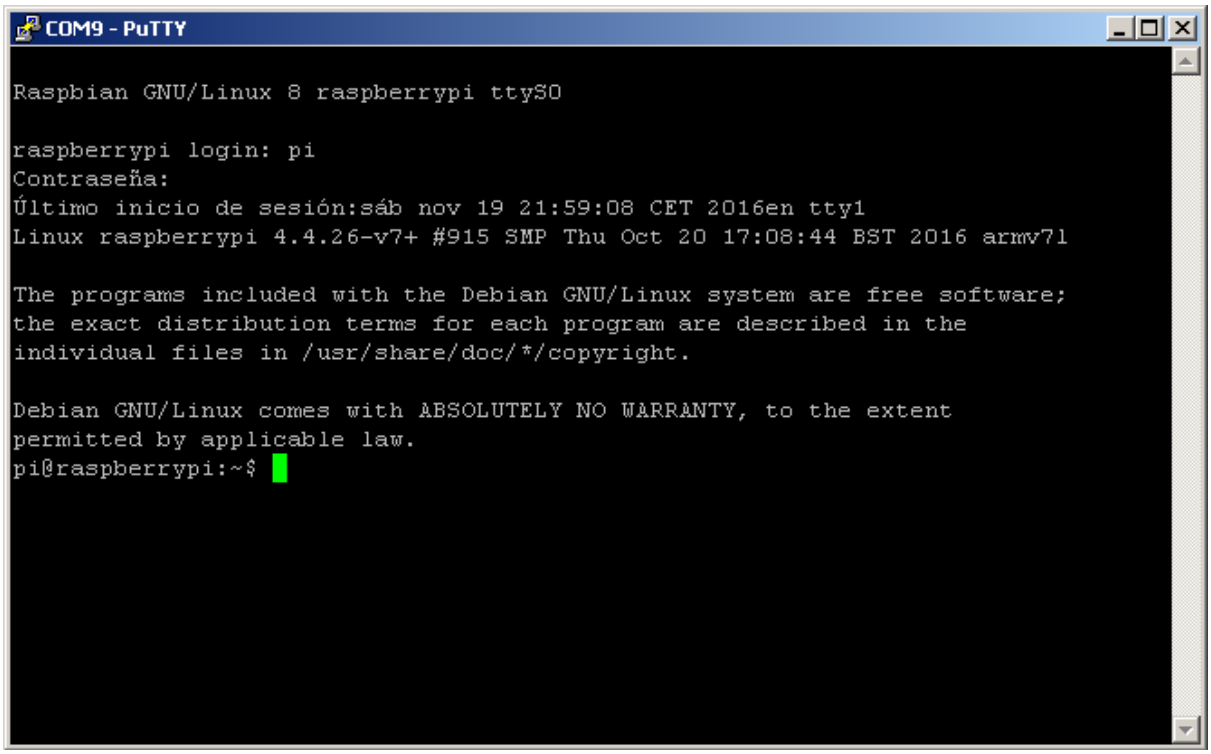


Fig. 47: Raspbian prompt after successful login.

Logout by typing “exit”:

Fig. 48: Raspbian logout in Putty.

The lines are displayed correctly, they do not overlap as it happened in Linux.

3.4.5 Troubleshooting

For troubleshooting purposes, Ubuntu 14.04.5 32bits version was installed into another pc, but without updating it.

After installing Putty, it worked with both flow control types (with or without XON/XOFF):

```
sudo putty
```

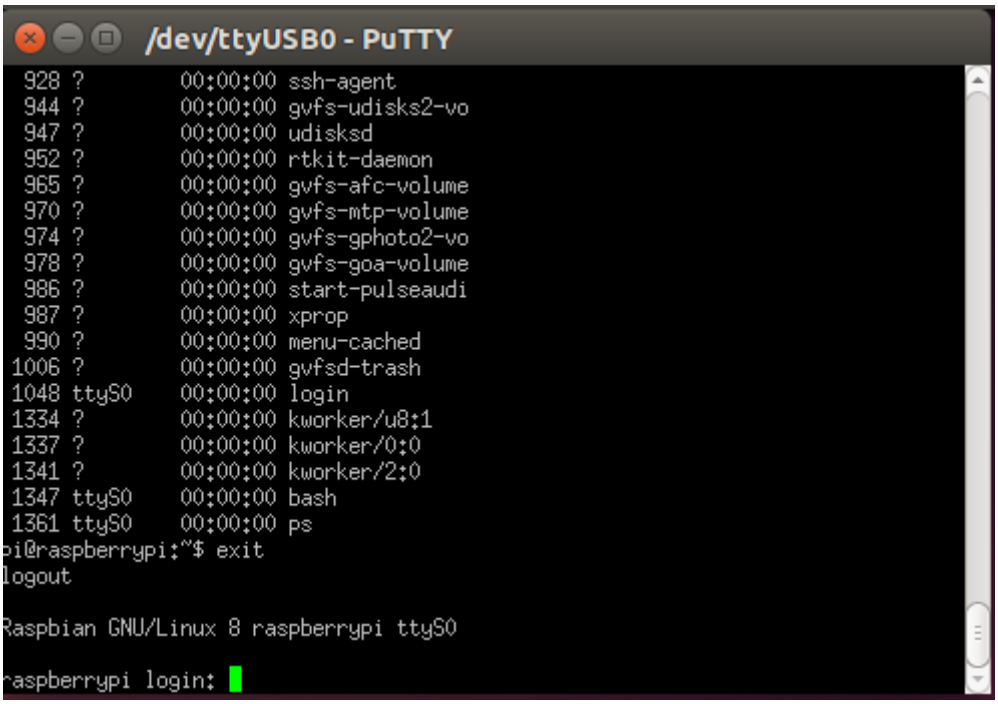


Fig. 49: Raspbian login in Linux using Putty.

After installing **"screen"** program, run it using **"*sudo /dev/ttyUSB0*".** Strange characters are displayed, but login appears to be correct (Fig. 50).

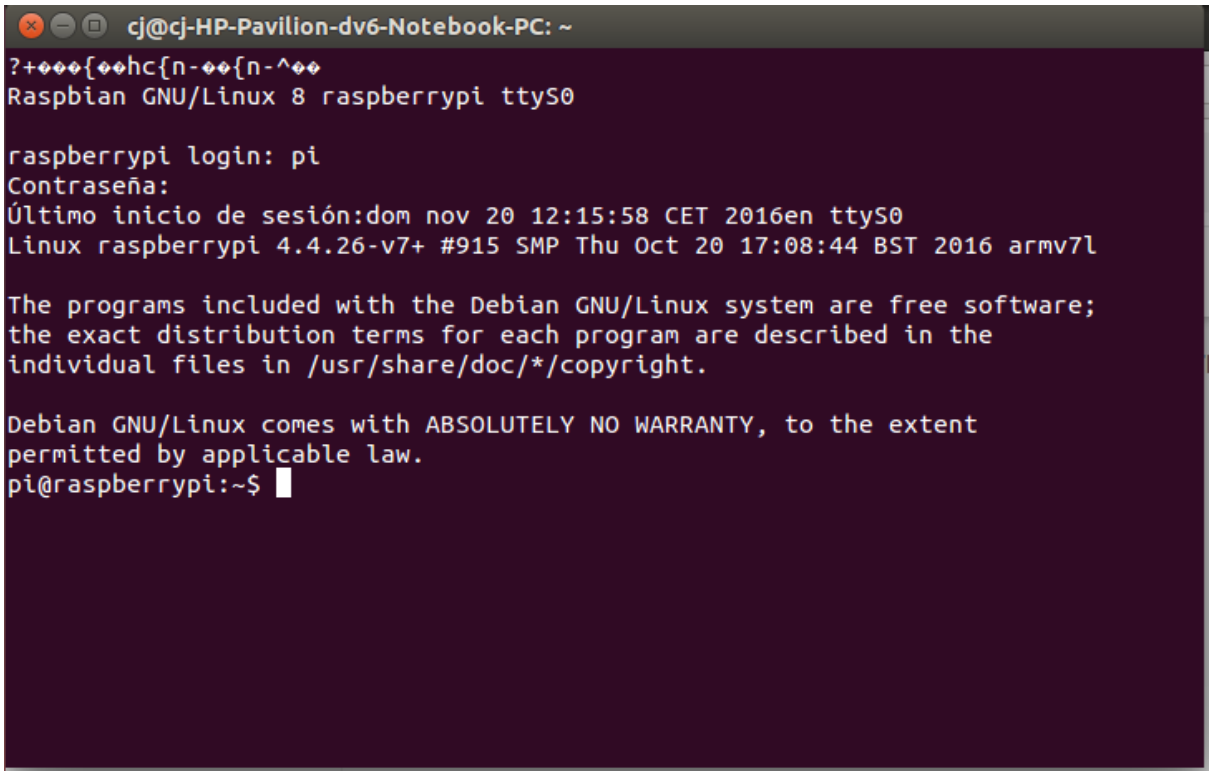


Fig. 50: Raspbian login in Linux using Screen.

Logout by pressing "Ctrl+A" and then type "K" and "Y" to kill the process.

If we check the connection again using Putty or Screen, then complete lines and characters are missing again. The issue is still there.

Uninstall "screen" using the following command:

```
sudo apt-get remove screen
```

Everything is back to normal after rebooting the computer.
Install **screen** program again, run it once, it works fine. Then we run **Putty**, and the issue appears again, characters are missing.
Every time we run **screen**, the issue surface again. Therefore, we are not going to use **screen** anymore, and use **Putty** instead.
After uninstalling **screen** and rebooting computer, everything works fine using Putty.

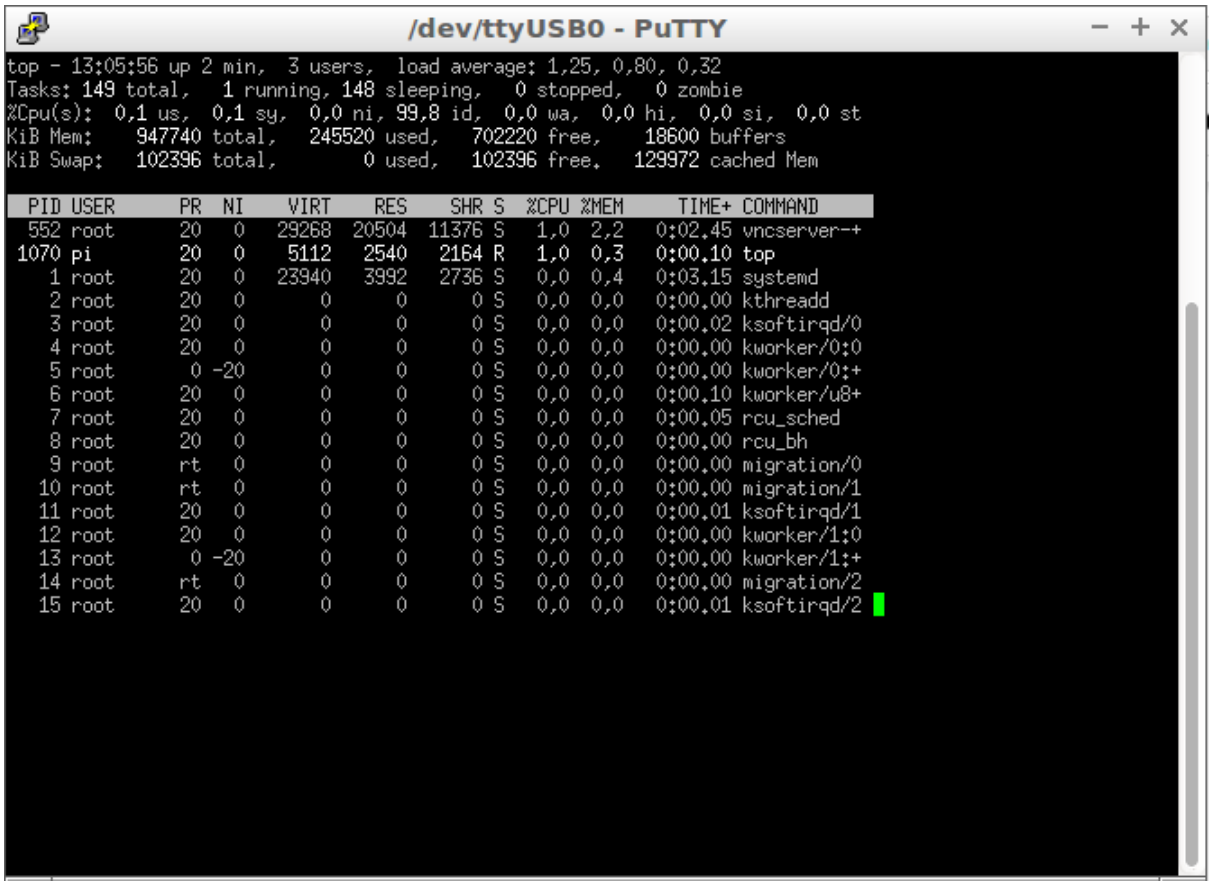


Fig. 51: Command "top" running in Raspbian using Putty.

Fig. 51 shows “top” command running, showing Raspbian system resources.

These are the commands used to shut down or restart the rpi (Fig. 52):

```
sudo shutdown -h now
sudo shutdown -r now
```

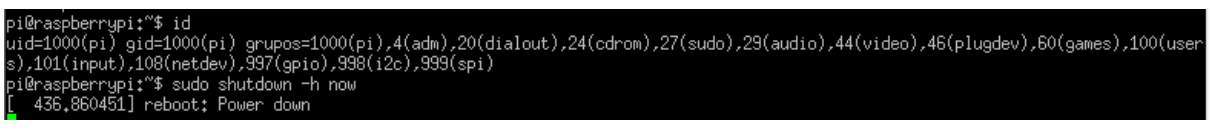


Fig. 52: Shutting down the rpi the right way.

Important: you should not unplug rpi until “Power down” message appears (green light will blink faster, then will stay on and finally will go off, whole process takes about 50 seconds).

Run the following commands to force Ubuntu 14 to update:

```
sudo apt-get update
sudo apt-get upgrade
```

We note that, after update process is complete, Putty connection still works fine. But we notice an error appearing in the background (Fig. 53).

```
(putty:2340): GLib-CRITICAL **: Source ID xxxx was not found when attempting
to remove it
```

Where xxxx represents a number that increments every time we type something in the Putty session.

It is a known bug that was fixed in Ubuntu Utopic (14.10) and Vivid (15.04), according to:
https://bugs.launchpad.net/ubuntu/+source/putty/+bug/1458033

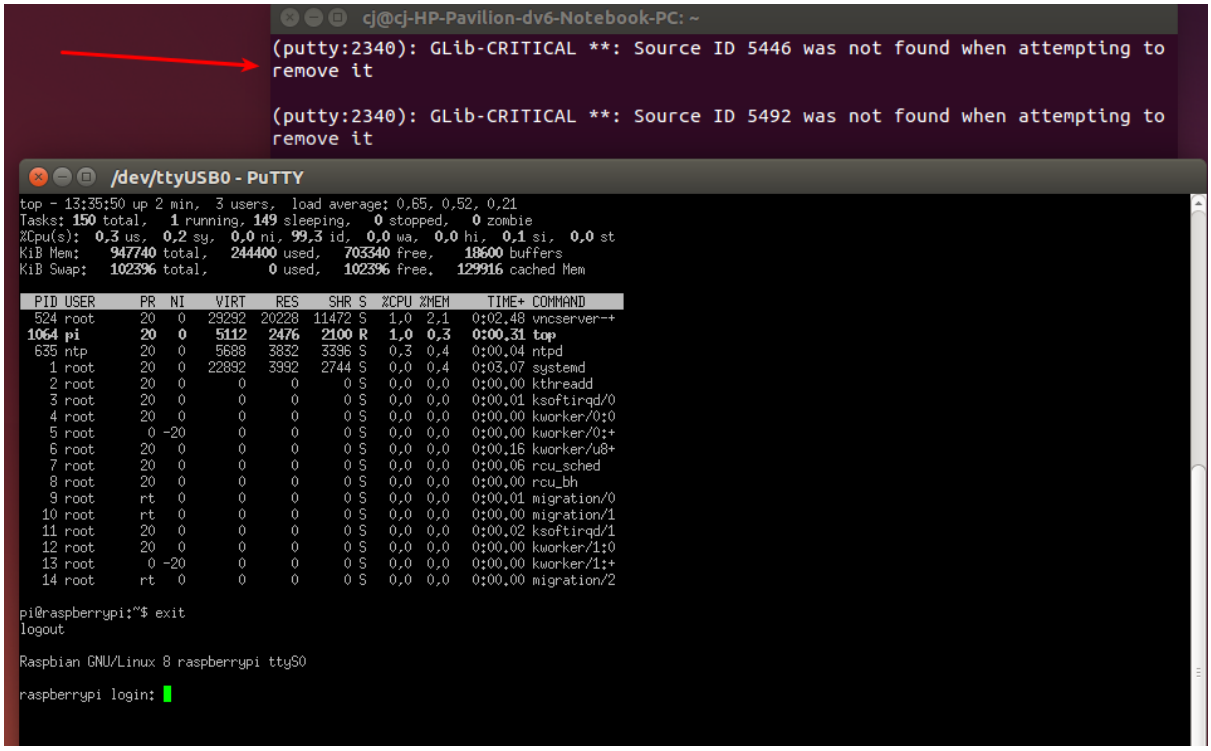


Fig. 53: Bug running Putty in Ubuntu 14.04.

4 INSTALLING AND TESTING EPICS IN RASPBIAN

4.1 Documentation on EPICS

These are the links to documentation on EPICS that I have reviewed:
EPICS main web page: http://www.aps.anl.gov/epics/
EPICS Wiki: https://wiki-ext.aps.anl.gov/epics/index.php/Main_Page
Lectures and presentations on EPICS:
http://www.aps.anl.gov/epics/docs/GSWE.php (2004 y 2005)
http://www.aps.anl.gov/epics/docs/USPAS2010.php (2010)
http://www.aps.anl.gov/epics/docs/USPAS2014.php (2014)
http://www.aps.anl.gov/epics/docs/APS2014.php (2014)
http://www.aps.anl.gov/epics/docs/APS2015.php (2015)
Documents on EPICS base 3.14.12 version, available at:
http://www.aps.anl.gov/epics/base/R3-14/12.php
"EPICS Application developers guide" October 2016, see [RD3]

4.2 Installing EPICS in Raspberry Pi

4.2.1 About EPICS

EPICS (Experimental Physics and Industrial Control System) can be defined in three different ways:

- It is a worldwide collaboration to share designs, software tools and experiences to implement distributed control systems on a large scale.
- It is the architecture of a control system, which uses a client/server model with a very efficient communications protocol (CA - Channel Access) to send/receive data between the different devices that are part of the control system.
- It is a set of software tools, libraries and open source applications, used to create large size control systems, such as particle accelerators, large telescopes, etc.

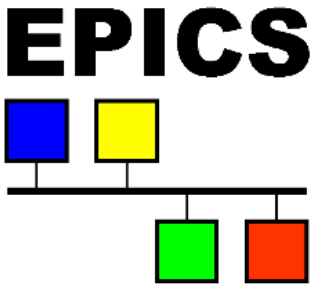


Fig. 54: Experimental Physics and Industrial Control System logo.

Main EPICS components (Fig. 55) are:

- **PVs** (Process Variables): variables are exchanged between clients and servers. These variables are defined in databases that contain different records. For example, the state of a valve in a pumping station.
- **IOCs** (Input Output Controllers): these are the elements that handle these variables. Correspond to the servers in the client/server architecture. They are databases that contain records that represent

the data in the control system. A record consists of a series of attributes (fields) and a code that defines the behavior of the record when it is active.

- **CA** (Channel Access): the backbone of the control network. A communication protocol based on a TCP/IP client/server architecture. It defines how process variables will be transferred between servers and clients.
- **OPIs** (Operator Interface): tools to generate, monitor, archive, or modify the exchanged variables. There are tools to generate a database, such as VDCT, tools (Clients) to monitor variables, such as EDM, MEDM, or CSS.

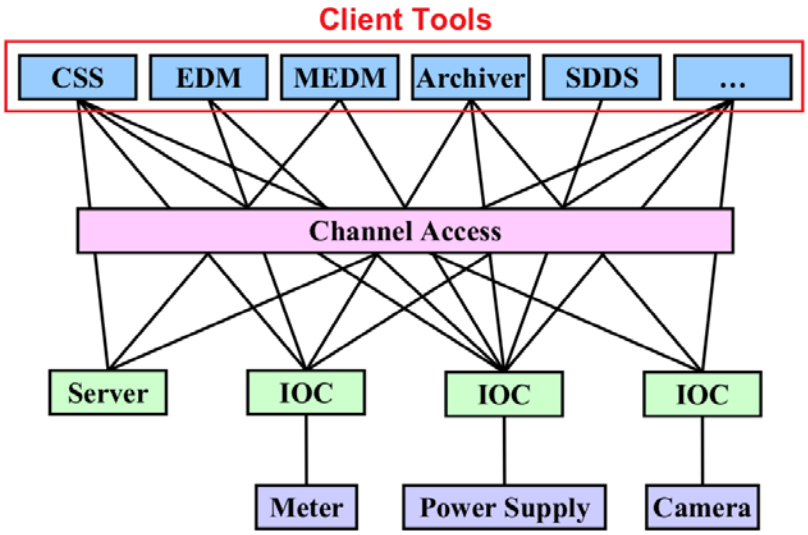


Fig. 55: General overview of EPICS system.

4.2.2 Prior to the installation

EPICS (**"Experimental Physics and Industrial Control System software"**) uses a client/server architecture and to keep things simple in this first approach, we will execute both the server and the client in the rpi.
EPICS server is divided into ***EPICS Base***, which provides development libraries and a few applications and utilities, and ***SynApps***, which provides additional capabilities.
In terms of the client side, given that Raspbian distro incorporates Python, we are going to use ***PyEpics***, to link EPICS with Python language.
As the basis of the installation, we are going to use the same Raspberry Pi 3 v1.2 card. In this card Raspbian OS "Jessie with Pixel" kernel version 4.4 (September 23, 2016) has been installed. A 16 GB microsd card has been used.

We are going to install EPICS in it, following the instructions on the web: https://prjemian.github.io/epicspi/

To do this, we will use USB FTDI cable and a Putty session. Raspberry Pi must be connected to the Internet, to download EPICS installation packages.

Connect USB FTDI cable to the rpi and pc, and establish a Putty session with the connection parameters shown in Fig. 56.

```
sudo putty
```

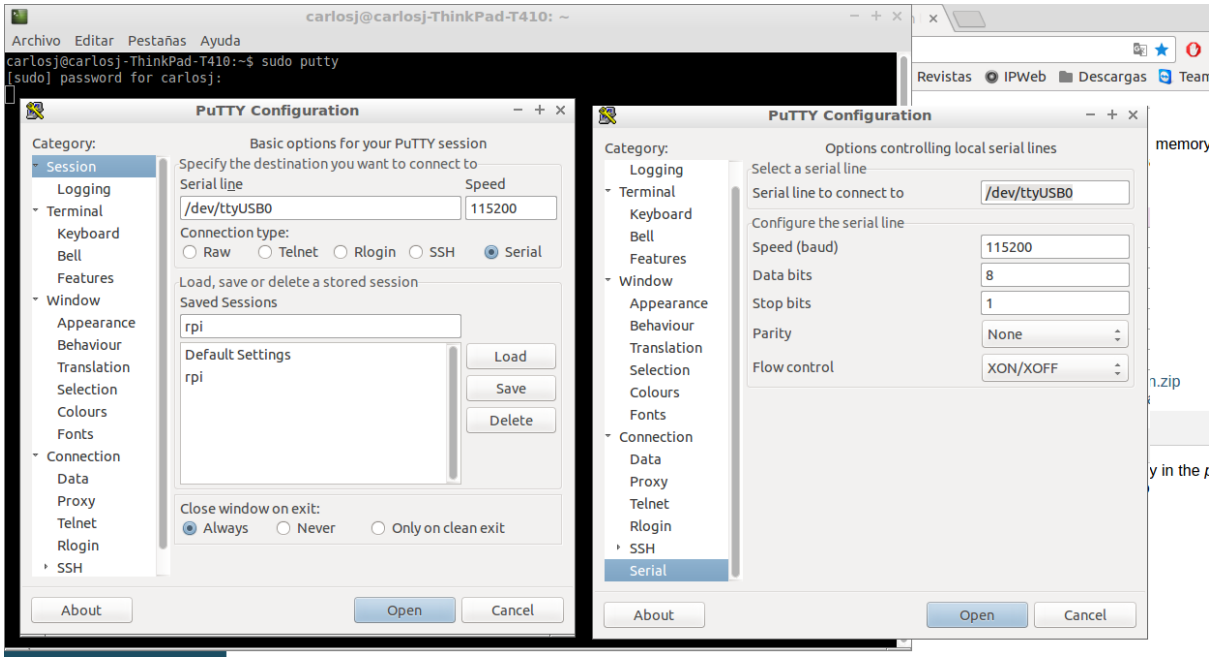


Fig. 56: Putty configuration parameters.

Log in using "**pi**" as user and "**raspberry**" as password.

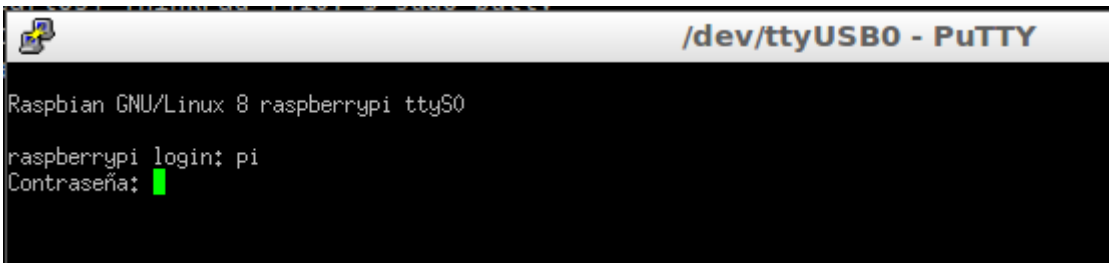


Fig. 57: rpi login using Putty.

[Help]: use "***clear***" command to clear screen terminal, and "***pwd***" to show current path. Use "***ls***" to list files and folders of current folder.
Commands from this document can be copied&pasted into Putty session by pressing *Shift* and *Insert* at the same time.

4.2.3 Preparing for EPICS

We need to prepare the microsd card folder where EPICS will be installed. Everything will be built into a folder named "pi" (*home/pi/Apps/epics*), but first, **a symbolic link to that folder, called */usr/local/epics,*** needs to be defined. By making the EPICS directory into "pi" account, we will be able to modify any of our EPICS resources without needing to gain higher privileges.
"***sudo su***" command allow us to gain access as superuser and that way create the symbolic link. More info about "sudo" and "sudo su" at: https://www.raspberrypi.org/documentation/linux/usage/root.md

Execute these commands:

```
cd ~
mkdir -p ~/Apps/epics
```

NOTE: ~ ≡ AltGr+4

We can check that we are located at the right folder (“*home/pi”)* using “*pwd”* and “*ls”* commands. A new folder “*Apps”* has been created.

```
pwd
ls
```

Fig. 58: Creating Apps folder for EPICS.

Execute commands:

```
sudo su
cd /usr/local
ln -s /home/pi/Apps/epics
exit
cd ~/Apps/epics
```

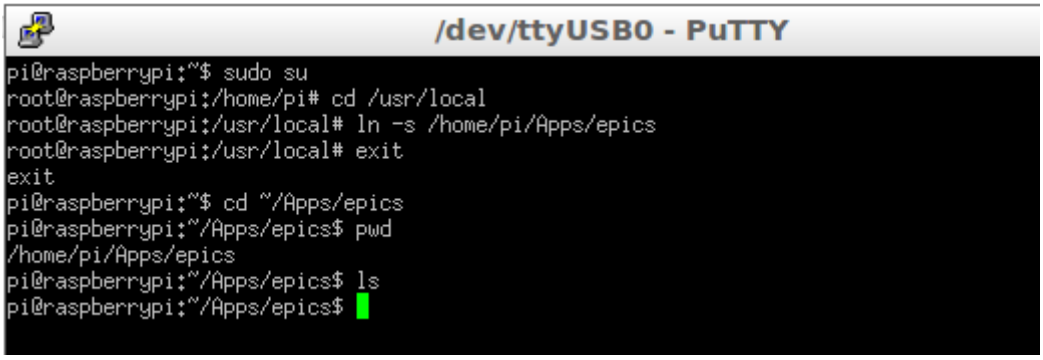


Fig. 59: Creating symbolic link to "epics" folder.

A symbolic link appears (notice a little arrow in the upper left corner of the folder, see Fig. 60) into "*usr/local*" folder, pointing to "*/home/pi/Apps/epics*" folder.

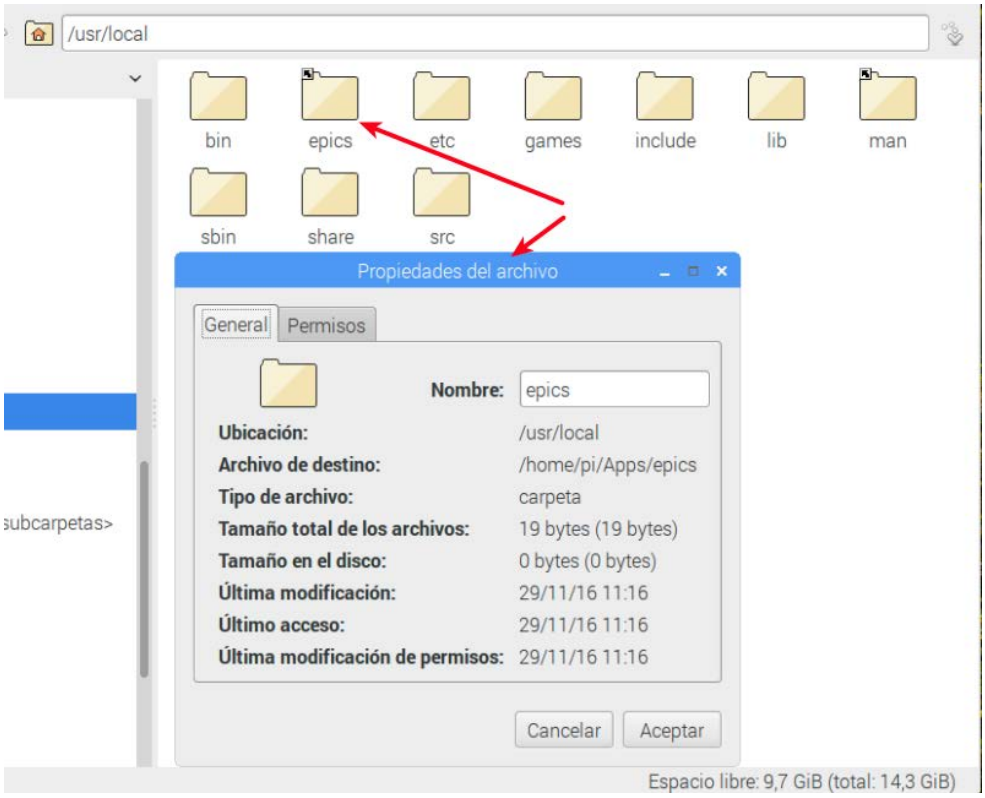


Fig. 60: Folder structure for installing EPICS in Raspbian.

4.2.4 EPICS Base

EPICS base is very easy to build, since Raspbian distro already has all the necessary tools. All that is necessary is to define the host architecture and then build it.

4.2.4.1.1 Downloading EPICS Base

Warning: Web page used as reference, https://prjemian.github.io/epicspi/ mentions EPICS base *3.14.12.3* version, but at the time of writing this document (November 2016), latest stable version of EPICS base is *3.14.12.5* according to http://www.aps.anl.gov/epics/base/R3-14/12.php. It can be downloaded from: https://www.aps.anl.gov/epics/download/base/baseR3.14.12.5.tar.gz

Thus, we will change all references from 3.14.12.3 to 3.14.12.5.

```
1  wget http://www.aps.anl.gov/epics/download/base/baseR3.14.12.3.tar.gz
2  tar xzf baseR3.14.12.3.tar.gz
3  ln -s ./base-3.14.12.3 ./base
```

Cambiamos el 3 por un 5

Fig. 61: Downloading EPICS Base using commands in Raspbian.

We need to check we are located in the correct folder using "*pwd*" command, and then execute the following command to download EPICS base to that folder:

```
wget https://www.aps.anl.gov/epics/download/base/baseR3.14.12.5.tar.gz
```

NOTE: web used as reference declared only "http", not "https", clearly a mistake. As a result of the process, a 1.4 Mb must be downloaded.
Result is shown in Fig. 62, use "*ls*" to check the downloaded file.

Fig. 62: EPICS Base downloading process in Raspbian.

Extract the contents of the compressed file, passing arguments to "tar" command: "zxf" x=extract, z=gzip format, f=file to extract.

```
tar xzf baseR3.14.12.5.tar.gz
```

As a result, a folder named "base-3.14.12.5" with several subdirectories is created (Fig. 63).

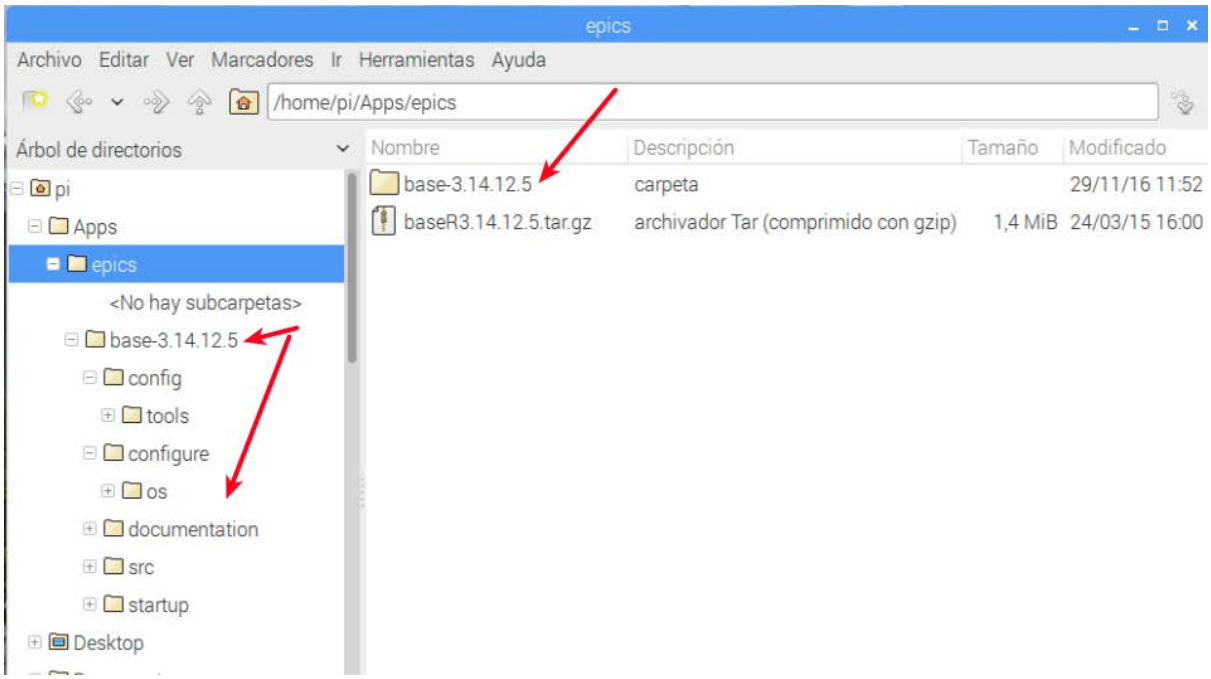


Fig. 63: Folders created after EPICS Base extraction.

Now we create a symbolic link (Fig. 64). *"ln -s"* command will create a symbolic link so anytime that we refer to *"/base"*, we are really referring to *"/base-3.14.12.5"* folder.

```
ln -s ./base-3.14.12.5 ./base
```

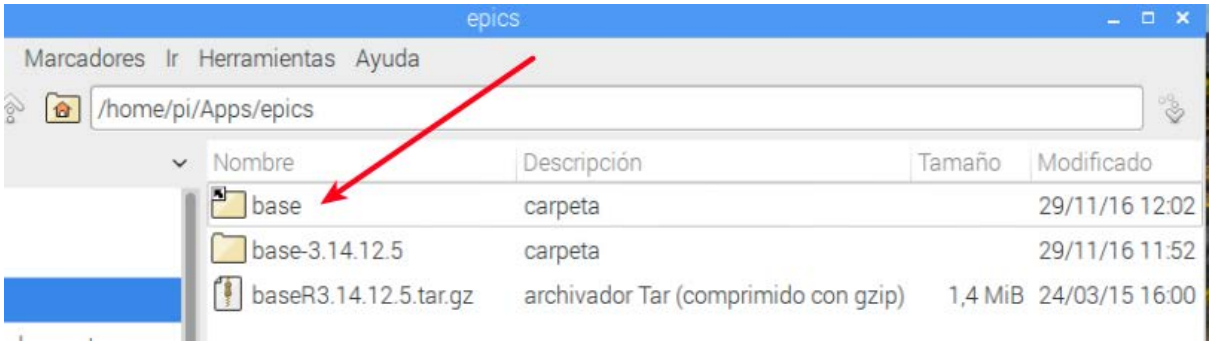


Fig. 64: Symbolic link to EPICS Base folder.

4.2.4.1.2 Building EPICS Base

EPICS Base can be built for many different operating systems and computers. Each build is directed by the EPICS_HOST_ARCH environment variable.

```
export EPICS_HOST_ARCH=$(/usr/local/epics/base/startup/EpicsHostArch)
```

We can check the value of that environment variable, just to be sure that it matches "linux-arm", our Raspberry Pi architecture (Fig. 65).

```
echo $EPICS_HOST_ARCH
```

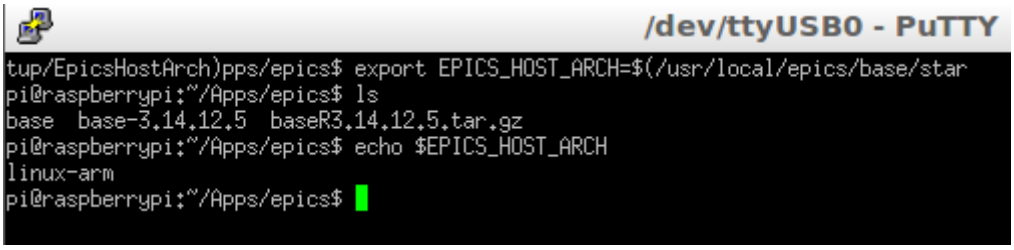


Fig. 65: Checking EPICS_HOST_ARCH environment variable.

For this to be done permanently, we can include *export* command into ~/.bashrc file, using a text editor like "nano" (Fig. 66). There is another way of doing this, adding it to ~/.bash_aliases file (we will see that later)

```
nano ~/.bashrc
```

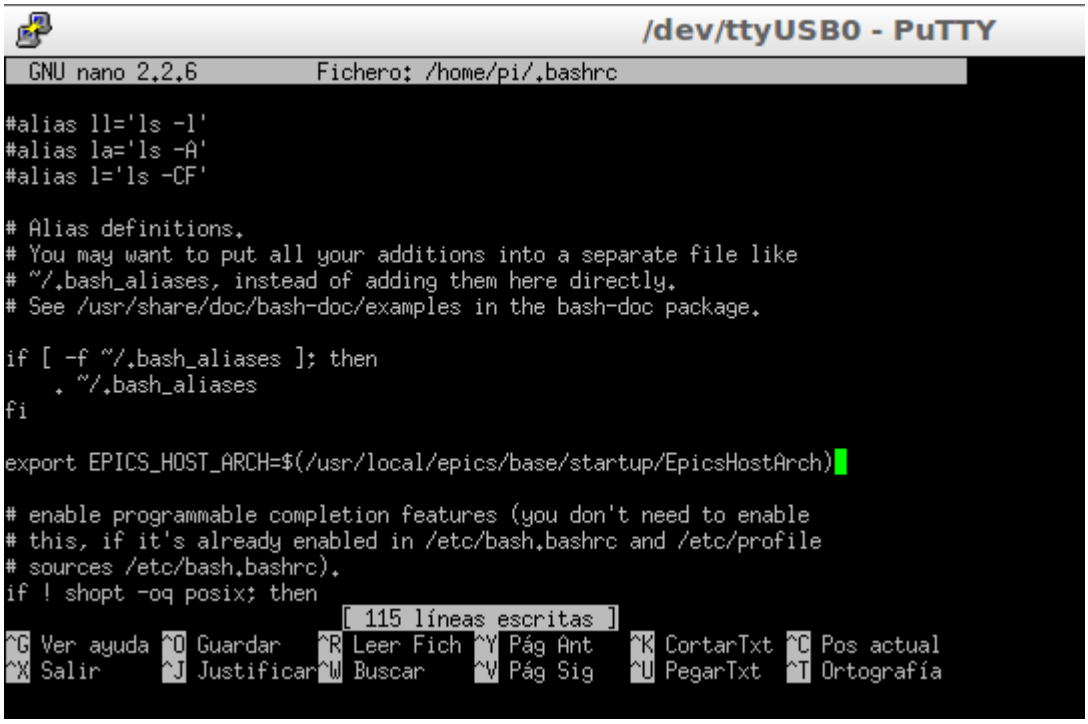


Fig. 66: Editing bashrc file.

Now, build EPICS Base for the first time:

```
cd ~/Apps/epics/base
make
```

Building process will take as much as 30 minutes, and a complete log of all messages shown is available in this file: "mensajes compilacion EPICS 29 nov v1.txt"

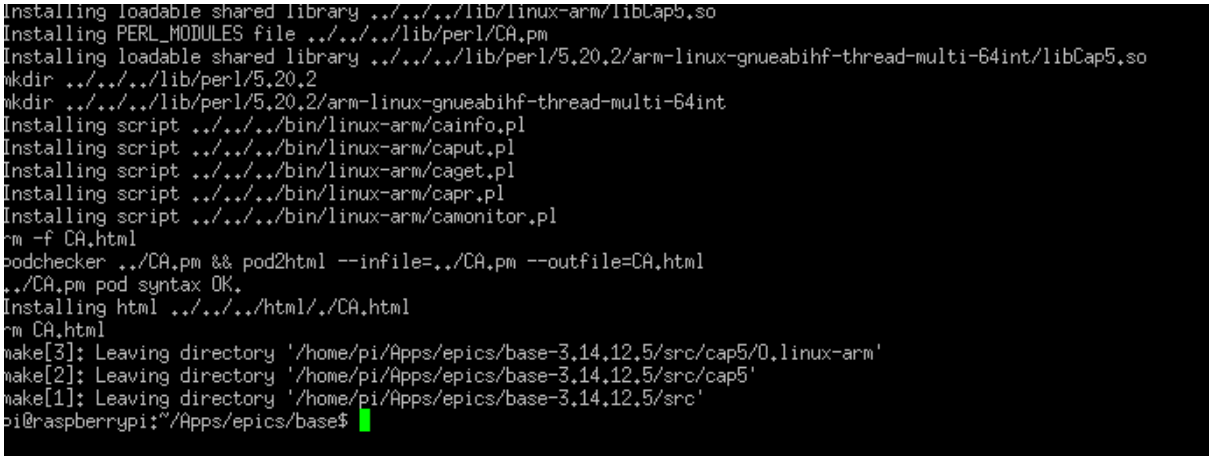


Fig. 67: Building process of EPICS Base.

New folders appear as a result of the compilation, as "*bin", "db", "dbd", "html", "include", "lib",* or *"templates".*

If there are no errors, process will finish as shown in Fig. 67. If there were errors, we can execute *make clean uninstall* command to undo the build, correct the errors, and execute *make* command again.

Disk space is shown in Fig. 68 ("base-3.14.12.5" folder size is 105Mb):

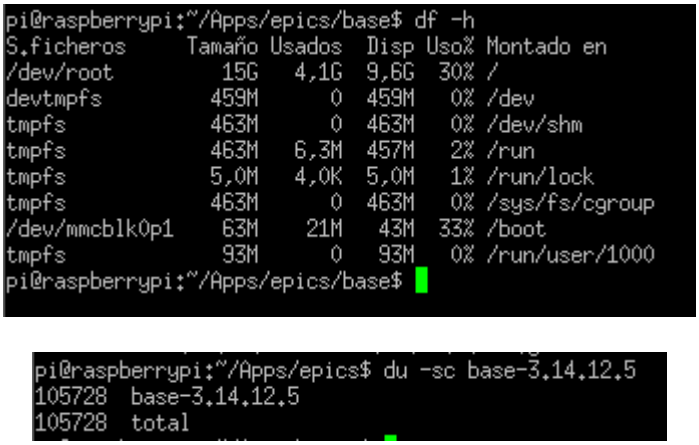


Fig. 68: Disk space after building EPICS Base.

4.2.4.1.3 Starting EPICS Base

Although there is not much added functionality at this point, we can test EPICS, using the first command to launch a basic command line prompt, and then start a basic IOC with the second command:

```
./bin/linux-arm/softIoc
iocInit
```

We see that EPICS has started (Fig. 69).

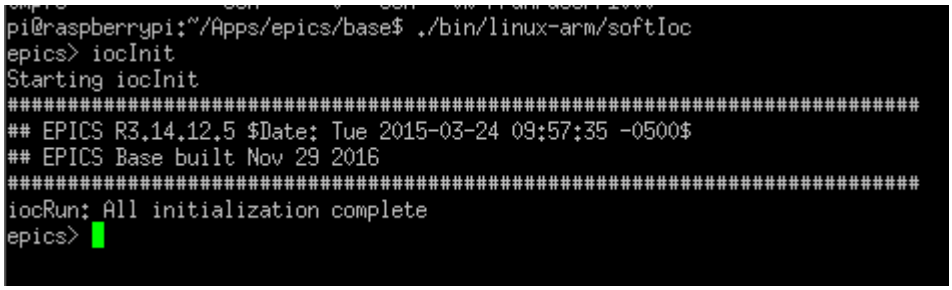


Fig. 69: EPICS works.

4.2.4.1.4 Environment Declarations

To simplify the use of EPICS Base tools, it is best to make a number of declarations in our environment, creating and populating "*~/.bash_aliases*" file.
More info available at:
https://www.raspberrypi.org/documentation/linux/usage/bashrc.md
https://www.raspberrypi.org/forums/viewtopic.php?f=63&t=61945&p=460443

First we need to edit "*~/.bashrc*" file and erase the reference to "*export EPICS_HOST_ARCH=$(/usr/local/epics/base/startup/EpicsHostArch)*" that we made in a previous step. Then we create a new file, "*~/.bash_aliases*" and put these declarations into it:

```
export EPICS_HOST_ARCH=$(/usr/local/epics/base/startup/EpicsHostArch)
export EPICS_ROOT=/usr/local/epics
export EPICS_BASE=${EPICS_ROOT}/base
export EPICS_HOST_ARCH=`${EPICS_BASE}/startup/EpicsHostArch`
export EPICS_BASE_BIN=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}
export EPICS_BASE_LIB=${EPICS_BASE}/lib/${EPICS_HOST_ARCH}
if [ "" = "${LD_LIBRARY_PATH}" ]; then
    export LD_LIBRARY_PATH=${EPICS_BASE_LIB}
else
    export LD_LIBRARY_PATH=${EPICS_BASE_LIB}:${LD_LIBRARY_PATH}
fi
export PATH=${PATH}:${EPICS_BASE_BIN}
```

From /home/pi folder, type:

```
nano ~/.bash_aliases
```

That way the file will be created. Now fill the file with all declarations (Fig. 70). Save the file pressing Ctrl+O, press Intro to confirm file name, and press Ctrl+X to exit.

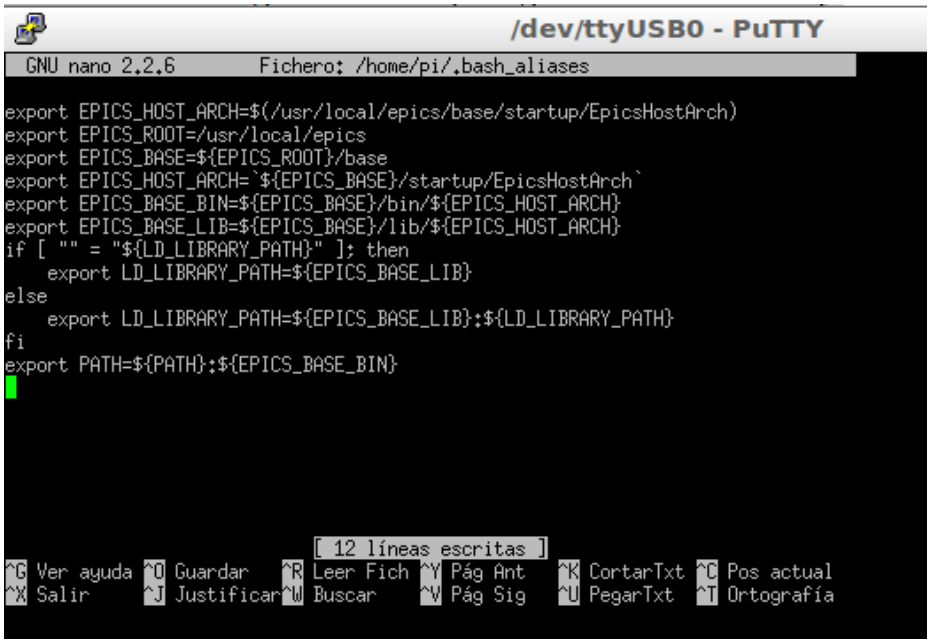


Fig. 70: Including EPICS environment variables into bash_aliases file.

To apply the changes, we can either reboot the rpi or use the command below. Now, using *env* command, Fig. 71 shows that all the environment variables have been loaded correctly:

```
. ~/.bash_aliases
env
```

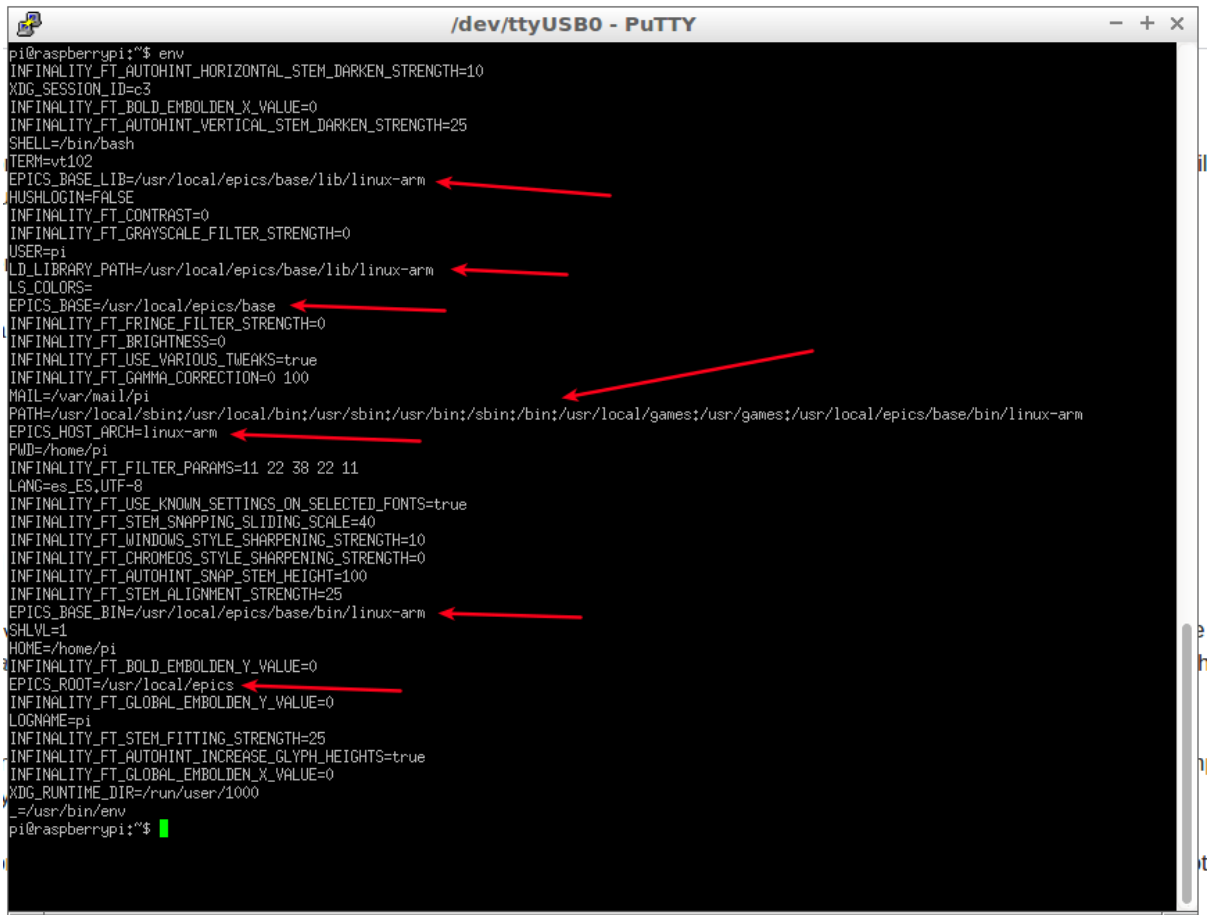


Fig. 71: Checking environment variables using env command.

4.2.5 SynApps

It is a collection of software tools that helps to create a control system for beamlines. It contains beamline-control and data-acquisition components for an EPICS based control system.

More info available at:
https://www1.aps.anl.gov/BCDA/synApps
http://aps.anl.gov/bcda/synApps/synApps_5_8.html

4.2.5.1.1 Download

Information available at our reference web page (https://prjemian.github.io/epicspi/) uses SynApps release 5.6. But nowadays (November 2016) latest release is 5.8 (March 27, 2015). Following instructions have been adapted to 5.8 version.

Remember to move prompt to "~/Apps/epics" folder.

Then download and unzip the compressed source archive file (138 Mb), using the commands shown at Fig. 72.

wget https://www1.aps.anl.gov/files/download/BCDA/synApps/tar/synApps_5_8.tar.gz
tar xzf synApps_5_8.tar.gz

Fig. 72: SynApps download and extraction process.

After extraction process (see Fig. 73), synApps_5_8 folder is 540Mb.

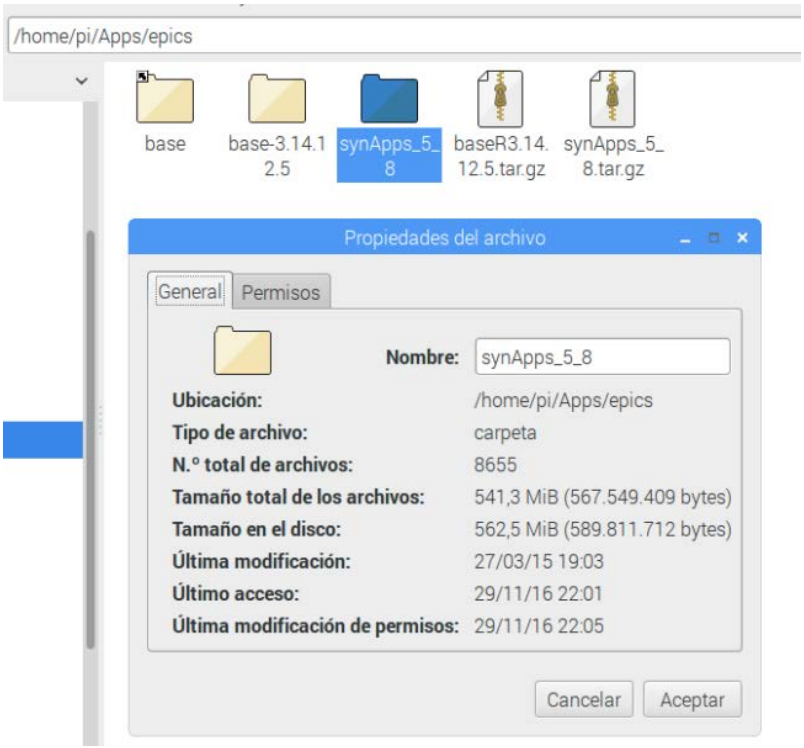


Fig. 73: SynApps folder properties after extraction.

4.2.5.1.2 Configuring

All work will be relative to "support" folder:

```
cd ~/Apps/epics/synApps_5_8/support
```

Follow the instructions in the README file.
Edit "*configure/RELEASE*" file, and change these lines: (Fig. 74)

```
SUPPORT=/usr/local/epics/synApps_5_8/support
EPICS_BASE=/usr/local/epics/base
```

Type:

```
cd configure
nano RELEASE
```

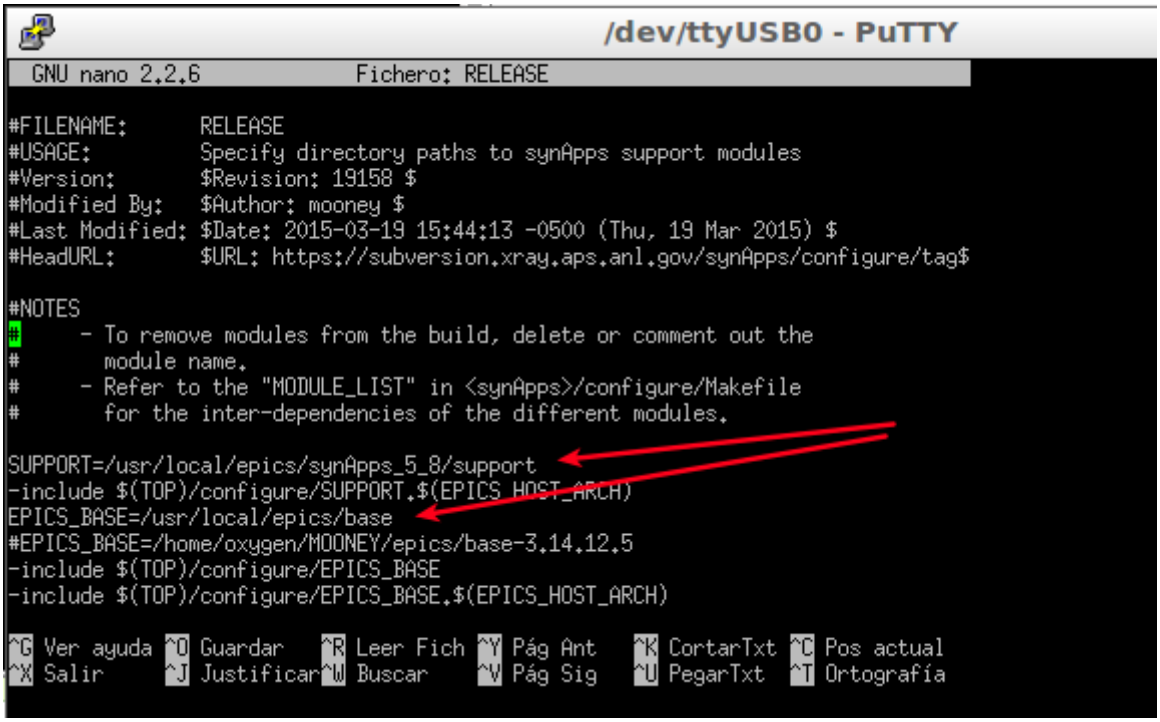


Fig. 74: Configuring SynApps RELEASE file.

Save changes by pressing Ctrl+O, Intro and Ctrl + X

It is needed to propagate the changes to all module RELEASE files, by running:

```
cd ~/Apps/epics/synApps_5_8/support
make release
```

NOTE POSIBLE CHANGE: In web page *https://prjemian.github.io/epicspi/* it is noted that it is necessary to edit "makefile" file to remove support for some modules:

Edit `Makefile` and remove support for these modules:

- ALLEN_BRADLEY
- DAC128V
- IP330

But, if we read notes into the same "*makefile*" file, it is noted that to remove modules from the build, it must be done from synApps/configure/RELEASE file, not from "makefile":

```
#       - To remove modules from the build, delete or comment out the module
#         in the <synApps>/configure/RELEASE file; not here.
```

Therefore, we are going to edit (Fig. 75) "configure/RELEASE", commenting the following modules by including # in front of those lines:
ALLEN_BRADLEY, DAC128V, IP330, IPUNIDIG, LOVE, IP, VAC, SOFTGLUE, QUADEM, DELAYGEN, CAMAC, VME, AREA_DETECTOR, DXP

Execute these commands:

```
cd ~/Apps/epics/synApps_5_8/support/configure
nano RELEASE
```

And put "#" symbol in front of those modules. Leave uncommented the following modules: ALIVE, ADCORE, ADBINARIES, ASYN, AUTOSAVE, BUSY, CALC, CAPUTRECORDER, DEVIOCSTATS, IPAC, MCA, MEASCOMP, MODBUS, MOTOR, OPTICS, SNCSEQ, SSCAN, STD, STREAM, XXX

Save the file pressing Ctrl+O, Intro and Ctrl+X

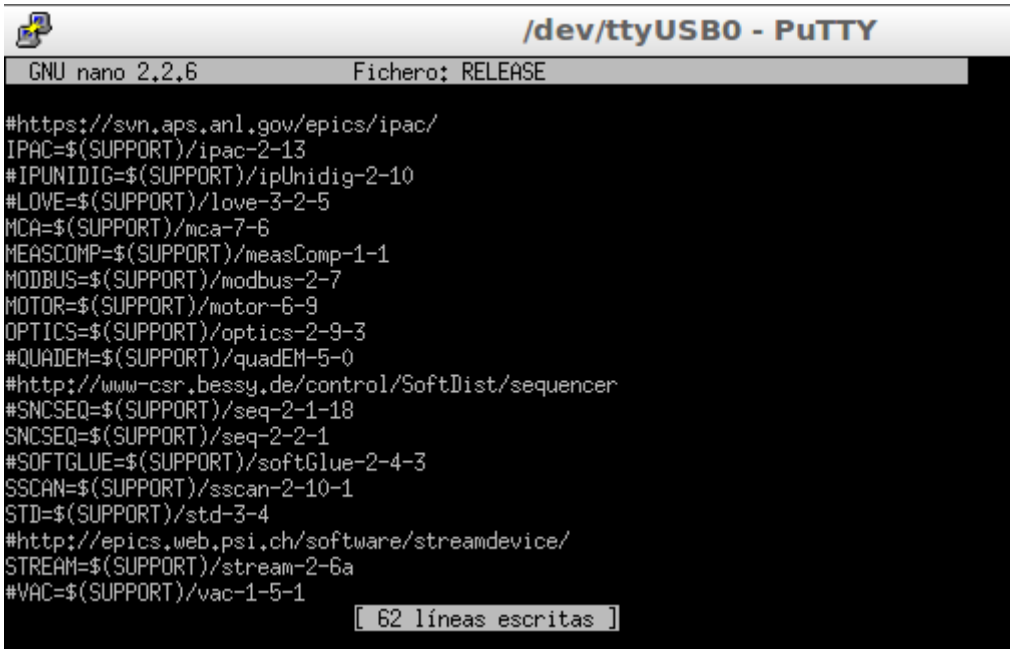


Fig. 75: Removing support for unnecessary modules in RELEASE file.

Propagate changes again:

```
cd ~/Apps/epics/synApps_5_8/support
make release
```

4.2.5.1.3 Reconfiguring xxx module

The **"xxx"** module is an example and template EPICS IOC, demonstrating configuration of many synApps modules. APS (Advanced Photon Source, synchrotron located in USA) beamline IOCs are built using xxx as a template.

This part has not been carried out, because the files that are detailed in the instructions do not appear in 5.8 version.

Fig. 76 shows the part of the instructions that has not been carried out:
https://prjemian.github.io/epicspi/#xxx-module-reconfigure

xxx module: reconfigure ¶

The **xxx** module is an example and template EPICS IOC, demonstrating configuration of many synApps modules. APS beam line IOCs are built using *xxx* as a template.

In `xxx-5-6/configure/RELEASE`, place a comment on lines 19 and 32 to remove build support for *areaDetector* in *xxx*:

```
#AREA_DETECTOR=$(SUPPORT)/areaDetector-1-8beta1

#IP=$(SUPPORT)/ip-2-13
```

In `xxx-5-6/xxxApp/src/xxxCommonInclude.dbd`, place a comment on line 34:

```
#include "ipSupport.dbd"
```

Then, in `xxx-5-6/xxxApp/src/Makefile`, comment out all lines that refer to *areaDetector* components, such as *ADsupport*, "NDPlugin*, *simDetector*, and *netCDF*, as well as *dxp* support. Here are the lines I found:

```
#iocxxxWin32_DBD += ADSupport.dbd  NDFileNetCDF.dbd
#xxx_LIBS_WIN32 += ADBase NDPlugin netCDF
#iocxxxCygwin_DBD += ADSupport.dbd  NDFileNetCDF.dbd
#xxx_LIBS_cygwin32 += ADBase NDPlugin netCDF
#iocxxxCygwin_DBD += ADSupport.dbd NDFileNetCDF.dbd
#xxx_LIBS_cygwin32 += ADBase NDPlugin netCDF
#iocxxxLinux_DBD += ADSupport.dbd  NDFileNetCDF.dbd
#xxx_LIBS_Linux += ADBase NDPlugin netCDF

#iocxxxCygwin_DBD += simDetectorSupport.dbd commonDriverSupport.dbd
#xxx_LIBS_cygwin32 += simDetector
#iocxxxLinux_DBD += simDetectorSupport.dbd commonDriverSupport.dbd
#xxx_LIBS_Linux += simDetector

#xxx_Common_LIBS += ip
```

Fig. 76: Reconfiguring "xxx" module.

4.2.5.1.4 Installing necessary EPICS extensions

First, we need to set up the extensions subdirectory structure, more info at: http://aps.anl.gov/epics/extensions/configure/index.php

Type these commands to download and build the structure (12kb file, result is shown in Fig. 77):

```
cd ~/Apps/epics
wget http://www.aps.anl.gov/epics/download/extensions/extensionsTop_20120904.tar.gz
tar xzf extensionsTop_20120904.tar.gz
```

Fig. 77: Downloading and unpacking EPICS extensions structure.

File structure is shown in Fig. 78.

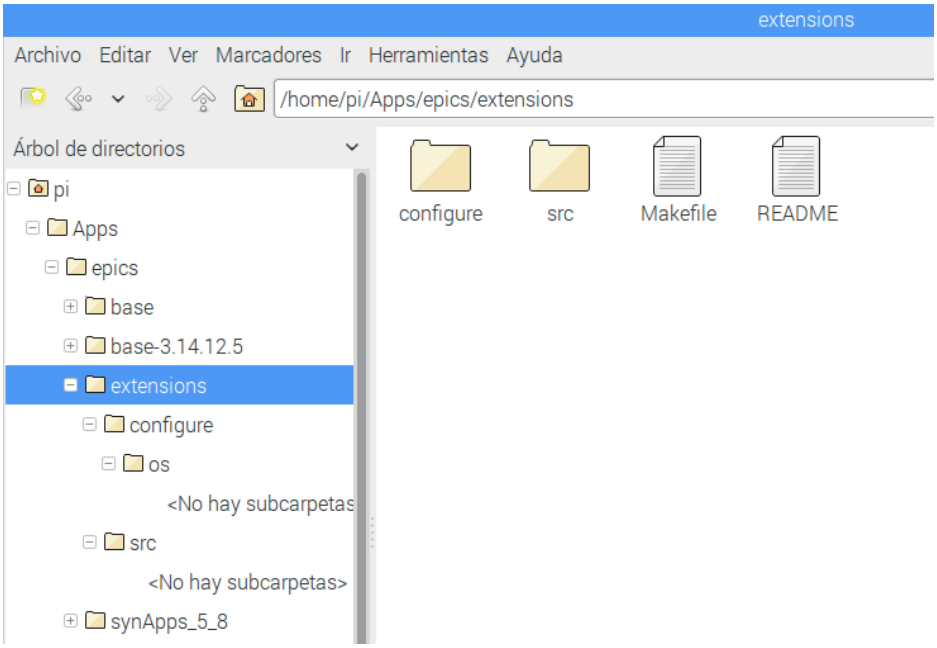


Fig. 78: EPICS extensions directory structure.

Warning: SynApps needs EPICS extension called ***"msi".*** 1.5 version is used in reference web page, but latest release today (November, 2016) is 1.7.
http://www.aps.anl.gov/epics/extensions/msi/index.php

Let's download, unpack and install **"msi"** extension executing these commands:

```
wget http://www.aps.anl.gov/epics/download/extensions/msi1-7.tar.gz
cd extensions/src
tar xzf ../../msi1-7.tar.gz
cd msi1-7
make
```

If everything goes according to plan, “make” result will be like Fig. 79.

Fig. 79: Extracting and building msi extension.

Make these additional declarations in your environment, i.e., include new declarations in “*~/.bash_aliases”* file.

Execute these two commands:

```
cd ~
nano .bash_aliases
```

Include these lines in the file:

```
export EPICS_EXT=${EPICS_ROOT}/extensions
export EPICS_EXT_BIN=${EPICS_EXT}/bin/${EPICS_HOST_ARCH}
export EPICS_EXT_LIB=${EPICS_EXT}/lib/${EPICS_HOST_ARCH}
if [ "" = "${LD_LIBRARY_PATH}" ]; then
    export LD_LIBRARY_PATH=${EPICS_EXT_LIB}
else
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${EPICS_BASE_LIB}
fi
export PATH=${PATH}:${EPICS_EXT_BIN}
```

This is “*.bash_aliases”* file content:

```
export EPICS_HOST_ARCH=$(/usr/local/epics/base/startup/EpicsHostArch)
export EPICS_ROOT=/usr/local/epics
export EPICS_BASE=${EPICS_ROOT}/base
```

```
export EPICS_HOST_ARCH=`${EPICS_BASE}/startup/EpicsHostArch`
export EPICS_BASE_BIN=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}
export EPICS_BASE_LIB=${EPICS_BASE}/lib/${EPICS_HOST_ARCH}
if [ "" = "${LD_LIBRARY_PATH}" ]; then
    export LD_LIBRARY_PATH=${EPICS_BASE_LIB}
else
    export LD_LIBRARY_PATH=${EPICS_BASE_LIB}:${LD_LIBRARY_PATH}
fi
export PATH=${PATH}:${EPICS_BASE_BIN}
export EPICS_EXT=${EPICS_ROOT}/extensions
export EPICS_EXT_BIN=${EPICS_EXT}/bin/${EPICS_HOST_ARCH}
export EPICS_EXT_LIB=${EPICS_EXT}/lib/${EPICS_HOST_ARCH}
if [ "" = "${LD_LIBRARY_PATH}" ]; then
    export LD_LIBRARY_PATH=${EPICS_EXT_LIB}
else
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${EPICS_BASE_LIB}
fi
export PATH=${PATH}:${EPICS_EXT_BIN}
```

Package ***"re2c"*** is also needed (see http://re2c.org/)

Execute the following command:

```
sudo apt-get install re2c
```

And this is the result (Fig. 80):

Fig. 80: Installing re2c package.

4.2.5.1.5 Building SynApps

Now, we are going to build "SynApps", executing these commands:

```
cd ~/Apps/epics/synApps_5_8/support
make release
make rebuild
```

```
Inflating database from ../ioc.template
msi    -I. -I.. -I../O.Common -I../../../db -I/usr/local/epics/base/db ../ioc.template > ioc.tmp
/bin/sh: 1: msi: not found
/usr/local/epics/base/configure/RULES.Db:321: recipe for target '../O.Common/ioc.db' failed
make[4]: *** [../O.Common/ioc.db] Error 127
make[4]: Leaving directory '/home/pi/Apps/epics/synApps_5_8/support/devIocStats-3-1-13/iocAdmin/Db/O.linux-arm'
/usr/local/epics/base/configure/RULES_ARCHS:64: recipe for target 'install.linux-arm' failed
make[3]: *** [install.linux-arm] Error 2
make[3]: Leaving directory '/home/pi/Apps/epics/synApps_5_8/support/devIocStats-3-1-13/iocAdmin/Db'
/usr/local/epics/base/configure/RULES_DIRS:87: recipe for target 'Db.install' failed
make[2]: *** [Db.install] Error 2
make[2]: Leaving directory '/home/pi/Apps/epics/synApps_5_8/support/devIocStats-3-1-13/iocAdmin'
/usr/local/epics/base/configure/RULES_DIRS:87: recipe for target 'iocAdmin.install' failed
make[1]: *** [iocAdmin.install] Error 2
make[1]: Leaving directory '/home/pi/Apps/epics/synApps_5_8/support/devIocStats-3-1-13'
/usr/local/epics/base/configure/RULES_DIRS:87: recipe for target '/usr/local/epics/synApps_5_8/support/devIocStats-3-1-1
3.install' failed
make: *** [/usr/local/epics/synApps_5_8/support/devIocStats-3-1-13.install] Error 2
pi@raspberrypi:~/Apps/epics/synApps_5_8/support$ exit
logout
```

Fig. 81: Building SynApps.

[Help]: If the process results in an error like the one shown in Fig. 81, shutdown the rpi, boot again, and repeat the three previous commands.

Final step will take as much as 30 minutes, depending on your computer's processor speed (complete log available at: putty_02_12_090443.log).

4.2.6 PyEpics

It is possible to run "PyEpics" from Matt Newville on the rpi, more information at: http://cars.uchicago.edu/software/python/pyepics3/

"PyEpics is an interface for the Channel Access (CA) library of the Epics Control System to the Python Programming language. The PyEpics package provides a base epics module to python, with methods for reading from and writing to Epics Process Variables (PVs) via the CA protocol. The package includes a fairly complete, thin layer over the low-level Channel Access library in the ca module, and higher-level abstractions built on top of this basic functionality."

4.2.6.1.1 Preparing Python

To simplify installation, we will use "*easy_install*" from "*setuptools*".
More information about it is available at this link: http://setuptools.readthedocs.io/en/latest/easy_install.html

We need to work as superuser (root access), using the command:

```
sudo su
```

Prompt changes to a symbol “#”, indicating we have entered superuser mode:

```
pi@raspberrypi:~$ sudo su
root@raspberrypi:/home/pi#
```

Install *setuptools* and *ipython* packages from OS repository:

```
sudo apt-get install python-setuptools ipython
```

This is the result (Fig. 82).

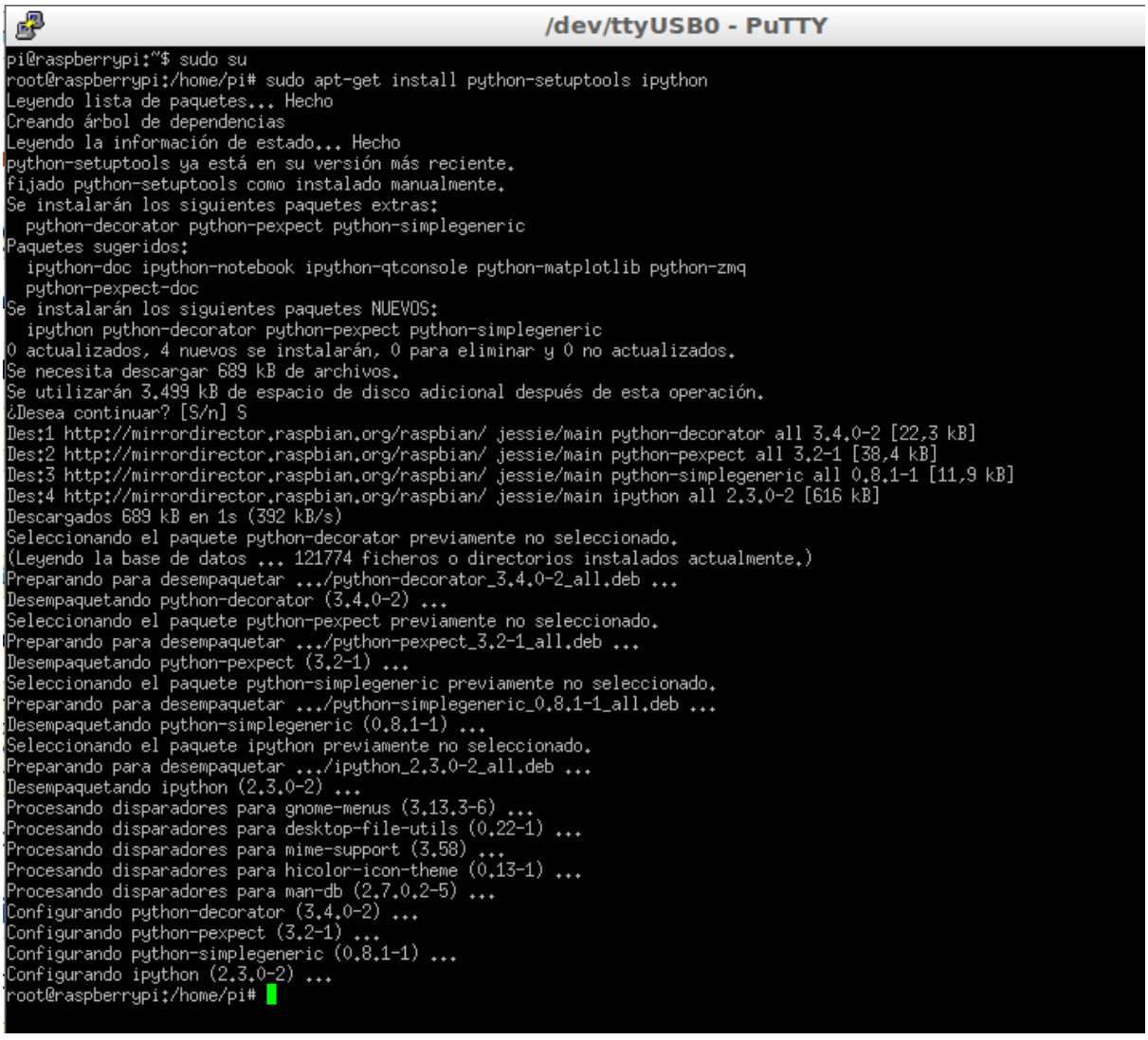


Fig. 82: Installing Python in Raspbian.

Let’s check which version of Python will be run. First we can check where *python* is located, and then we can see information about that folder:

```
which python
ls -lAFg /usr/bin/python
```

Fig. 83 show us that Python is found in "/usr/bin/python" directory, and that version is 2.7.
More detailed information will be shown if we execute Python environment:

```
/usr/bin/python
```

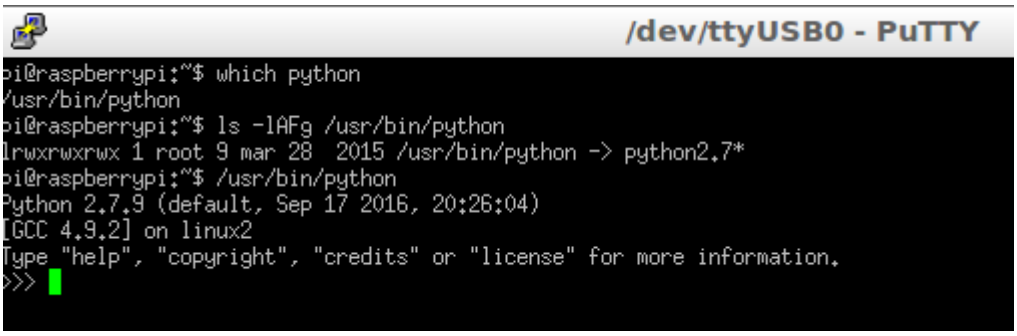


Fig. 83: Finding out Python version.

Version 2.7.9 (September, 2016) is shown.
Exit from Python environment pressing Ctrl+D.

4.2.6.1.2 Installing PyEpics

With the "*setuptools"* installed, it is fairly simple to install PyEpics (still as root):

```
easy_install -U PyEpics
```

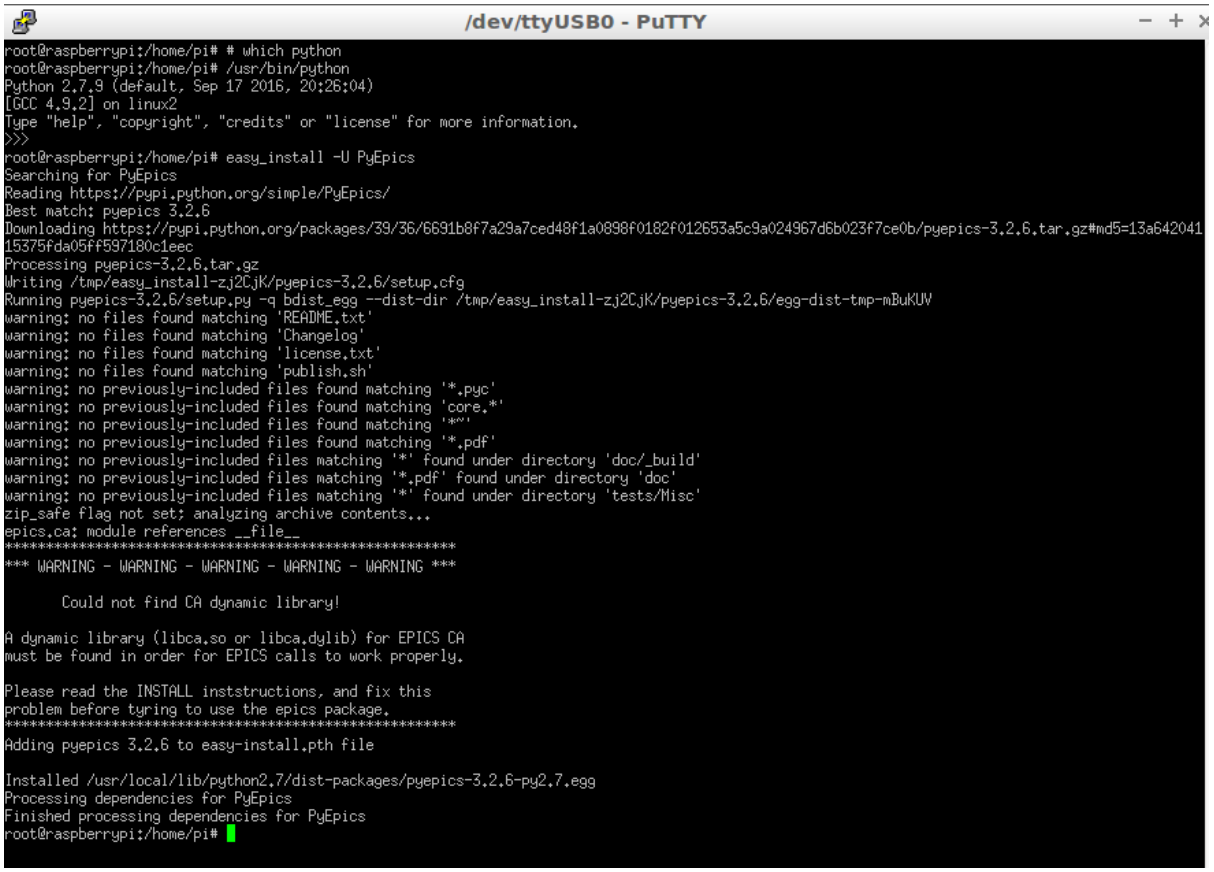


Fig. 84: Installing PyEpics.

Warning: Fig. 84 shows that PyEpics 3.2.6 has been installed, unlike reference web (https://prjemian.github.io/epicspi/#pyepics) where version 3.2.1 was installed. Therefore, all routes and commands listed below have been adapted to reflect *epics* (3.14.12.5) and *PyEpics* (3.2.6) versions that we are using.

A warning message about missing dynamic libraries (*libca and libCom*) is shown.
Solve the issue executing these commands (still as root):

```
cd /usr/local/lib/python2.7/dist-packages/pyepics-3.2.6-py2.7.egg
cp /home/pi/Apps/epics/base-3.14.12.5/lib/linux-arm/libca.so.3.14 ./
cp /home/pi/Apps/epics/base-3.14.12.5/lib/linux-arm/libCom.so.3.14 ./
ln -s libca.so.3.14  libca.so
ln -s libCom.so.3.14  libCom.so
```

Exit from *root* back to the *pi* account session:

```
exit
```

The resulting file structure is shown in Fig. 85.

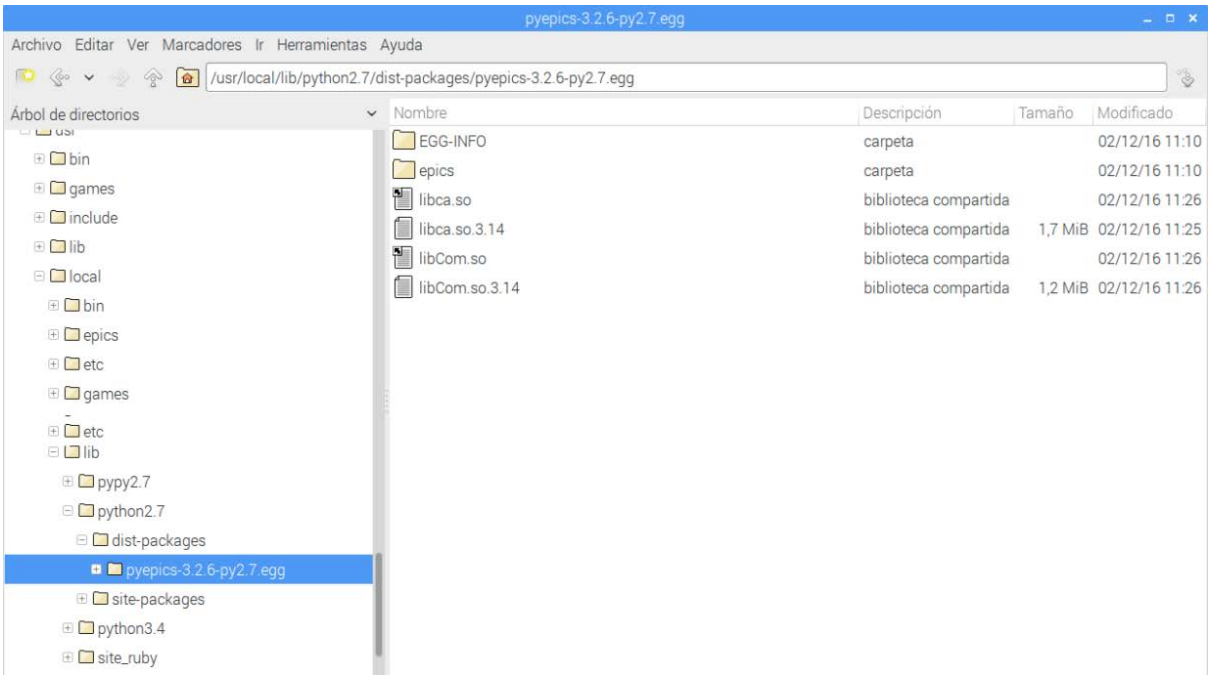


Fig. 85: PyEpics file structure.

4.2.6.1.3 Testing PyEpics

In order to test PyEpics, create a file called "*verify.pi*", folder "*/home/pi*", with this content:

```
#!/usr/bin/env python
import epics
print epics.__version__
print epics.__file__
```

These commands will edit the file, make it executable, and run it:

```
nano verify.pi
chmod +x verify.py
./verify.py
```

The result shows that PyEpics is installed, but it does not prove that EPICS is working:

```
pi@raspberrypi:~$ ./verify.py
3.2.6
/usr/local/lib/python2.7/dist-packages/pyepics-3.2.6-py2.7.egg/epics/__init__.pyc
pi@raspberrypi:~$
```

4.2.6.1.4 Testing PyEpics with an IOC

The best way to test PyEpics is to use the softIOC support from EPICS base, creating a simple EPICS database.

It is recommended to create several terminal windows, because we will use several tools at the same time.

Create a simple EPICS database file, "*simple.db"* (Fig. 86):

```
record(bo, "rpi:trigger")
{
        field(DESC, "trigger PV")
        field(ZNAM, "off")
        field(ONAM, "on")
}
record(stringout, "rpi:message")
{
        field(DESC, "message on the RPi")
        field(VAL,  "RPi default message")
}
```

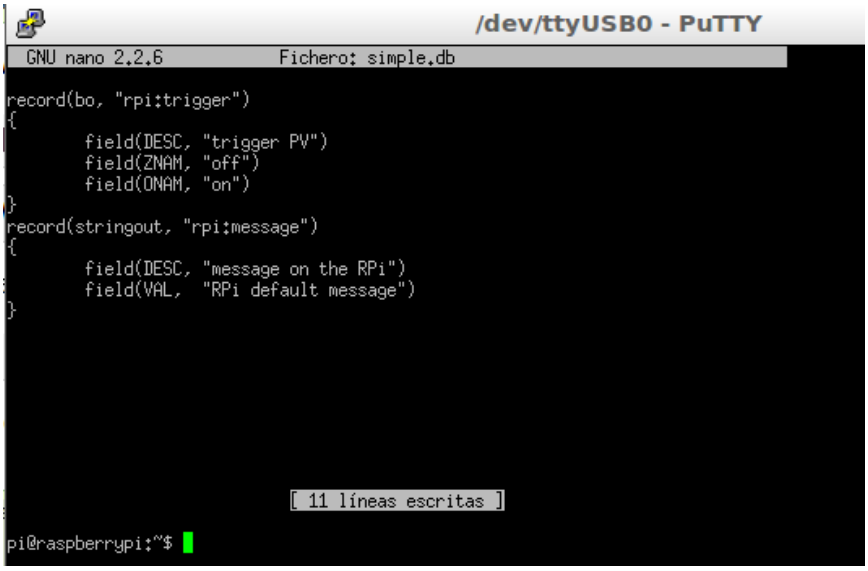


Fig. 86: Creating simple.db EPICS database file.

We have defined two EPICS records, ***rpi:trigger*** and ***rpi:message***. The first record can take values 0 or 1 (strings "off and "on"), and the second record is a string.

In order to change PVs values, another python file must be created ("*test.py"*). Put this code into the file:

```
#!/usr/bin/env python
import epics
print epics.caget('rpi:trigger.DESC')
print epics.caget('rpi:trigger')
print epics.caget('rpi:message.DESC')
print epics.caget('rpi:message')
epics.caput('rpi:message', 'setting trigger')
epics.caput('rpi:trigger', 1)
print epics.caget('rpi:trigger.DESC')
print epics.caget('rpi:trigger')
print epics.caget('rpi:message.DESC')
print epics.caget('rpi:message')
```

```
epics.caput('rpi:message', 'clearing trigger')
epics.caput('rpi:trigger', 0)
print epics.caget('rpi:trigger.DESC')
print epics.caget('rpi:trigger')
print epics.caget('rpi:message.DESC')
print epics.caget('rpi:message')
```

Edit *"test.py"* file, and make it executable with these commands:

```
nano test.py
chmod +x test.py
```

Now run the EPICS softIOC with our simple database example:

```
softIoc -d simple.db
```

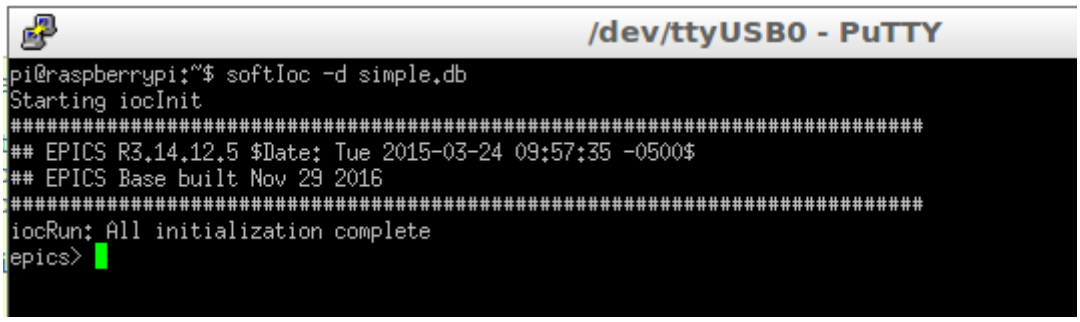


Fig. 87: Running a simple EPICS softIOC.

Fig. 87 shows that EPICS is running correctly. Now we can use "dbl" command to view the records inside the simple database. Two PVs are shown, "rpi:trigger" and "rpi:message" (see Fig. 88)

```
epics>dbl
```

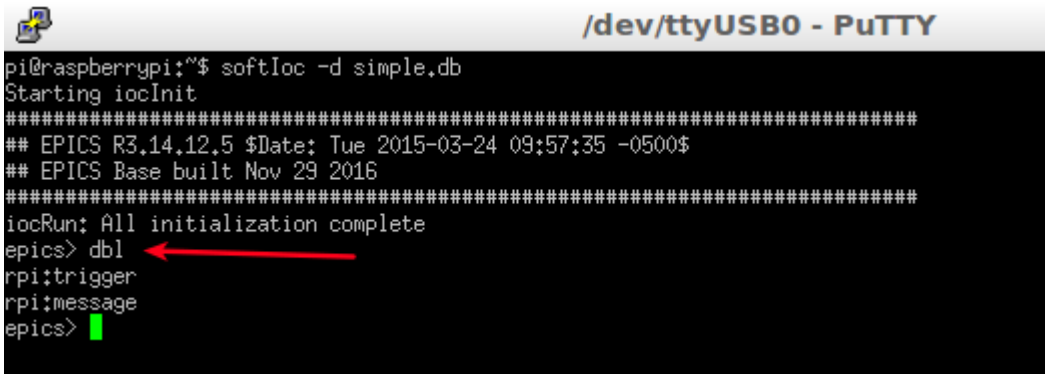


Fig. 88: Using dbl command to view database records.

In a separate terminal window, we will use "*camonitor*" command to watch the softIOC for any changes to EPICS PVs.

Type:

```
camonitor rpi:trigger rpi:trigger.DESC rpi:message rpi:message.DESC
```

And PVs are shown (Fig. 89), right now they have no values.

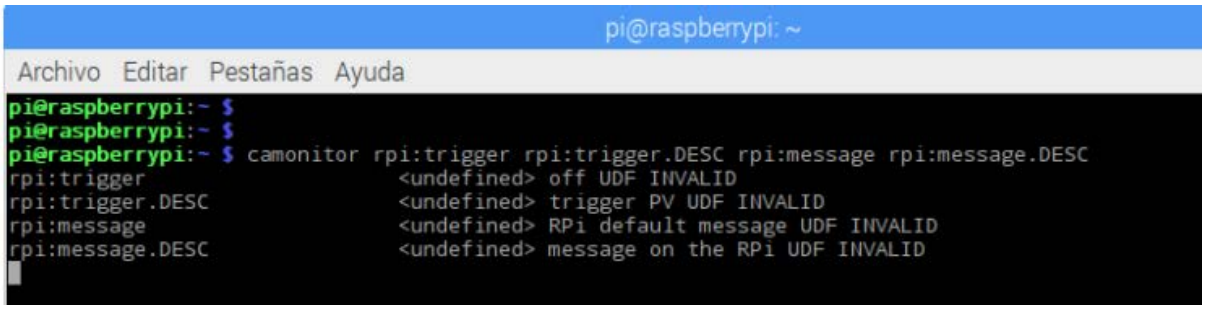


Fig. 89: Monitoring PVs using camonitor.

Next step is to communicate with softIOC PVs, running "*test.py*" file (from another terminal windows) that will make the PVs to change their values. Type:

```
./test.py
```

As a result, PVs values begin to change:

Fig. 90: Forcing PVs to change their values using test.py.

And these changes have been reflected also in another window, where "camonitor" is monitoring those PVs:

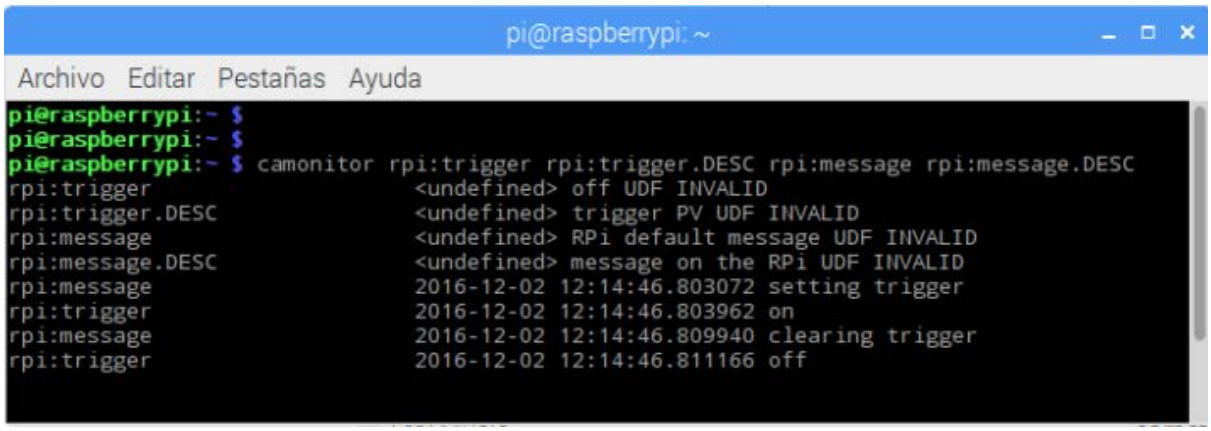


Fig. 91: Watching PVs values using camonitor.

Fig. 92 shows a screenshot with three terminal windows running at once:

Window 1, softIOC running simple.db simple database:

```
softIoc -d simple.db
```

Window 2, test file written in python that will cause changes in the PVs values:

```
./test.py
```

Window 3, "camonitor" running and monitoring softIOC PVs:

```
camonitor rpi:trigger rpi:trigger.DESC rpi:message rpi:message.DESC
```

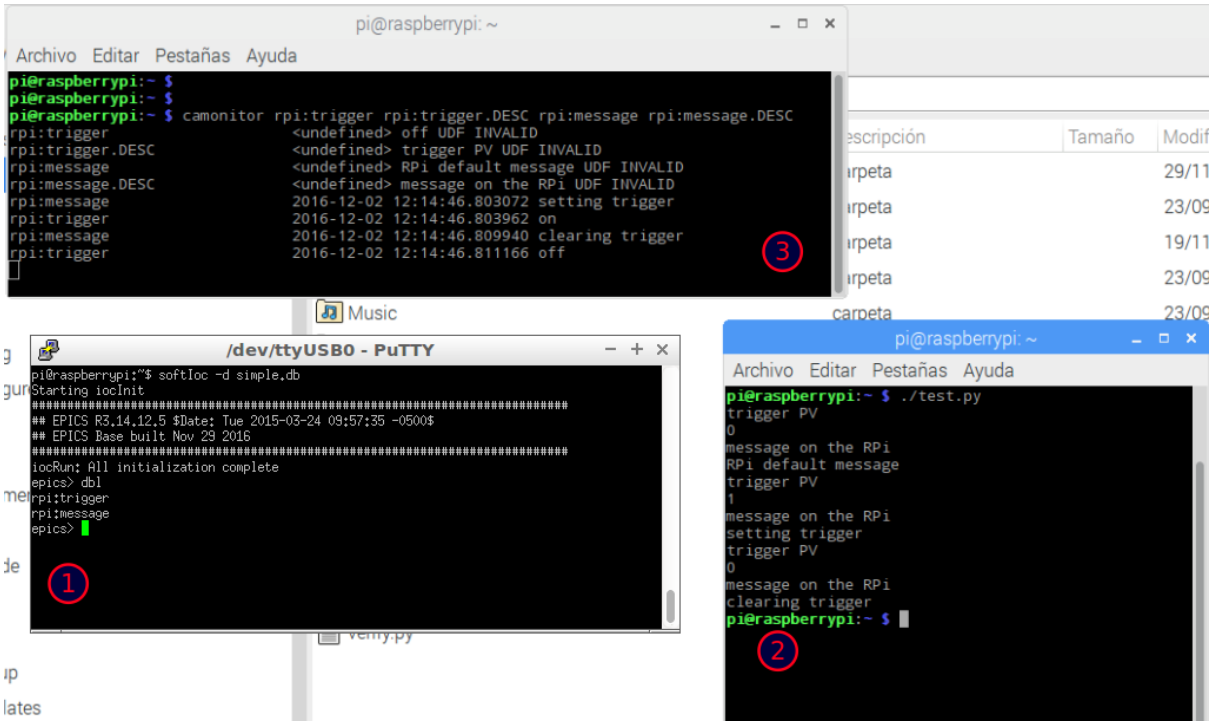


Fig. 92: Changing and monitoring PVs screenshot.

To finish softIOC execution (window 1), type:

```
exit
```

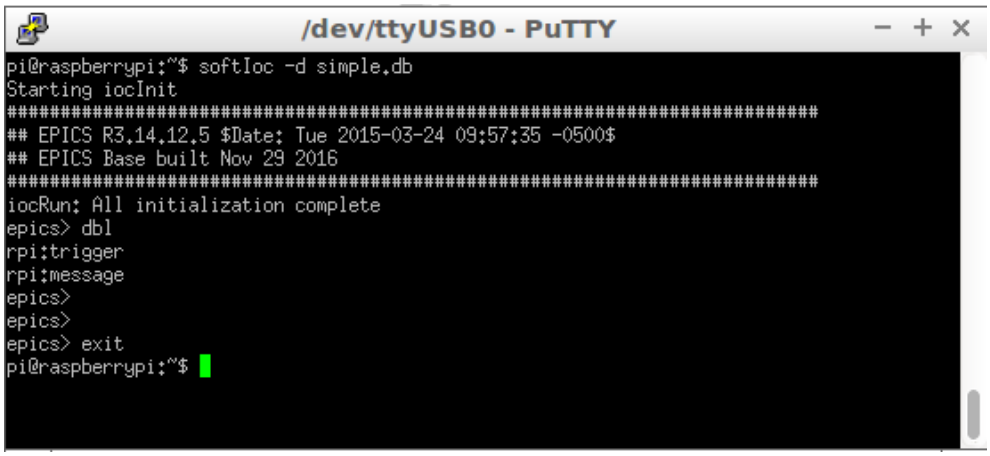
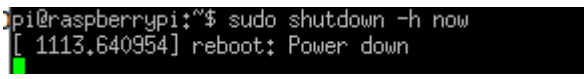


Fig. 93: Stopping softIOC.

Finally, turn off the rpi using this command:

```
sudo shutdown -h now
```



5 INSTALLING VMWARE VIRTUAL MACHINE AND BUILDROOT

5.1 Documentation on VMware, Buildroot and Ubuntu

- **VMware**
  VMware documentation main page
  Workstation Player 12 for Linux Documentation Center
  Ubuntu 14.04 LTS installation guide in the virtual machine (Guest)
  VMware Knowledge Base

- **Buildroot**
  Buildroot Training
  "Buildroot user manual" at [RD5]
  Conference "Learning the Basics of Buildroot - Thomas Petazzoni", YouTube (2015)
  Conference "Building embedded Linux systems made easy" Thomas Petazzoni, YouTube (2014)
  Building a Linux Filesystem on Raspberry Pi 3, Rohit Walavalkar

- **Ubuntu**
  Ubuntu 14 documentation available at [RD4]

5.2 Installing VMware Player 12.5.1

Download **VMware Workstation Player version 12.5.1-4542065** (Fig. 94) from:
http://www.vmware.com/products/workstation.html
http://www.vmware.com/products/workstation-for-linux.html (Linux 64 bits file, .bundle extension)
File: "VMware-Player-12.5.1-4542065.x86_64.bundle"

We need to download also **Ubuntu 14.04.5 LTS (Trusty Tahr) 32 bits** from:
https://www.ubuntu.com/download/alternative-downloads
A torrent file has been used, torrent file is this.

Downloaded .iso file is "ubuntu-14.04.5-desktop-i386.iso"

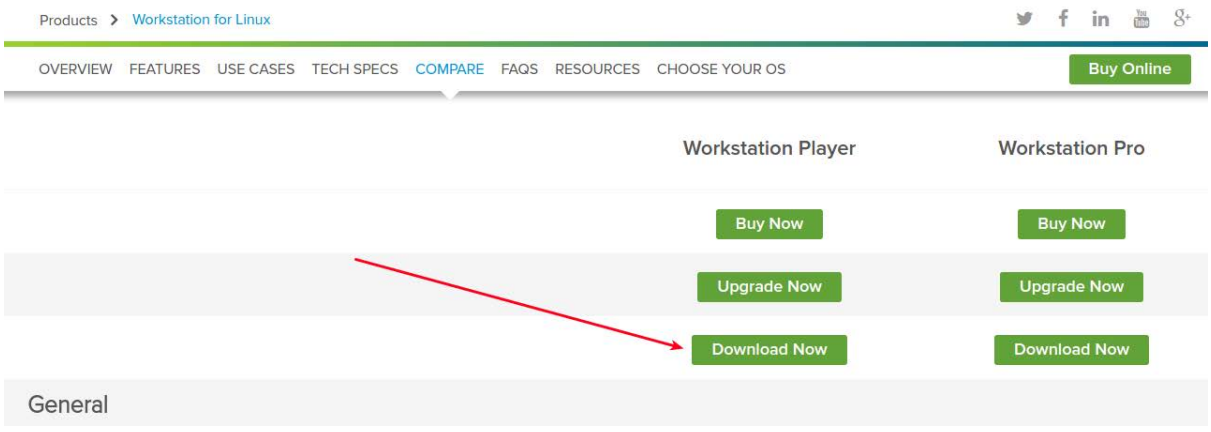


Fig. 94: Downloading VMWare Workstation Player 12.5.1-4542065.

For virtual machine purposes, our Host will be Lubuntu 15.10, Lenovo T410 computer equipped with Intel i5 M520 processor, and our Guest will be a virtual machine running Ubuntu 14.04.5 OS.

Host (Lenovo T410):

▾ Version	
Kernel	Linux 4.2.0-42-generic (x86_64)
Compiled	#49-Ubuntu SMP Tue Jun 28 21:26:26 UTC 2016
C Library	Unknown
Default C Compiler	GNU C Compiler version 5.2.1 20151010 (Ubuntu 5.2.1-22ubuntu2)
Distribution	Ubuntu 15.10
▾ Current Session	
Computer Name	carlosj-ThinkPad-T410
User Name	carlosj (CarlosJ)
Home Directory	/home/carlosj
Desktop Environment	LXDE (Lubuntu)
▾ Misc	
Uptime	2 hours, 15 minutes
Load Average	0,00, 0,00, 0,00

Fig. 95 to Fig. 100 show VMware installation process. From a terminal windows, type:

```
sudo sh VMware-Player-12.5.1-4542065.x86_64.bundle
```

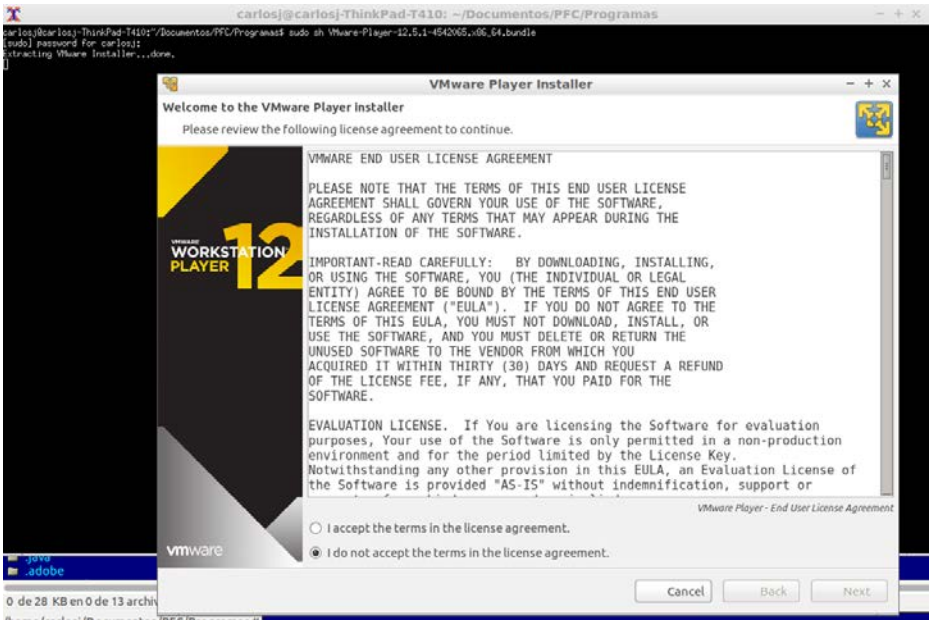


Fig. 95: VMWare Player end user license agreement.

Press “Next” twice if you agree to the licenses terms (both Player and OVF Tool), and answer “No” to request of looking for updates.

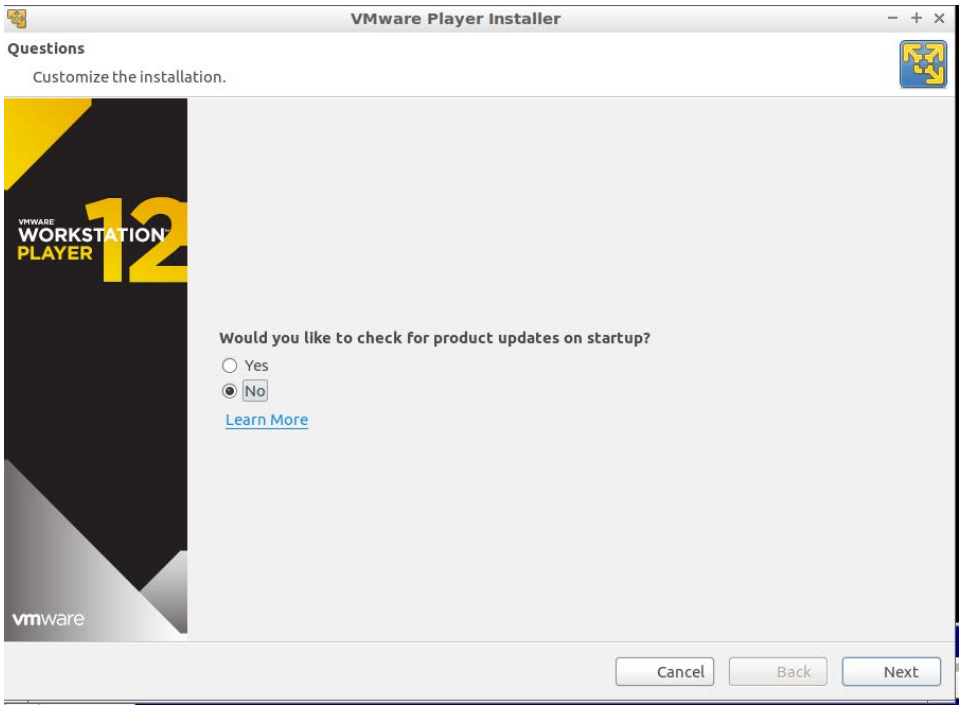


Fig. 96: VMware Player check for updates screen.

Leave license field empty

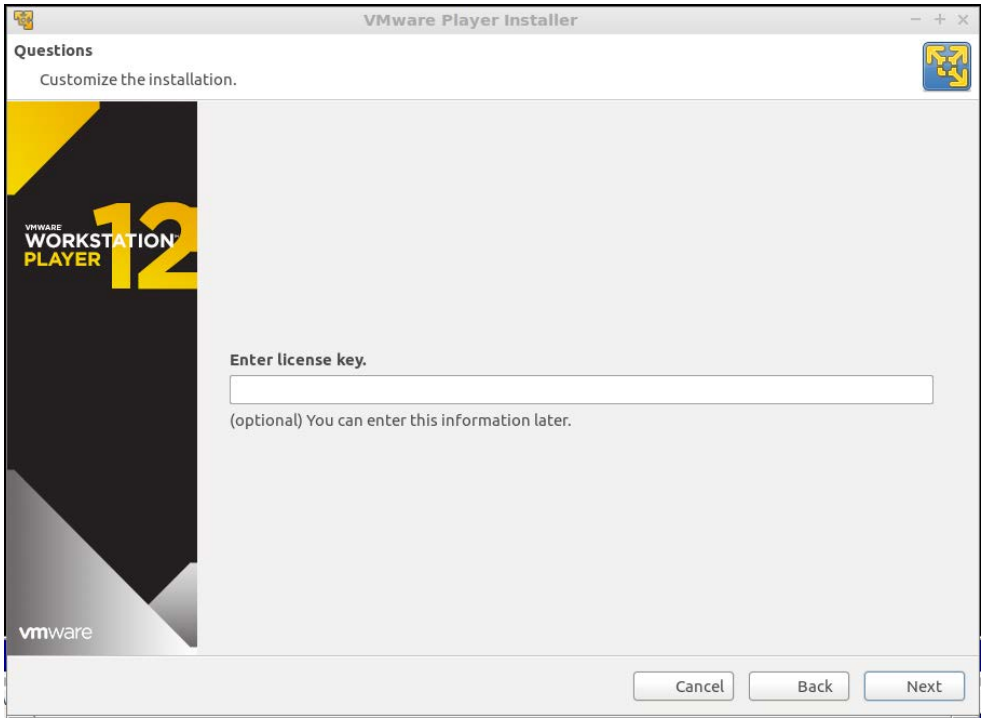


Fig. 97: VMware Player license key screen.

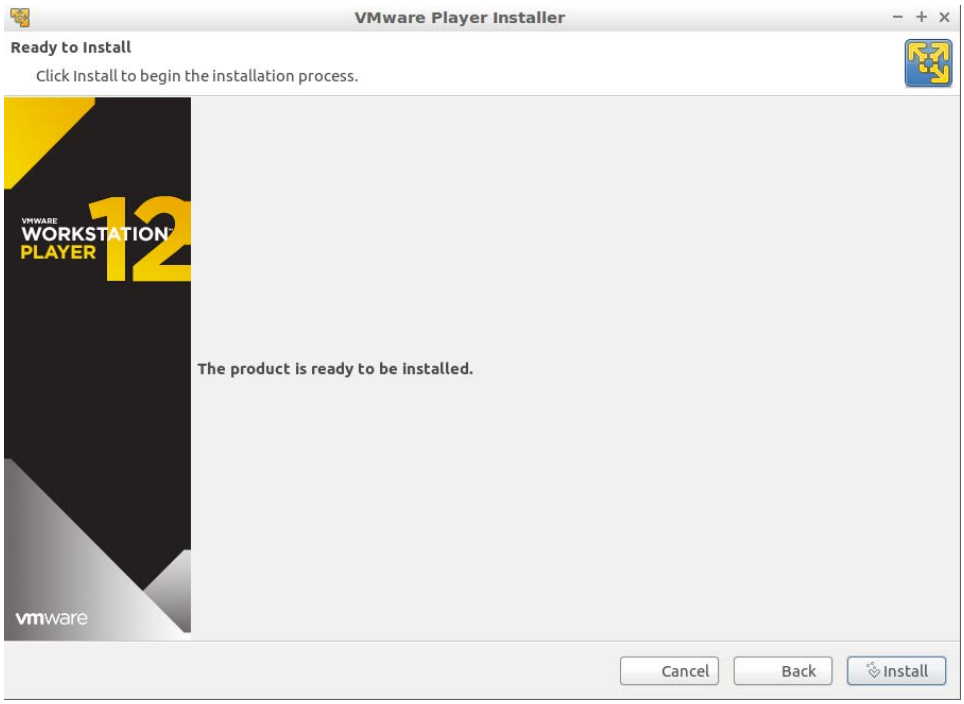


Fig. 98: VMware Player installer screen.

Click “Install” and process begin

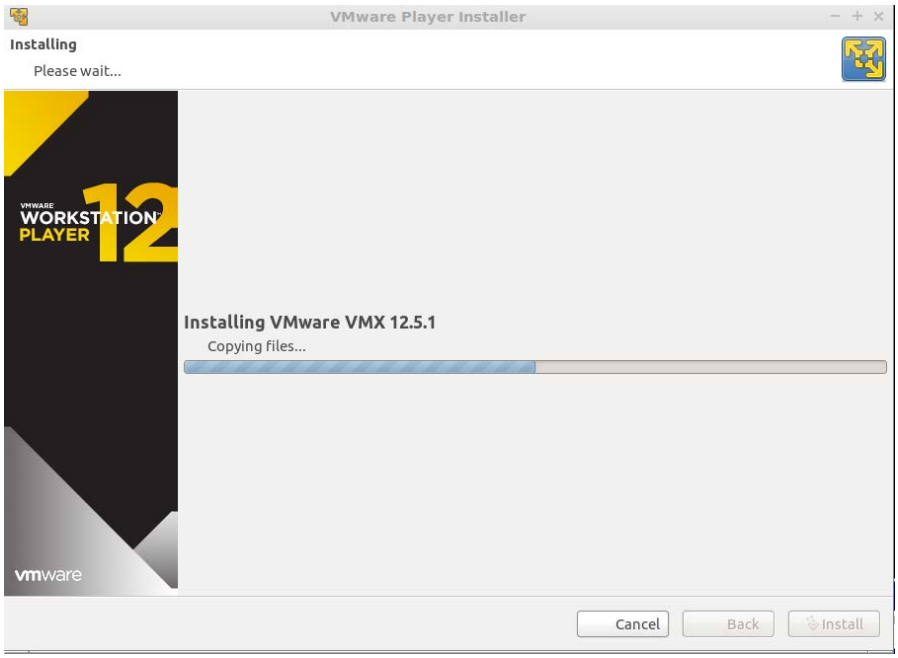


Fig. 99: VMware Player installation screenshot.

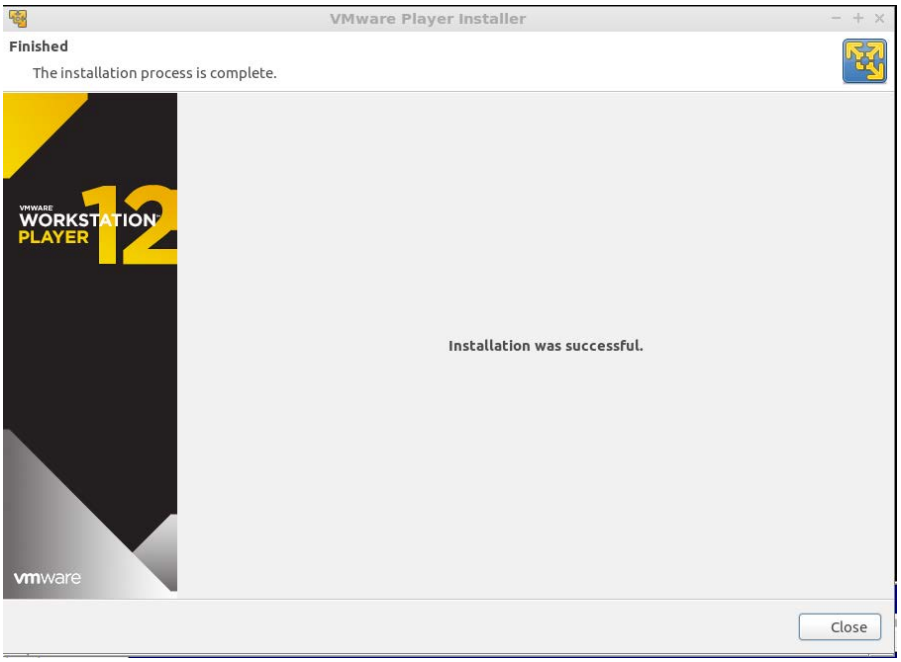


Fig. 100: VMware Player successful installation screen.

Program has been successfully installed, and can be found at “System Tools”
Run it:

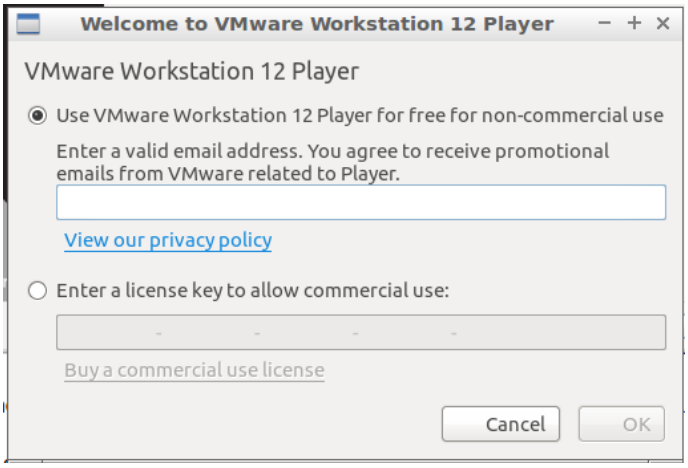


Fig. 101: VMware Player non-commercial use.

An e-mail address is necessary (Fig. 101), type it in and press “OK”

And home screen is shown:

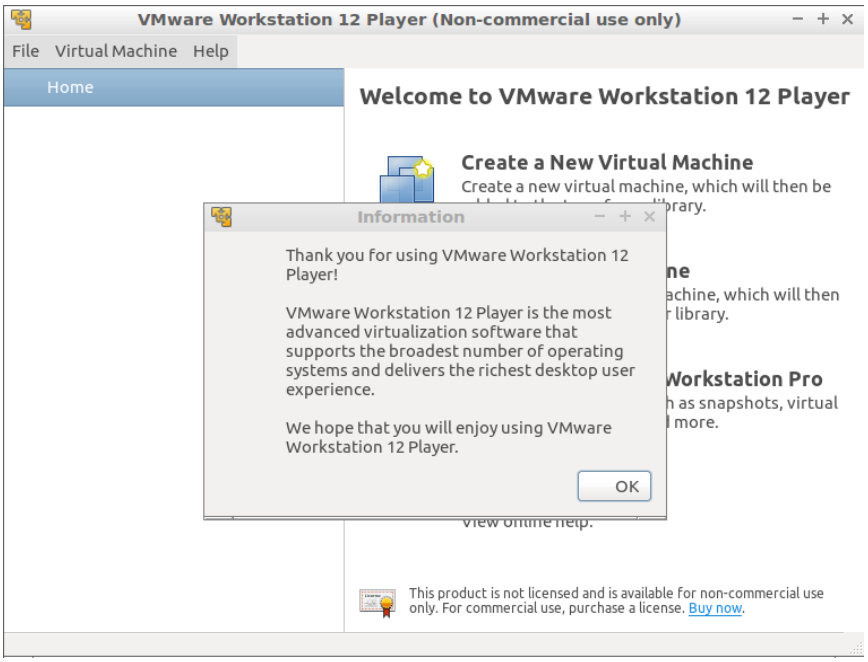


Fig. 102: VMware Player home screen.

If we click “About” option:

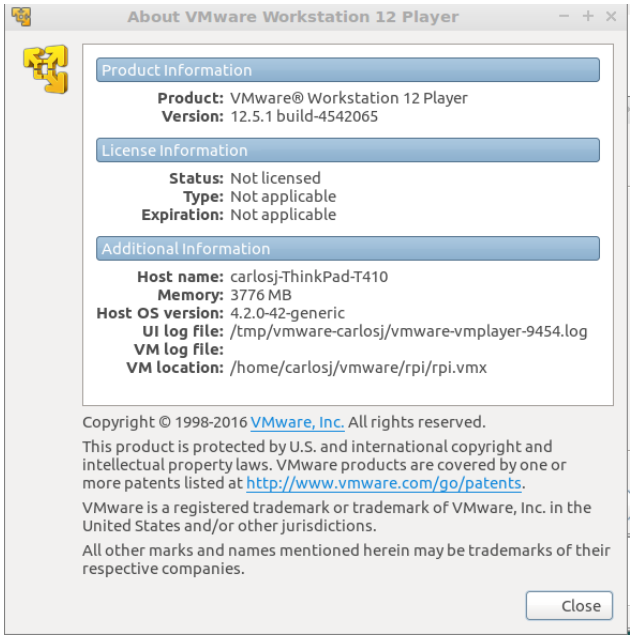


Fig. 103: About VMware Player.

5.3 Creating Ubuntu 14 (Guest) virtual machine

Let’s generate an Ubuntu 14 virtual machine.
First, click on “Create a new virtual machine” (Fig. 104) and a wizard will guide us.

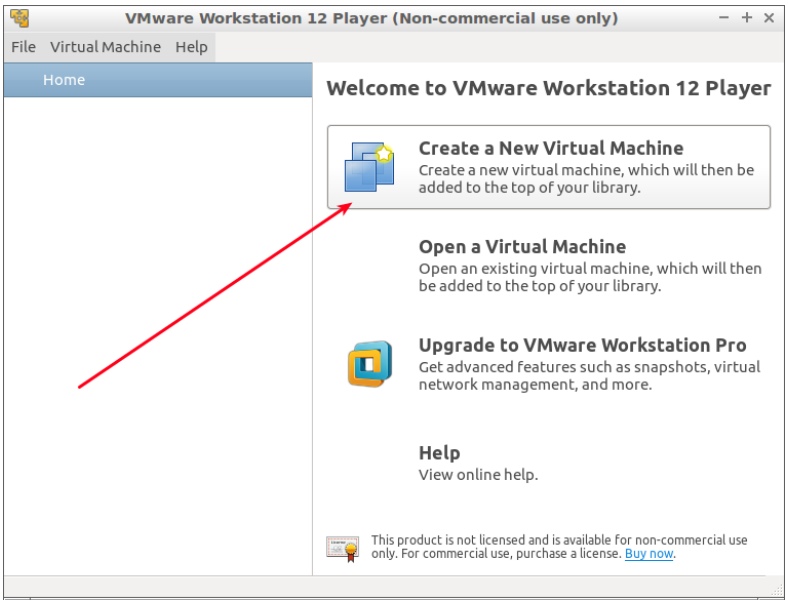


Fig. 104: Creating a virtual machine.

Check (Fig. 105) option “Use ISO image” and “Browse” to locate our downloaded Ubuntu 14 .iso image file.

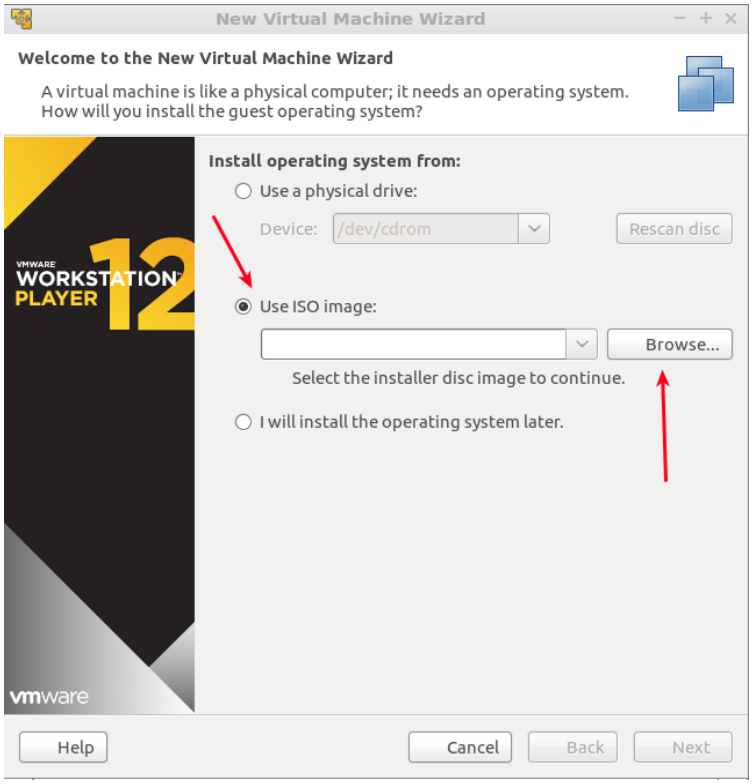


Fig. 105: Browsing to the .iso image file.

Click on the .iso file and then click “Open”

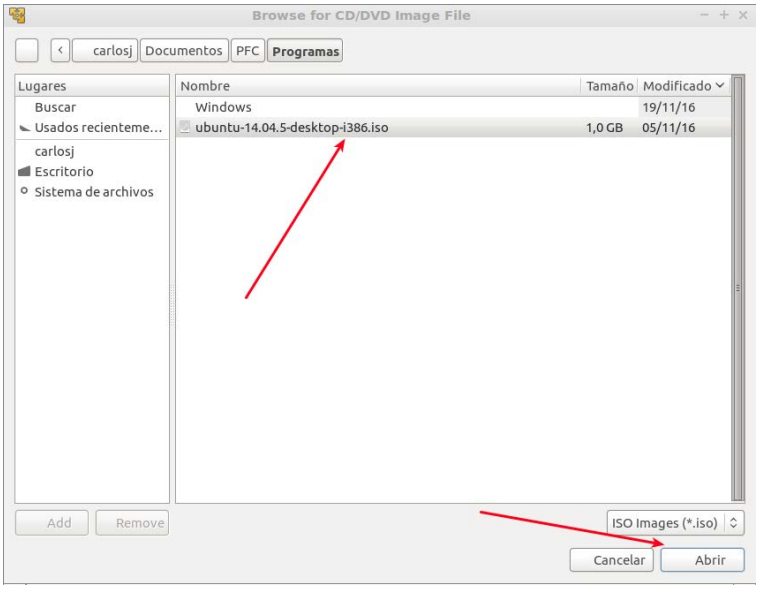


Fig. 106: Selecting ISO image.

Ubuntu 14.04.5 version is detected, click “Next”

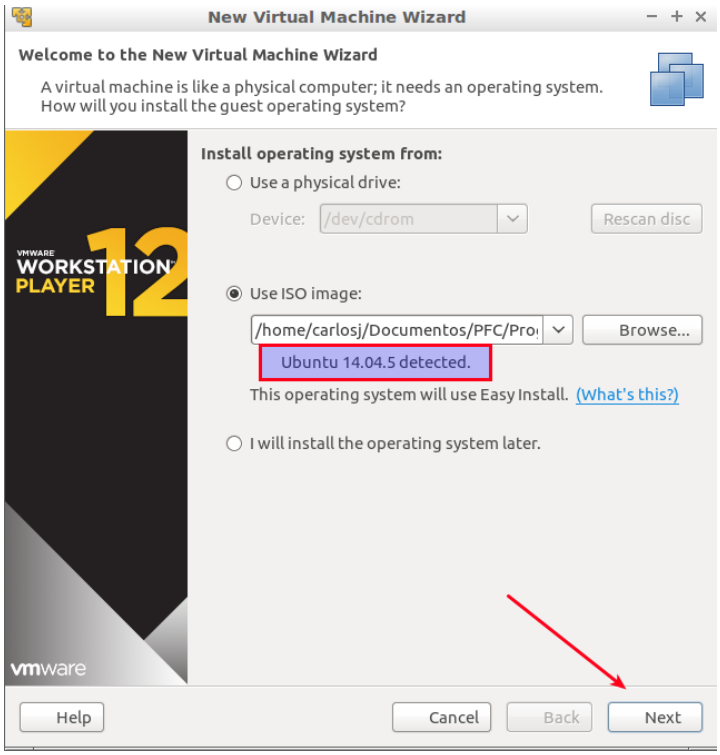


Fig. 107: Automatic detection of OS included in ISO image.

You must choose a user’s name and password:

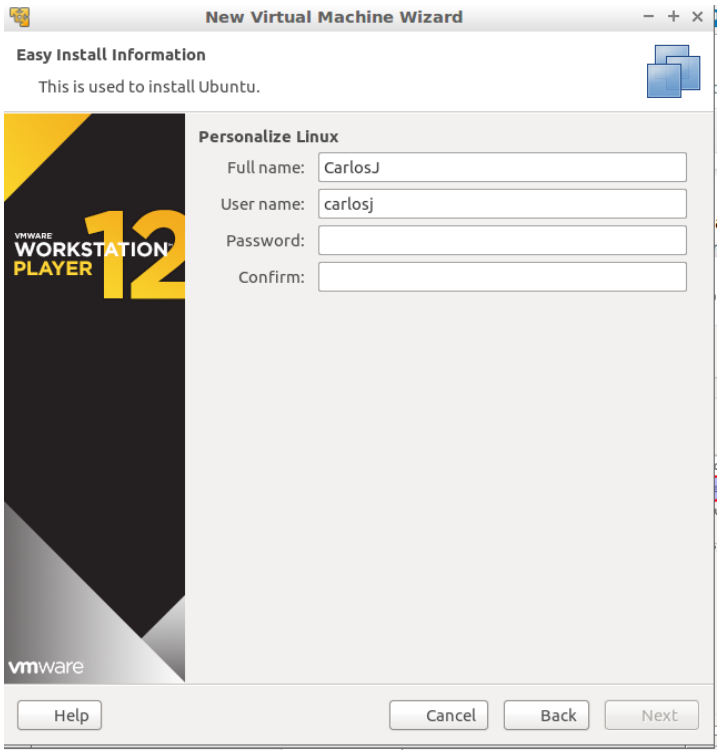


Fig. 108: Defining our Linux user's name and password.

Name the virtual machine and choose a location

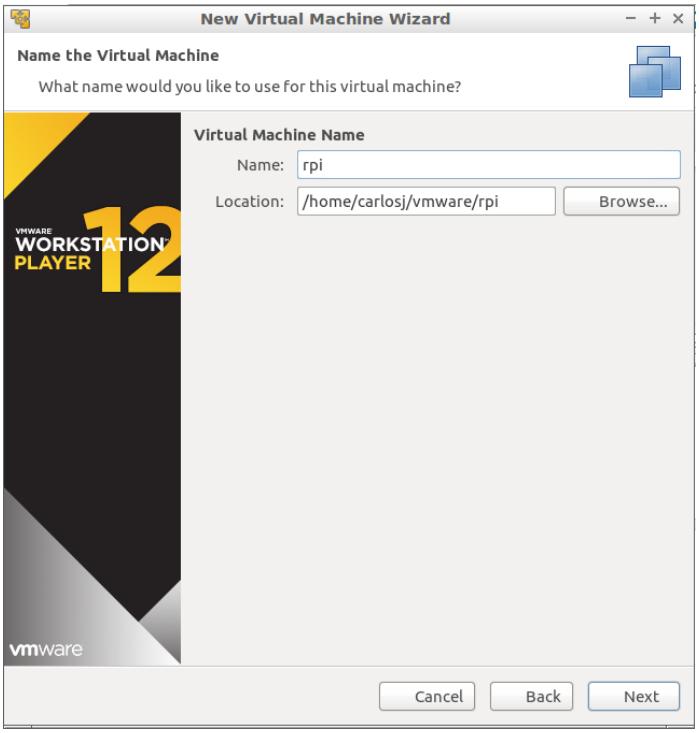


Fig. 109: Virtual machine name and location.

Next step is to specify disk size and how will be stored on the host computer's physical disk. Following directives from [RD1], define 80GB disk size, and select "Split virtual disk into multiple files"

Fig. 110: Specifying virtual machine size.

Installation summary screen

Fig. 111: Virtual machine installation summary screen.

Click on “Finish” and the process begins

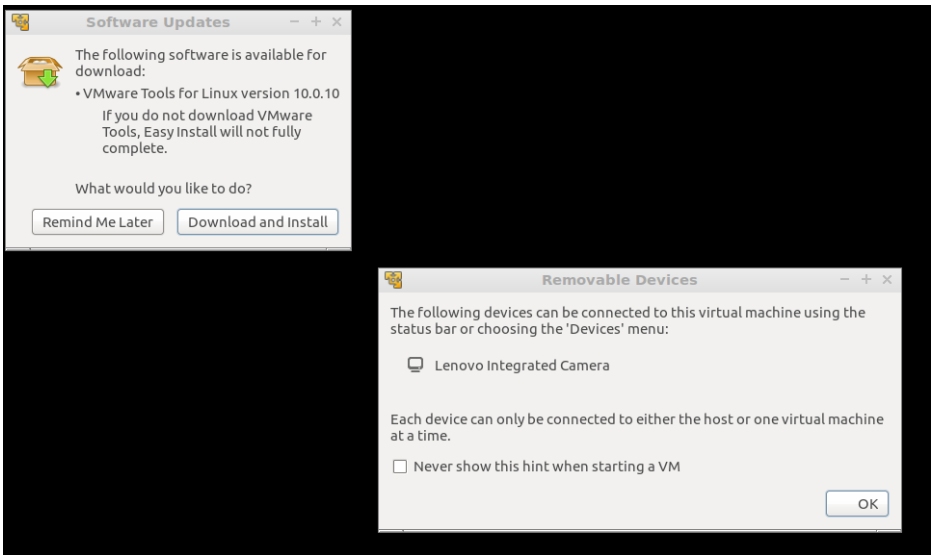


Fig. 112: Virtual machine creation process.

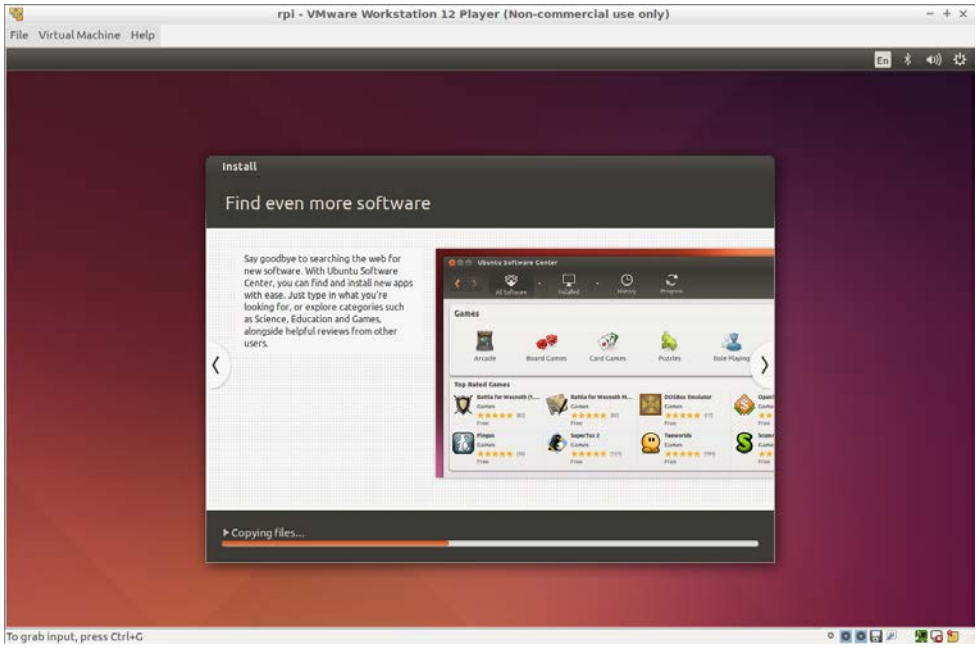


Fig. 113: Ubuntu 14 installation process in virtual machine.

Ubuntu 14 is asking for our password, installation process has finished

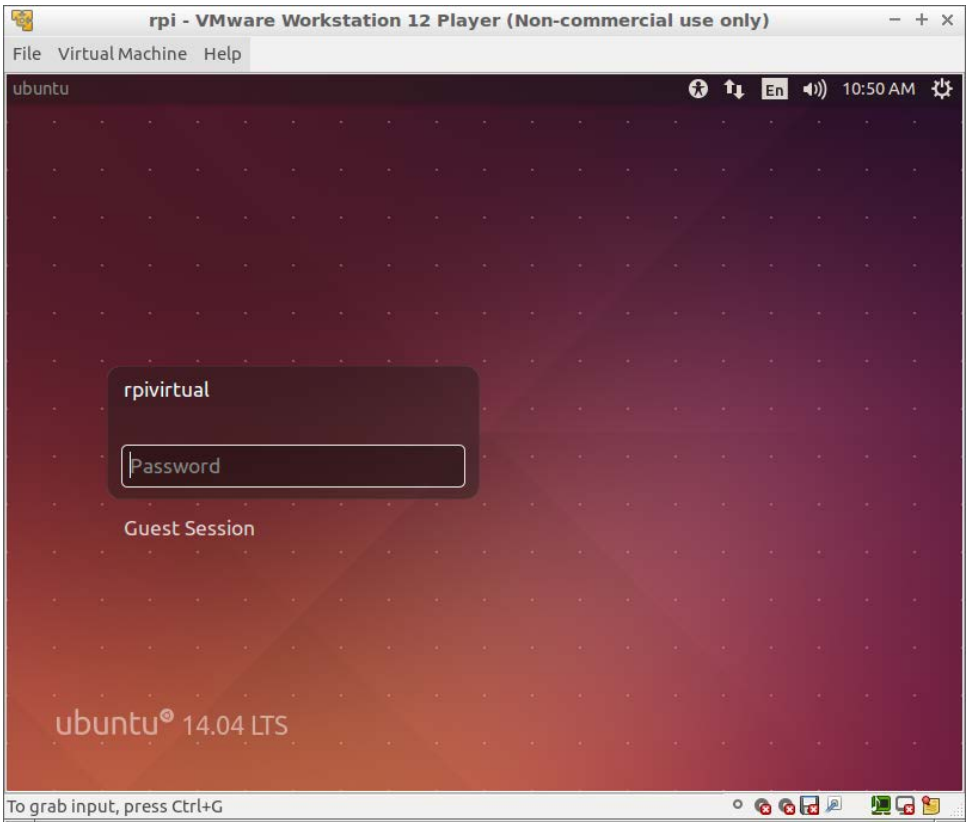
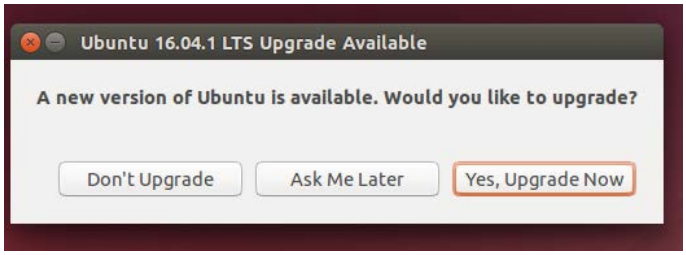


Fig. 114: Ubuntu 14 virtual machine login.

Click on “Don’t upgrade” if the following message appears:



5.4 Running the virtual machine

Run VMware Player, select the name you chose before for your virtual machine (“rpi”), and click “Power On”

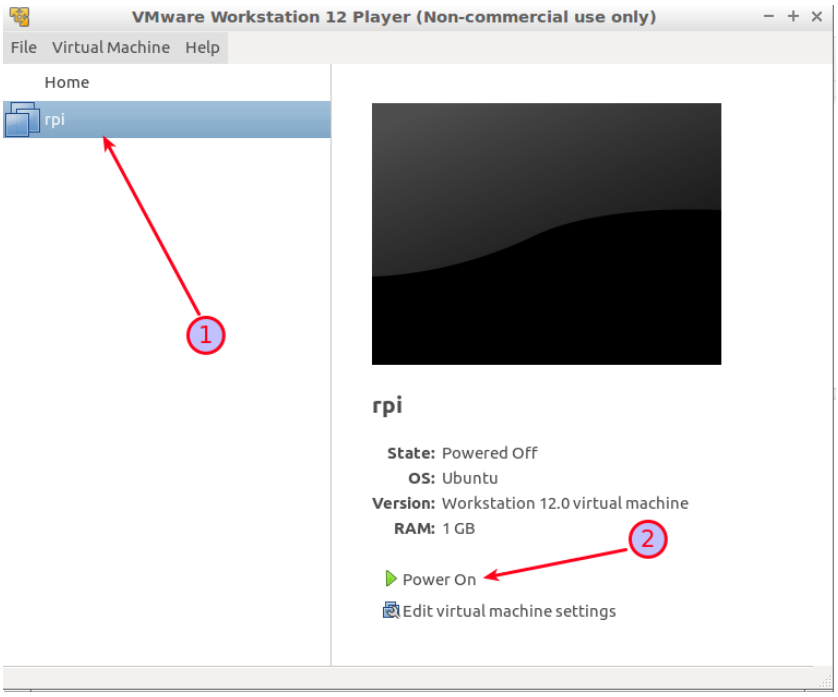


Fig. 115: Selecting virtual machine to run in VMware Player.

It is very important to check for updates in the first place. Execute these two commands from a terminal window:

```
sudo apt-get update
sudo apt-get upgrade
```

Operating system version can be found into “System Settings” -> “Details” (Fig. 116)

Fig. 116: About Ubuntu 14.

5.5 Virtual machine settings

5.5.1 Adjustments to improve performance

For a better performance of the virtual machine, we need to activate virtualization technology **“Intel VT”** in computer’s BIOS, and activate also **“Virtualize Intel VT-x”** option in “Edit virtual machine settings”, “Hardware” tab, “Processors” option.

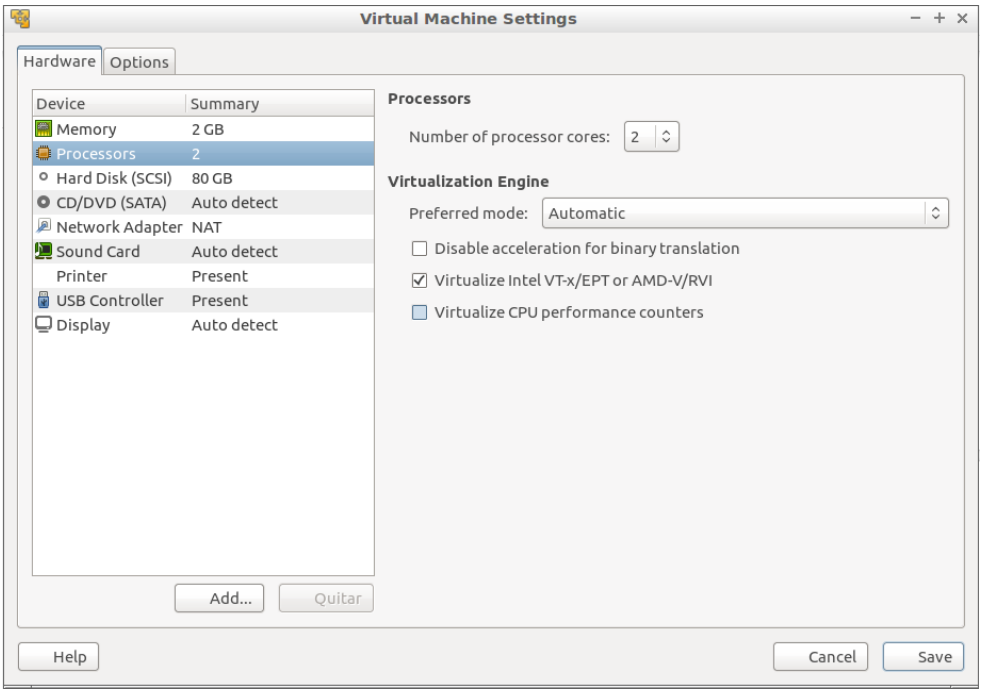


Fig. 117: Virtual machine settings.

Increasing virtual machine memory to a 2GB minimum, and use two or four processors will increase speed too.

5.5.2 Do not ask for password after being idle

To avoid the process of being required a password after some time being idle, go to “System Settings”) and then “Security & Privacy”. Uncheck two options shown in Fig. 118.

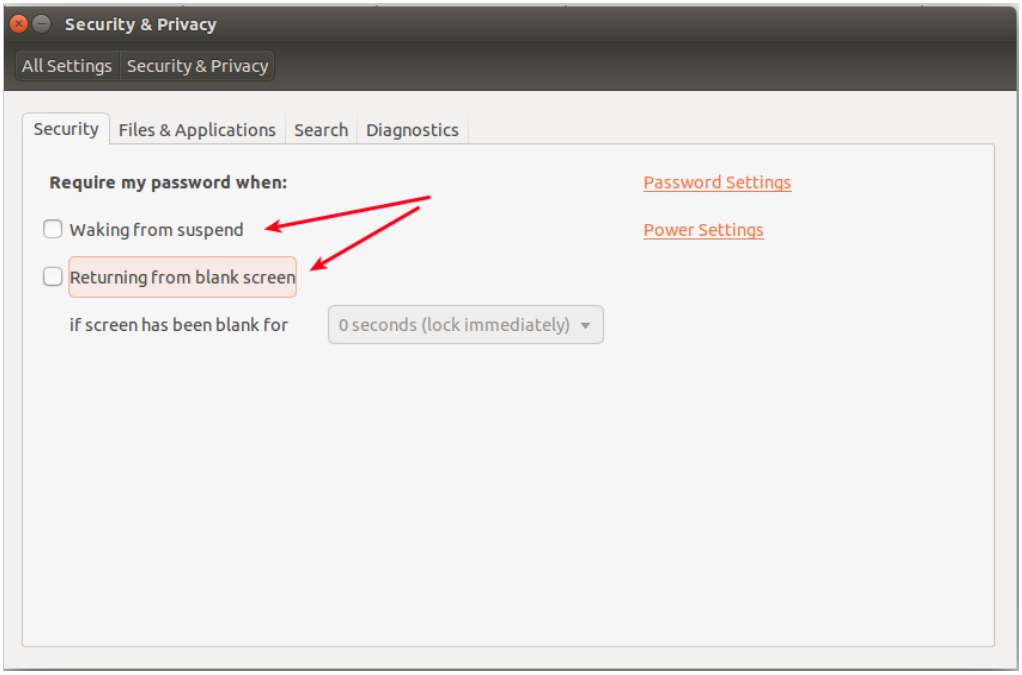


Fig. 118: Removing password after resuming in Ubuntu 14.

5.5.3 Setting time zone

Let's adjust time zone, but language settings will still be English.
Click on the clock of upper right top, and then click "Time and Date Settings".

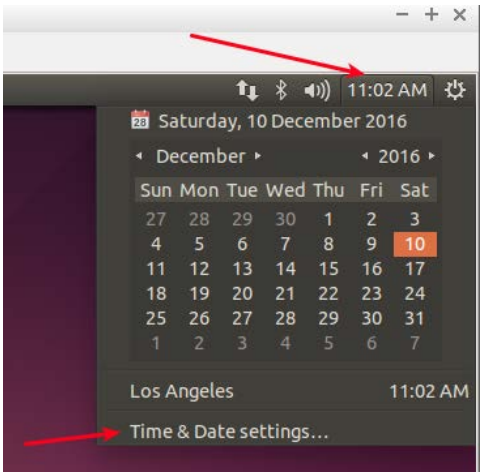


Fig. 119: Setting time zone in Ubuntu 14.

Choose Madrid (or your location) and close the window.

5.5.4 Automatic login without user or password

It could be useful and faster to login automatically into Ubuntu. Click on "System Settings" and then "User Accounts":

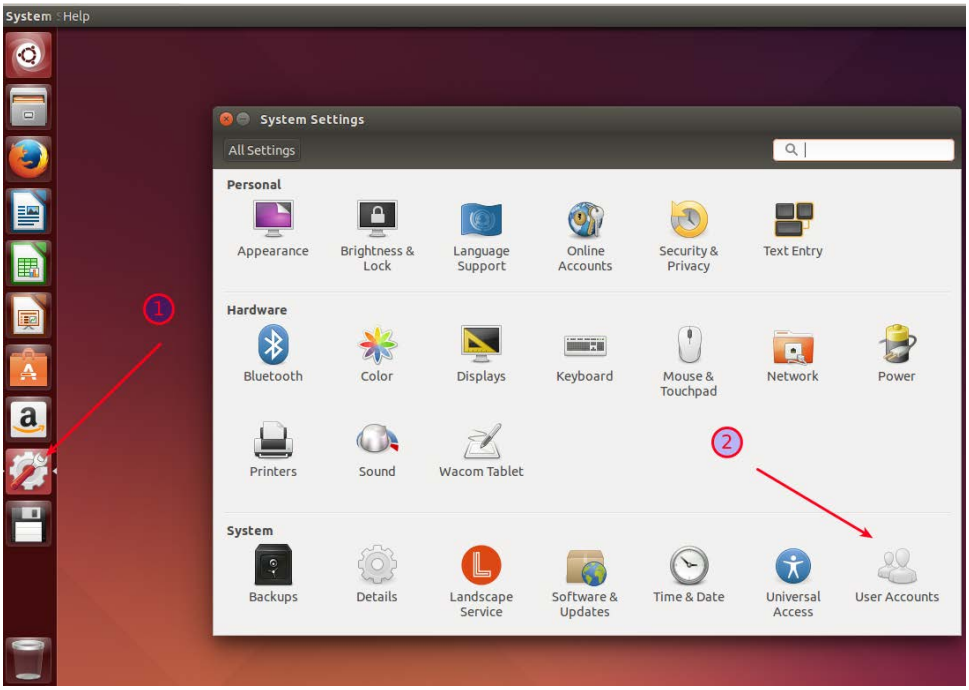


Fig. 120: User account settings in Ubuntu 14.

Unlock and check “Automatic login” option

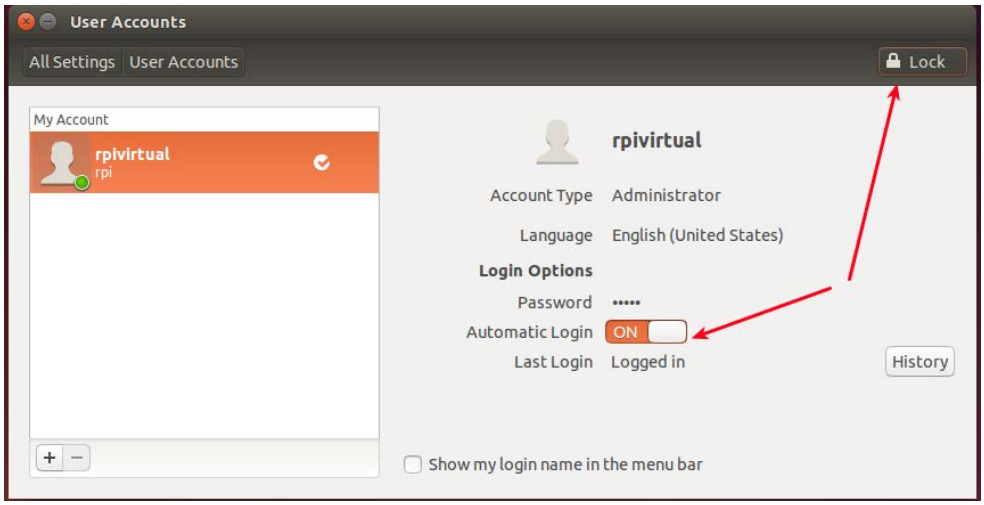


Fig. 121: Automatic login in Ubuntu 14.

Shutdown the virtual machine and check that its size is ~4,4GB.
Virtual machine can be found at */home/carlosj/vmware*, *“rpi”* folder.

5.5.5 Setting Spanish keyboard

It is possible to choose English as system language but use a different keyboard language. To do this, go to “System settings”, then “Keyboard”, click “Text Entry” (Fig. 122), then click on + symbol and add “Spanish” from the list (Fig. 123).

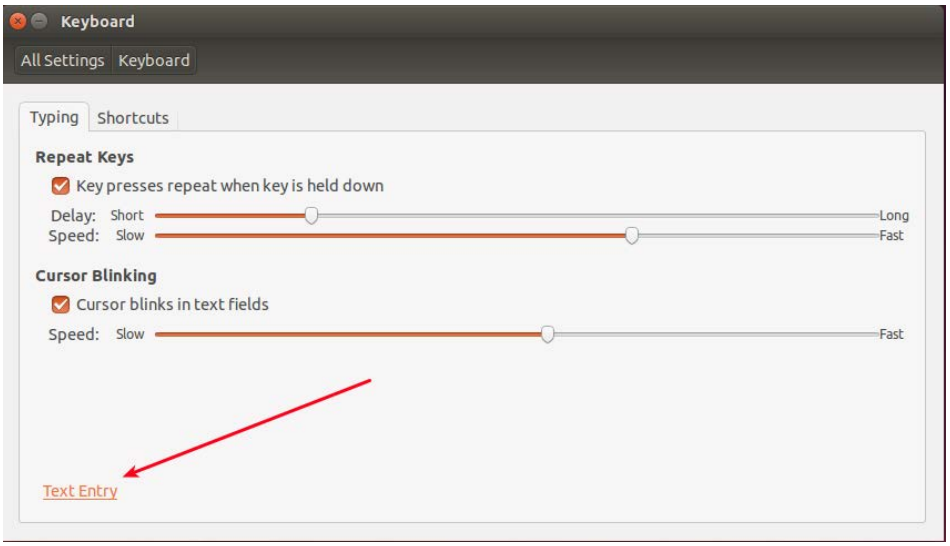
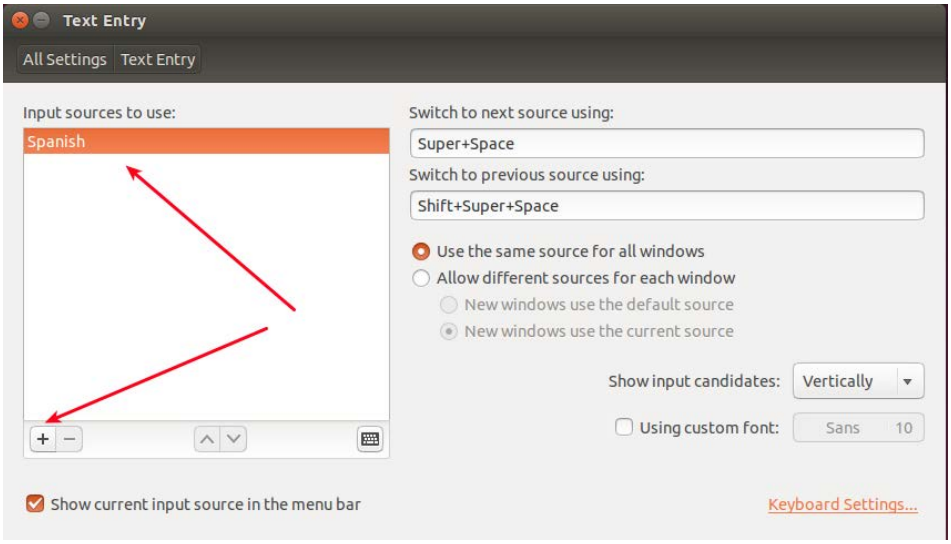


Fig. 122: Keyboard settings in Ubuntu 14.



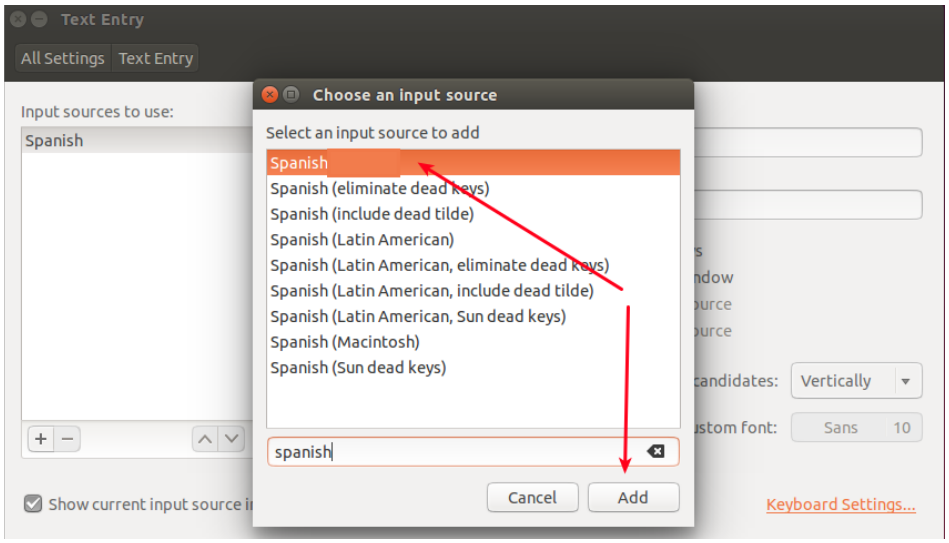


Fig. 123: Adding Spanish keyboard to Ubuntu 14.

Keyboard is now in Spanish, “Es” is shown at the upper right corner of the screen.

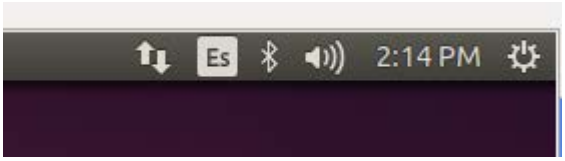


Fig. 124: Keyboard language icon at Ubuntu 14 taskbar.

5.6 VMware Tools and Open VM Tools

Along with other utilities, VMware provide us with VMware Tools, which allow using full screen and a shared clipboard between host and guest. In Linux, it is possible to user another option, Open VM Tools.
https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2073803
In Ubuntu, VMware developers recommend using Open VM Tools instead of VMware Tools:
http://partnerweb.vmware.com/GOSIG/Ubuntu_14_04_LTS.html#Tools

5.7 Installing Buildroot required packages

First we need to update the operating system:

```
sudo apt-get update
sudo apt-get upgrade
```

We have checked, using Synaptic and looking package by package, if Ubuntu 14 default installation includes the packages that Buildroot needs:
https://buildroot.org/downloads/manual/manual.html#requirement

Annex 12 in this document contains the full list of packages whose installation is needed.

Build tools:

- *which* (INSTALLED)
- *sed* (INSTALLED, 4.2.2-4)
- *make* (version 3.81 or later) = INSTALLED, 3.81-8.2
- *binutils* (INSTALLED, 2.24-5)
- *build-essential* (only for Debian based systems) = **NOT INSTALLED,** 11.6
- *gcc* (version 2.95 or any later) = INSTALLED, 4.8.2
- *g++* (version 2.95 or any later) = **NOT INSTALLED,** 4.8.2
- *bash* (INSTALLED, 4.3-7)
- *patch* (INSTALLED, 2.7.1-4)
- gzip (INSTALLED, 1.6-3)
- bzip2 (INSTALLED, 1.0.6-5)
- perl (version 5.8.7 or any later) = (INSTALLED, 5.18.2-2)
- tar (INSTALLED, 1.27.1-1)
- cpio (INSTALLED, 2.11)
- python (version 2.6 or any later) = (INSTALLED, 2.7.5-5)

- unzip (INSTALLED, 6.0-9)
- rsync (INSTALLED, 3.1.0-2)

Source fetching tools:

- wget (INSTALLED, 1.15-1)

Optional packages:
Configuration interface dependencies: (runtime and development libraries needed)

- ncurses5 (INSTALLED, libncurses5 5.9)
- qt4 (**NOT INSTALLED,** 4.8.5)

Source fetching tools:

- git (**NOT INSTALLED,** 1.9.1)

Graph generation tools:

- graphviz (**NOT INSTALLED,** 2.36.0)
- python-matplotlib (**NOT INSTALLED,** 1.3.1-1)

Install all packages listed in Annex 1 of [RD1]

```
sudo apt-get install g++
```

40MB, g++ version 4.8.2-1 installed

```
sudo apt-get install dselect
```

Version 1.17.5 installed

```
sudo apt-get install git
```

Version 1.9.1 installed

```
sudo apt-get install gdbserver
```

Version 7.7.1 installed

```
sudo apt-get install uboot-mkimage
```

Installation error, obsolete package warning. Install “u-boot-tools” instead.

```
sudo apt-get install u-boot-tools
```

Version 2013.10-3 installed

```
sudo apt-get install qt3-dev-tools
```

Installation error. This package is no longer available (http://askubuntu.com/questions/386254/how-to-install-qt3-dev-tools-package), install qt4-dev-tools instead.

```
sudo apt-get install qt4-dev-tools
```

105MB download, version 4.8.5 installed

```
sudo apt-get install qt4-qmake
```

(Not needed, dev-tools already install it)

```
sudo apt-get install eclipse
```

244MB download, version 3.8.1-5.1 installed

```
sudo apt-get install eclipse-cdt
```

31MB download, version 8.3.0-1 installed

```
sudo apt-get install gparted
```

Version 0.18.0-1 installed

```
sudo apt-get install putty
```

Version 0.63-4 installed

```
sudo apt-get install nautilus-open-terminal
```

Installed OK

After all this installation process, virtual machine size is 6.3GB.

We will also install *ncurses5* for using menuconfig interface, *graphviz* for graph-depends and *python-matplotlib* for graph-build. “ncurses5” can not be located, but “ncurse” and “lincurse5” are already installed.

We can install Synaptic package manager too:

```
sudo apt-get install synaptic
```

Install ncurses-dev:

```
sudo apt-get install ncurses-dev
```

Install graphviz version 2.36.0:

```
sudo apt-get install graphviz
```

Install python-matplotlib package, version 1.3.1-1:

```
sudo apt-get python-matplotlib
```

Buildroot requirements can be found at Buildroot manual:
https://buildroot.org/downloads/manual/manual.html#requirement

5.8 About Buildroot

Previously, we used a full Linux distribution (Raspbian) to install EPICS in it. However, these distributions include a lot of packages that maybe we will not need, they are slower to boot and consume more resources. Keep in mind that embedded systems usually are very limited in terms of memory and processor capabilities.
Therefore, we will generate our own custom Linux distro, to install and run EPICS in it.
To do this, we are going to use Buildroot (https://buildroot.org/), a tool designed to create Linux oriented embedded systems in a relatively simple way.

Fig. 125: Buildroot logo.

We strongly recommend reading “Embedded Linux system development” [RD2] that details all aspects of embedded systems and the different tools that can be used to generate them.

Buildroot allow us to generate a custom Linux distribution with all the necessary elements, such as the file system, OS kernel, bootloader and toolchain, as shown in Fig. 126.

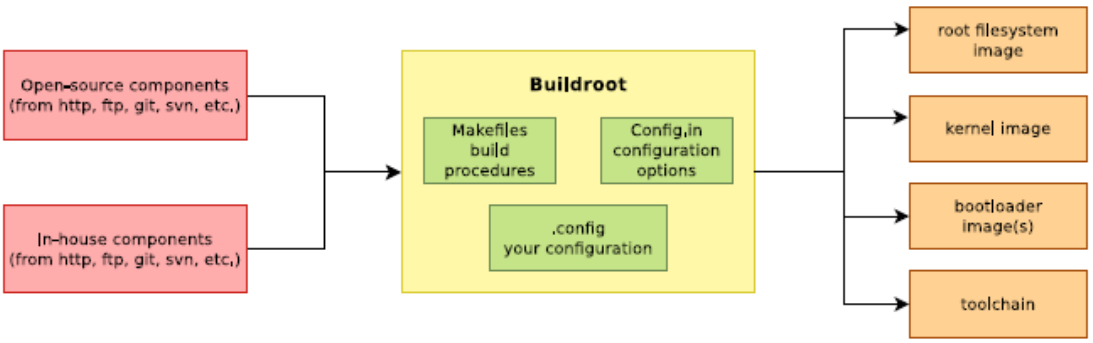


Fig. 126: Buildroot operation scheme (from http://free-electrons.com/training/).

Buildroot allow us to generate a cross-compilation environment (**Cross build**, Fig. 127). That means that development and compilation of the code will be done on a Linux machine with a x86 microprocessor (our Ubuntu 14 virtual machine), but the code will run in a machine with an ARM microprocessor (Raspberry Pi).

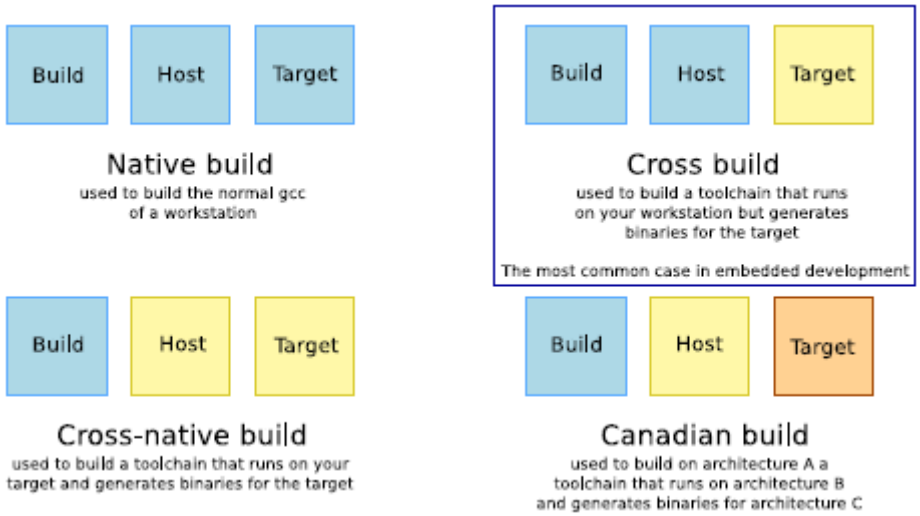


Fig. 127: Configurations for applications development in embedded systems (from http://free-electrons.com/training/).

5.9 Installing Buildroot in Ubuntu virtual machine

Buildroot 2016.11 version is available to download from Buildroot website https://buildroot.org/download.html Link is shown in Fig. 128:
https://buildroot.org/downloads/buildroot-2016.11.tar.gz

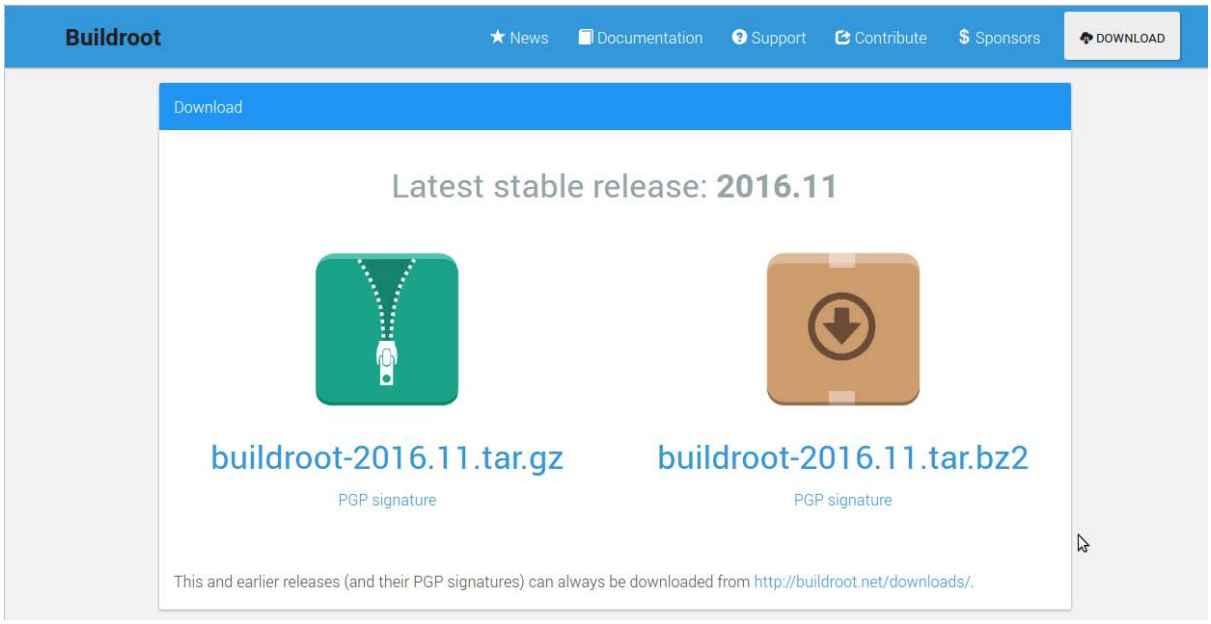


Fig. 128: Buildroot download web page.

Earlier releases can be found at https://buildroot.org/downloads/

buildroot-2016.08.tar.bz2	2016-09-01 09:38	4.8M
buildroot-2016.08.tar.bz2.sign	2016-09-01 09:38	535
buildroot-2016.08.tar.gz	2016-09-01 09:38	5.7M
buildroot-2016.08.tar.gz.sign	2016-09-01 09:38	532
buildroot-2016.11-rc1.tar.bz2	2016-11-03 22:41	4.7M
buildroot-2016.11-rc1.tar.bz2.sign	2016-11-03 22:41	1.2K
buildroot-2016.11-rc1.tar.gz	2016-11-03 22:41	5.5M
buildroot-2016.11-rc1.tar.gz.sign	2016-11-03 22:41	1.2K
buildroot-2016.11-rc2.tar.bz2	2016-11-13 19:59	4.2M
buildroot-2016.11-rc2.tar.bz2.sign	2016-11-13 19:59	1.2K
buildroot-2016.11-rc2.tar.gz	2016-11-13 20:00	5.0M
buildroot-2016.11-rc2.tar.gz.sign	2016-11-13 20:00	1.2K
buildroot-2016.11-rc3.tar.bz2	2016-11-28 23:17	4.7M
buildroot-2016.11-rc3.tar.bz2.sign	2016-11-28 23:17	547
buildroot-2016.11-rc3.tar.gz	2016-11-28 23:17	5.5M
buildroot-2016.11-rc3.tar.gz.sign	2016-11-28 23:17	544
buildroot-2016.11.tar.bz2	2016-11-30 22:19	4.7M
buildroot-2016.11.tar.bz2.sign	2016-11-30 22:19	535
buildroot-2016.11.tar.gz	2016-11-30 22:19	5.5M
buildroot-2016.11.tar.gz.sign	2016-11-30 22:20	532
buildroot-snapshot.tar.bz2	2016-12-12 00:15	3.9M
buildroot.html	2013-01-23 15:25	258
eclipse/	2014-09-21 09:45	-
manual/	2016-11-06 21:58	-
old/	2006-04-13 17:34	-
snapshots/	2016-12-12 00:15	-

Fig. 129: Buildroot earlier releases web page.

Run virtual machine from VMware Player:

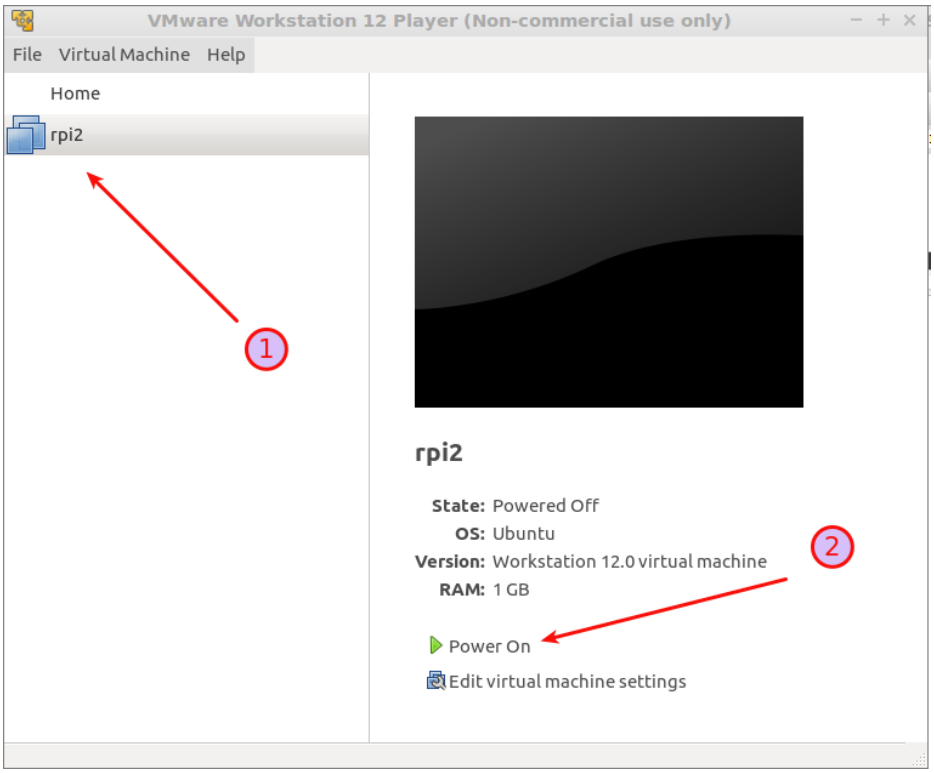


Fig. 130: Starting Ubuntu virtual machine in VMware Player.

Click on Firefox web browser

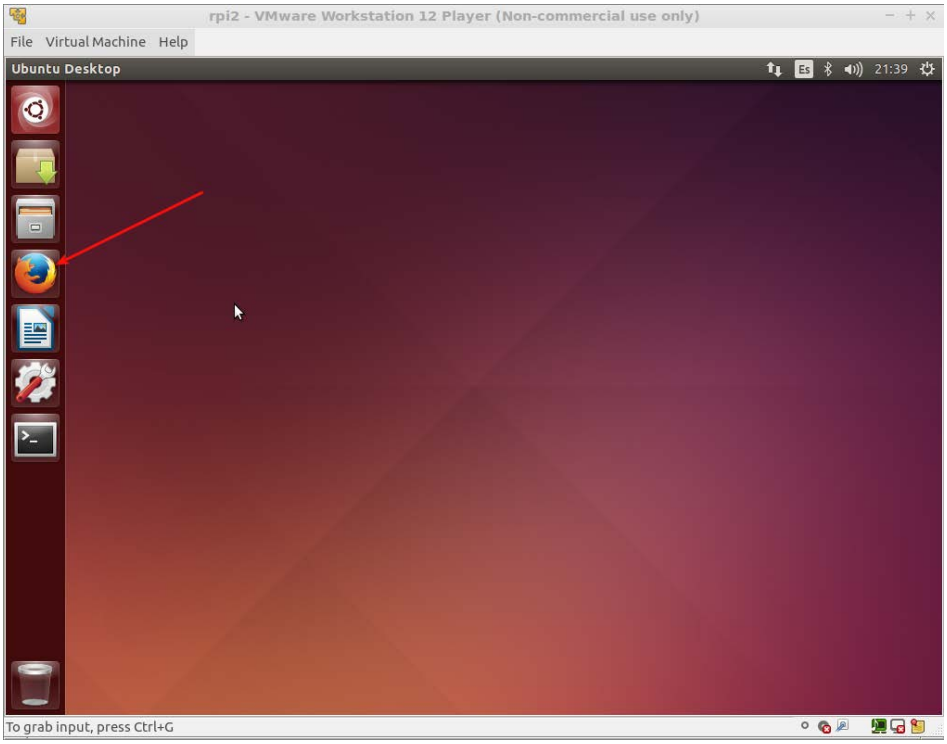


Fig. 131: Firefox web browser in Ubuntu.

Browse to web page https://buildroot.org/download.html

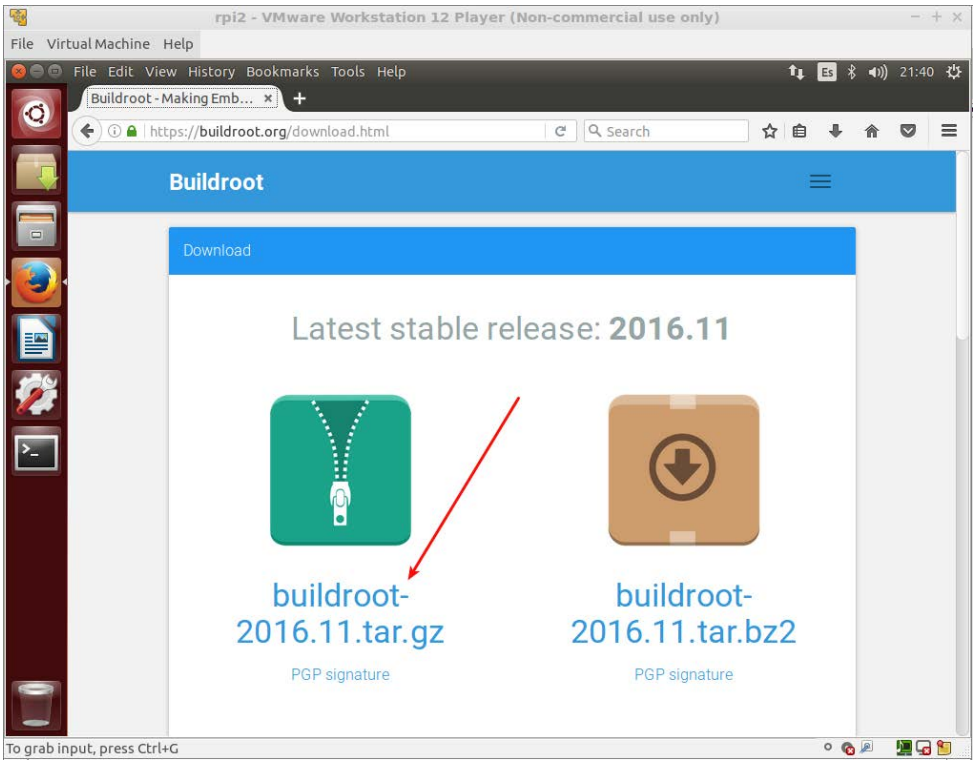


Fig. 132: Buildroot download web page.

Click on “buildroot-2016.11.tar.gz”, choose “Save File” option (Fig. 133), to download file to “Downloads” folder.

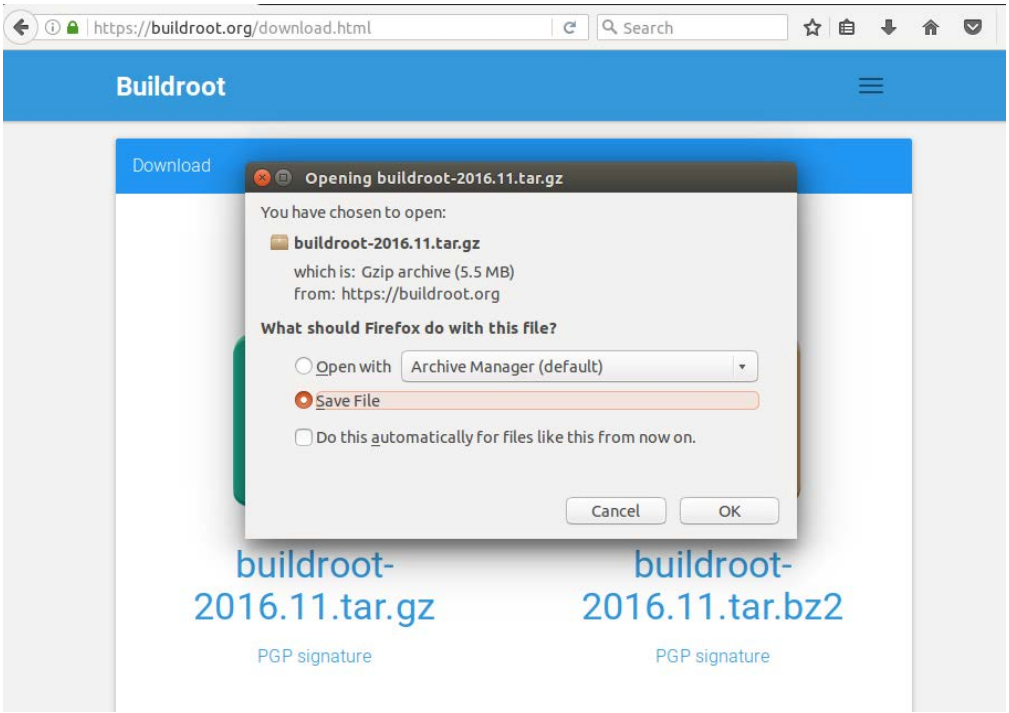


Fig. 133: Saving Buildroot installation file to a folder.

Now copy the file from “Downloads” folder to “Documents” folder, right click and select “Extract here”

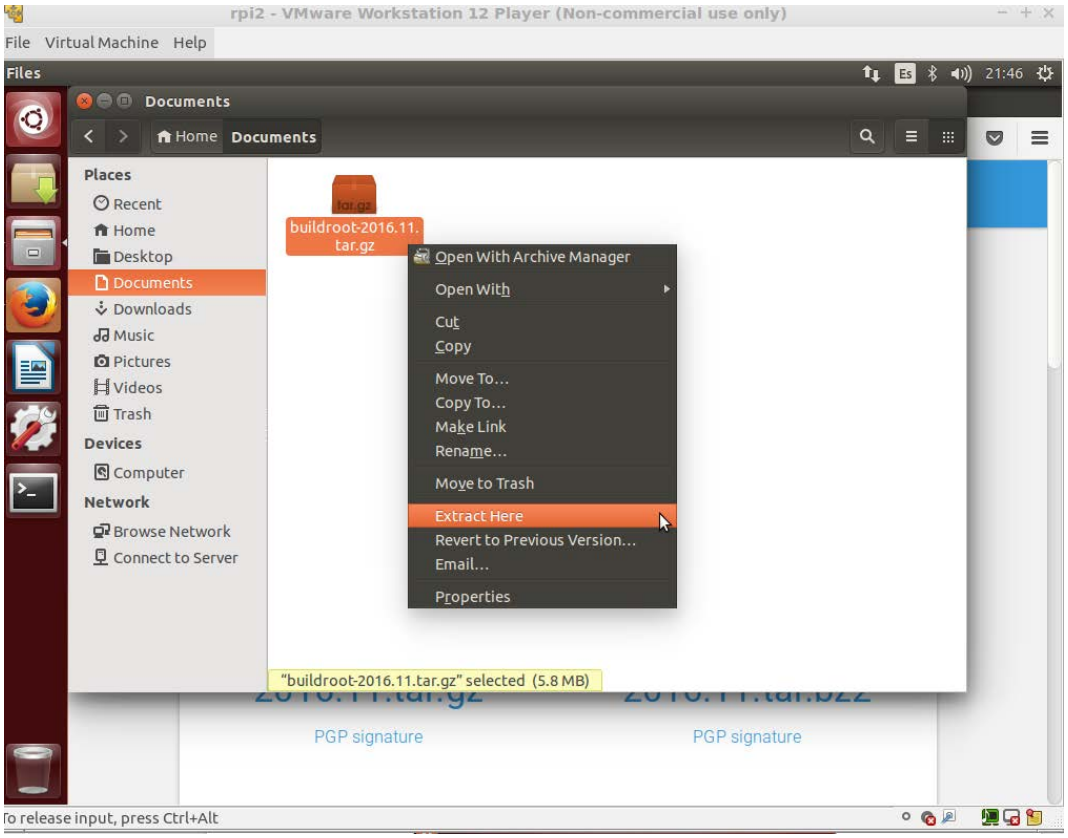


Fig. 134: Extracting Buildroot from installation file.

That creates a folder named “buildroot-2016.11” inside “Documents”

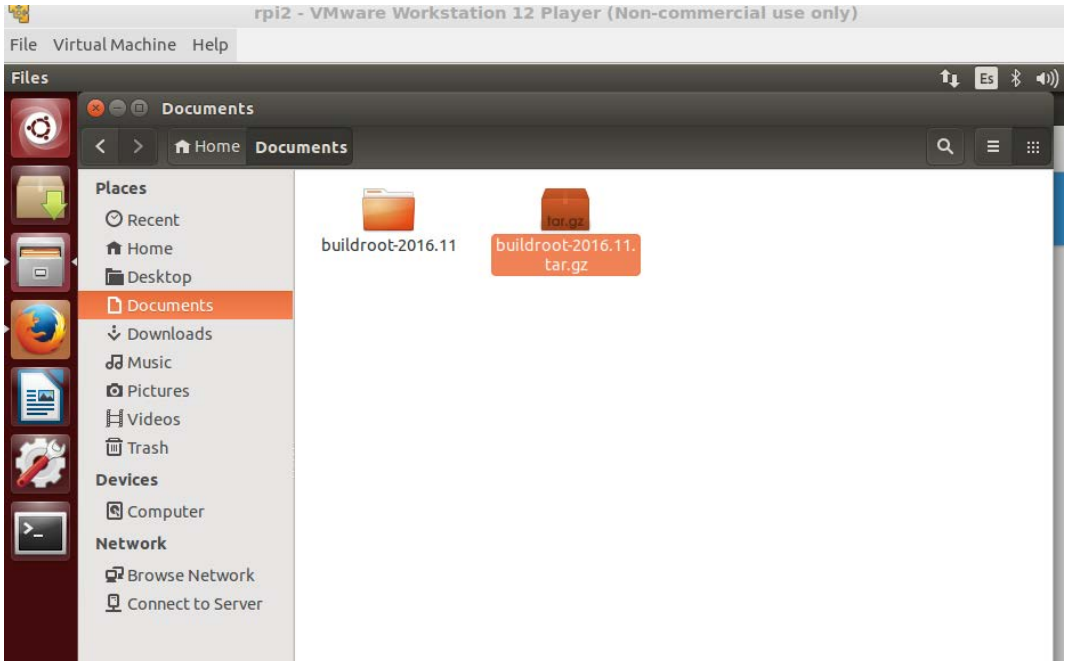


Fig. 135: New folder after Buildroot unpacking process.

Right click on a free area, and select “Open in Terminal”
Type these commands:

```
cd buildroot-2016.11
make xconfig
```

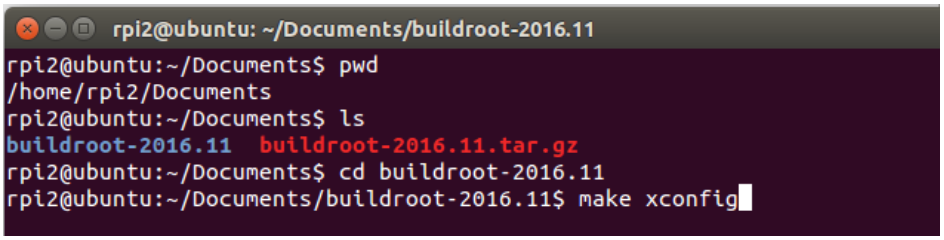


Fig. 136: First Buildroot run.

Buildroot graphical environment is shown (Fig. 137), but a text mode configuration window is available through “make menuconfig” command.

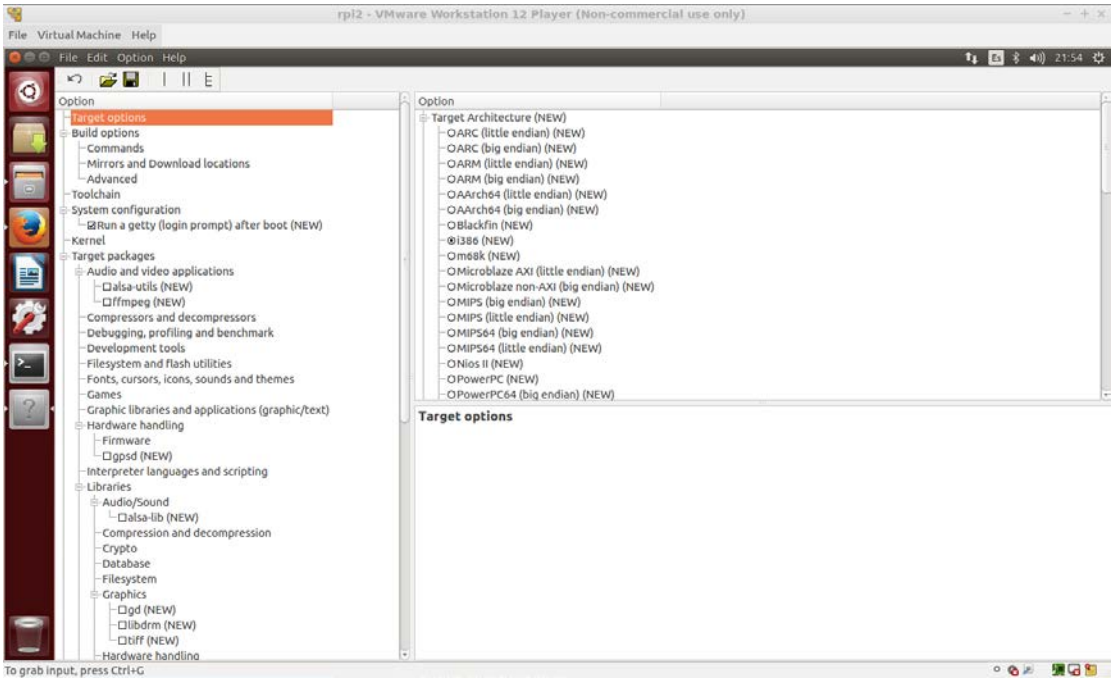


Fig. 137: Buildroot graphical environment.

6 GENERATING A LINUX OPERATING SYSTEM USING BUILDROOT

6.1 Default Buildroot settings for Raspberry Pi 3

Once we have begun Buildroot configuration, either in graphical mode ("make xconfig") or text mode ("make menuconfig"), it is necessary to configure the different sections. We will have to navigate through the various menus and select items to install or include in our resulting Linux image.
Buildroot configuration is an iterative process, in order to generate our embedded Linux system we will need to run the configuration several times.
We will use basic settings provided by Buildroot developers as starting point. These settings contain the minimum elements required to obtain a functional embedded Linux.

"buildroot-2016.11" folder contains another folder called "configs", a lot of default settings for different systems are available there. There are four files in particular referring to Raspberry different models:

- "raspberrypi0_defconfig"
- "raspberrypi2_defconfig"
- "raspberrypi_defconfig"
- "raspberrypi3_defconfig"

We are going to use last file to test a default configuration for our Raspberry Pi 3.

Execute commands:

```
cd ~/Documents/buildroot-2016.11
make raspberrypi3_defconfig
```

After a little while, result is shown (Fig. 138). A hidden file ".config" is created into "buildroot-2016.11" folder.

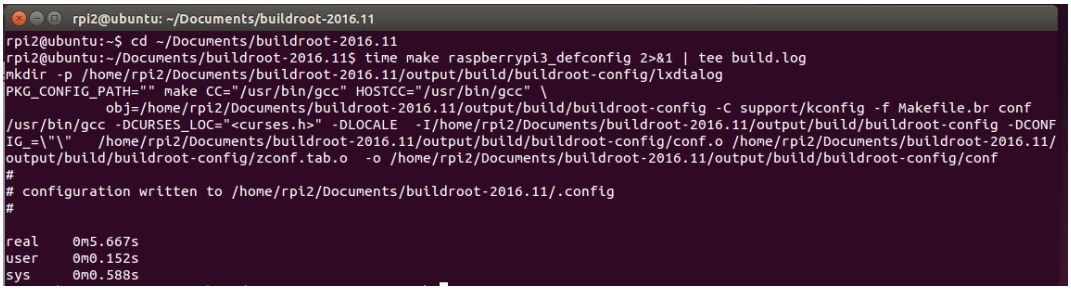


Fig. 138: Using Buildroot defaults settings for Raspberry Pi 3.

Compilation process starts by executing *make*.

```
make
```

[Help]: If you want to know how long does it take to compile, and you want to have a log with the resulting messages in a file called "build.log", the following command will do the job: ***time make 2>&1 | tee build.log***

Building process takes much longer, especially downloading kernel headers (1.5 GB). It will take as much as 2 hours or more.

The following lines were noticed in the final stage of the build process:

```
"<<< Executing post-image script board/raspberrypi3/post-image.sh"
"Adding 'dtoverlay=pi3-miniuart-bt' to config.txt (fixes ttyAMA0 serial
console)."
```

This applies a patch which should allow us to connect the USB FTDI cable without further issues (remember rpi 3 problem related to Bluetooth stack and the serial port).

"buildroot" folder size is now 4.4GB (26MB before building process).

These files have been generated in "Output/Images" folder (200MB):

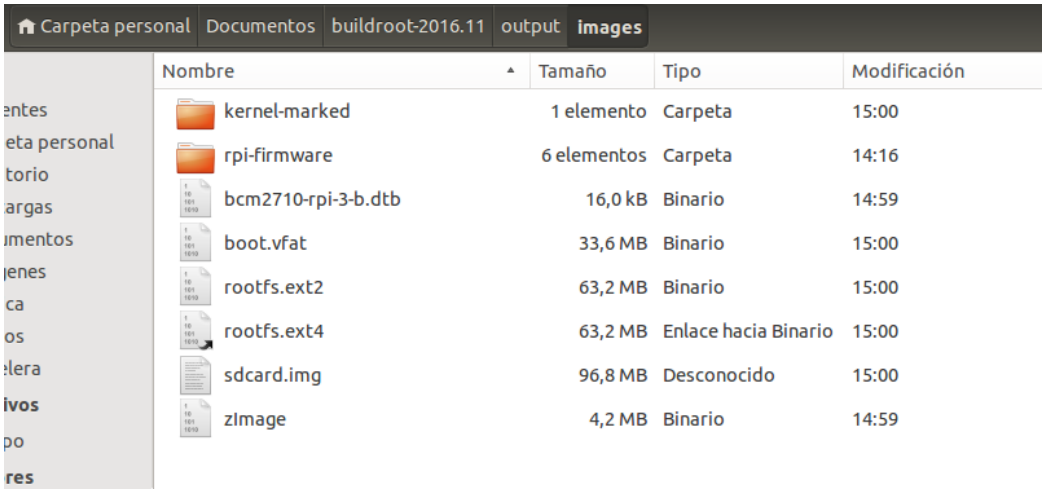


Fig. 139: Folder structure after first build.

Folder "buildroot-2016.11/dl" (Fig. 140) holds all elements downloaded during build process, its size is 400MB.

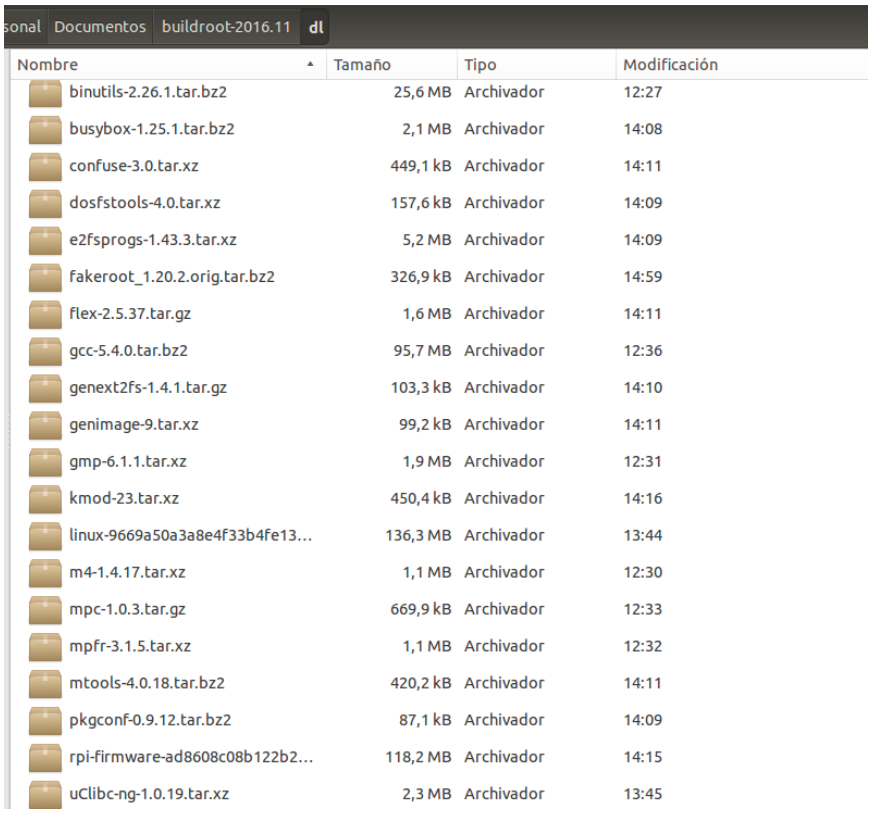


Fig. 140: Buildroot download folder.

Next step is copy the generated image to microsd card. Locate “buildroot-2016.11” folder.

We will use a microsd card already formatted (see http://lanzarduino.beautifullcode.com/formatear-una-sd-para-raspberry-pi/), and recognized by the system as “sdb” (use “dmesg” command after connecting it to find out if you have any doubts). Copy the resulting image of our Linux embedded system (“sdcard.img” file) using these commands:

```
sudo umount /dev/sdb
sudo dd if=output/images/sdcard.img of=/dev/sdb
```

```
rpi2@ubuntu:~/Documents/buildroot-2016.11$ sudo dd if=output/images/sdcard.img of=/dev/sdb
188993+0 records in
188993+0 records out
96764416 bytes (97 MB) copied, 68.3618 s, 1.4 MB/s
rpi2@ubuntu:~/Documents/buildroot-2016.11$
```

Fig. 141: Copying image created using Buildroot to microsd card.

As a result, the following structure is created into microsd card:

- Partition 1 (34MB size volume):

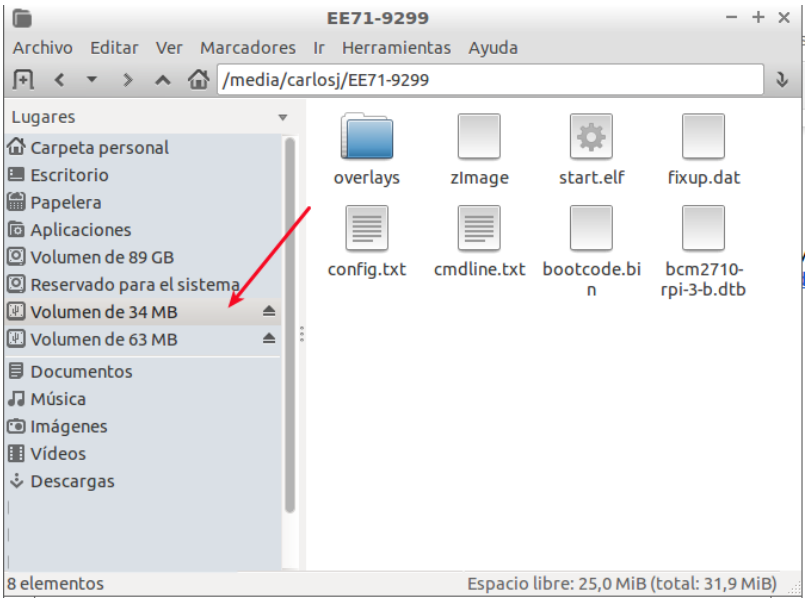


Fig. 142: Structure of the first partition into microsd card.

- Partition 2 (63MB size volume)

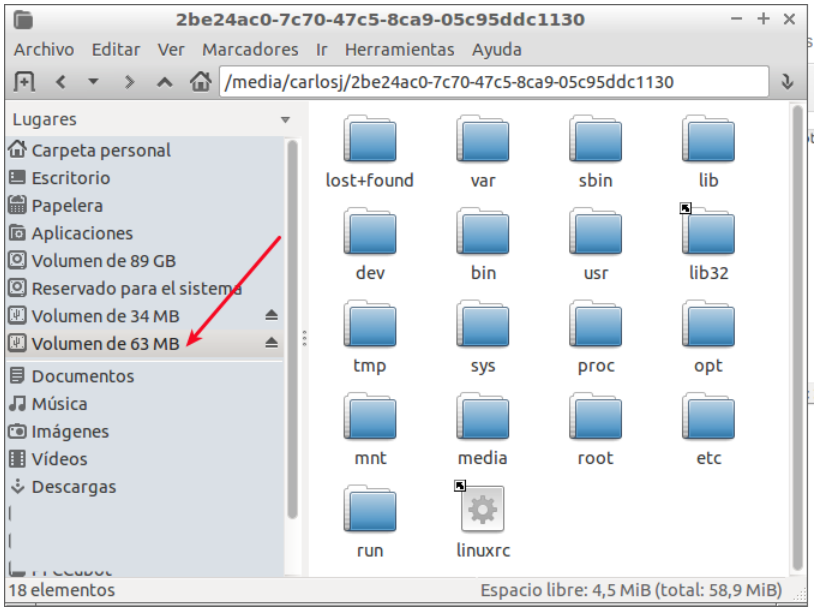


Fig. 143: Structure of the second partition into microsd card.

[Help]: If you want to access the microsd card from the virtual machine, you need to click VMware Player upper menu, “Virtual Machine”, “Removable Devices”, find your card reader, and click “Connect” (Fig. 144)

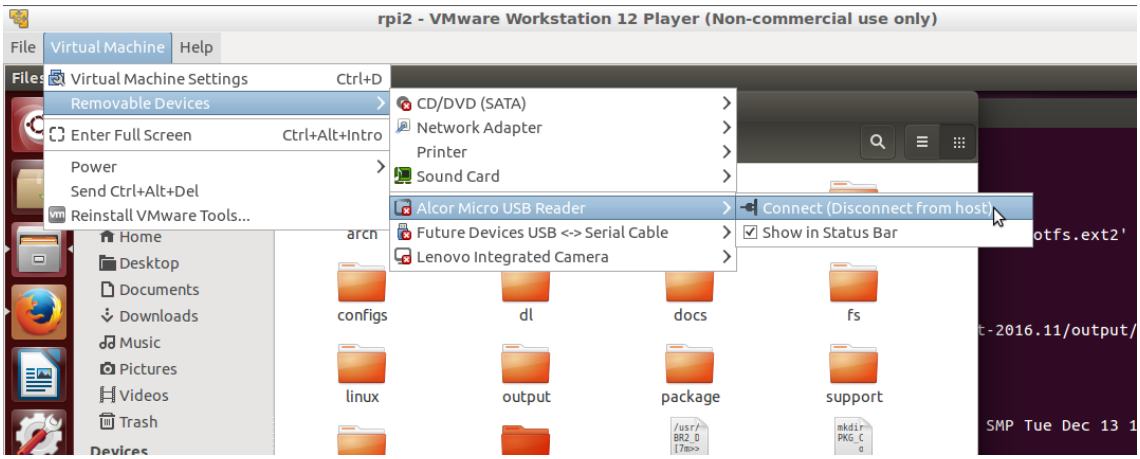


Fig. 144: Connecting a removable device to our VMWare virtual machine.

If you want to access the card reader unit, but from your host computer (outside of the virtual machine), repeat the process above, but choosing "Disconnect" option.

Insert your microsd card now into Raspberry Pi.

Connect the serial cable to the computer, load Putty from a terminal windows using "*sudo putty*" command, and set it up using data shown in Fig. 145.

```
sudo putty
```

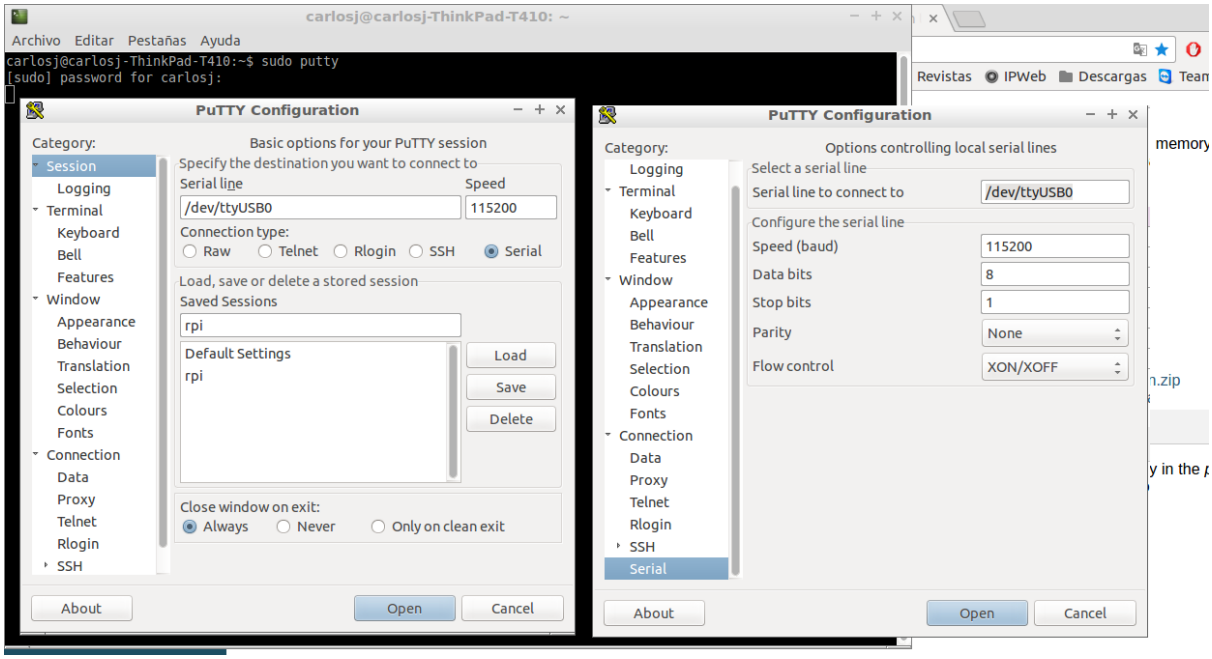


Fig. 145: Configuring Putty in the virtual machine.

To be able to use the FTDI cable into the virtual machine, we need to virtually connect it to the virtual machine. Use VMware Player upper menu, "Virtual Machine", "Removable Devices", look for FTDI cable (in our example, it is shown as "Future Devices USB<->Serial Device") and click "Connect".

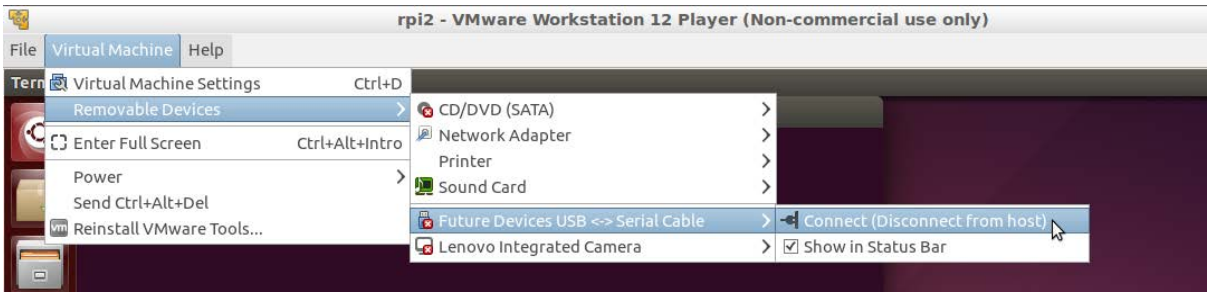


Fig. 146: Connecting FTDI USB cable to the virtual machine.

Turn on the Raspberry Pi.

Kernel messages will be shown in our “Putty” window:

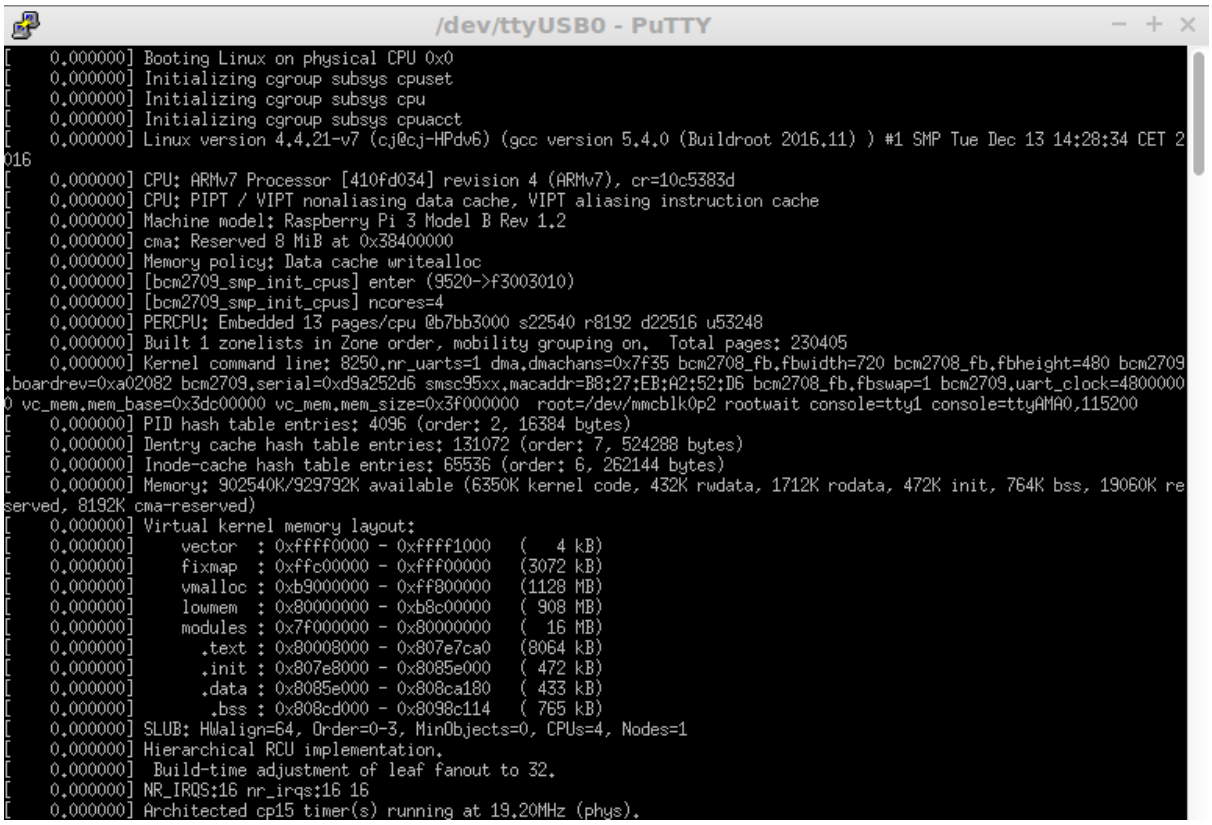


Fig. 147: Messages shown during Raspberry Pi boot up process.

And finally, our own custom Linux system welcome us:

Fig. 148: Welcome prompt of our own embedded Linux.

Login user is “root”, no password required.
Prompt # denotes that we have access to Linux shell.

The following command shows the directory structure of our system:

```
ls -al /
```

```
# ls -al /
total 31
drwxr-xr-x   18 root     root          1024 Dec 13  2016 .
drwxr-xr-x   18 root     root          1024 Dec 13  2016 ..
drwxr-xr-x    2 root     root          3072 Dec 13  2016 bin
drwxr-xr-x    7 root     root          2600 Jan  1 00:00 dev
drwxr-xr-x    5 root     root          1024 Jan  1 00:00 etc
drwxr-xr-x    4 root     root          1024 Dec 13  2016 lib
lrwxrwxrwx    1 root     root             3 Dec 13  2016 lib32 -> lib
lrwxrwxrwx    1 root     root            11 Dec 13  2016 linuxrc -> bin/busybox
drwx------    2 root     root         16384 Dec 13  2016 lost+found
drwxr-xr-x    2 root     root          1024 Nov 30  2016 media
drwxr-xr-x    2 root     root          1024 Nov 30  2016 mnt
drwxr-xr-x    2 root     root          1024 Nov 30  2016 opt
dr-xr-xr-x   99 root     root             0 Jan  1 00:00 proc
drwx------    2 root     root          1024 Jan  1 00:00 root
drwxr-xr-x    3 root     root           140 Jan  1 00:00 run
drwxr-xr-x    2 root     root          1024 Dec 13  2016 sbin
dr-xr-xr-x   12 root     root             0 Jan  1 00:00 sys
drwxrwxrwt    2 root     root            80 Jan  1 00:00 tmp
drwxr-xr-x    6 root     root          1024 Dec 13  2016 usr
drwxr-xr-x    4 root     root          1024 Dec 13  2016 var
#
```

Fig. 149: Directory structure of the generated image.

A summary of available commands will be shown using “help” command. (Fig. 150)

```
help
```

```
# help
Built-in commands:
------------------
        . : [ [[ alias bg break cd chdir command continue echo eval exec
        exit export false fg getopts hash help history jobs kill let
        local printf pwd read readonly return set shift source test times
        trap true type ulimit umask unalias unset wait

#
```

Fig. 150: Help command.

Type “exit”, to go back to Buildroot prompt (log out).

```
exit
```

```
# pwd
/root
# exit

Welcome to Buildroot
buildroot login:
```

Fig. 151: Exit command.

The complete log of this session, showing all kernel messages, etc. (“putty_Primera compilacion imagen rpi3_13_12_171647.log” file) is available at:

https://drive.google.com/open?id=0BwgR6nANeJK1dXhUS29mTkQ3MDg

[Help]: In case errors appeared, it is possible to start from scratch, using commands: ***"make clean"***, ***"make distclean"*** or ***"make clean all"***. Use of these commands and their effects are shown at:
https://buildroot.org/downloads/manual/manual.html#_general_buildroot_usage

6.2 Necessary changes in Buildroot

Some changes must be made to the default settings used in the previous step, in order to host and run EPICS in our embedded operating System.
First of all, we need to increase the size of the second partition, to make room for all EPICS necessary files (its size was only 60MB by default).
C library in use must change also, from *uclibc* to *glibc*. This is due to the fact that *uclibc* is a smaller version of *glibc*. Glibc, although it is larger, it is necessary to use EPICS.
We will also include Perl language in our embedded system, and in addition, a SSH server to be able to control the rpi from another computer.
Finally, we will activate the option to be able to debug applications, called "gdb".

Let's run "make xconfig" and make the relevant changes.

```
make xconfig
```

To expand the size of the second partition, go to (Fig. 152): "Filesystem images / filesystem label / exact size in blocks", double click, and set value to 412000.

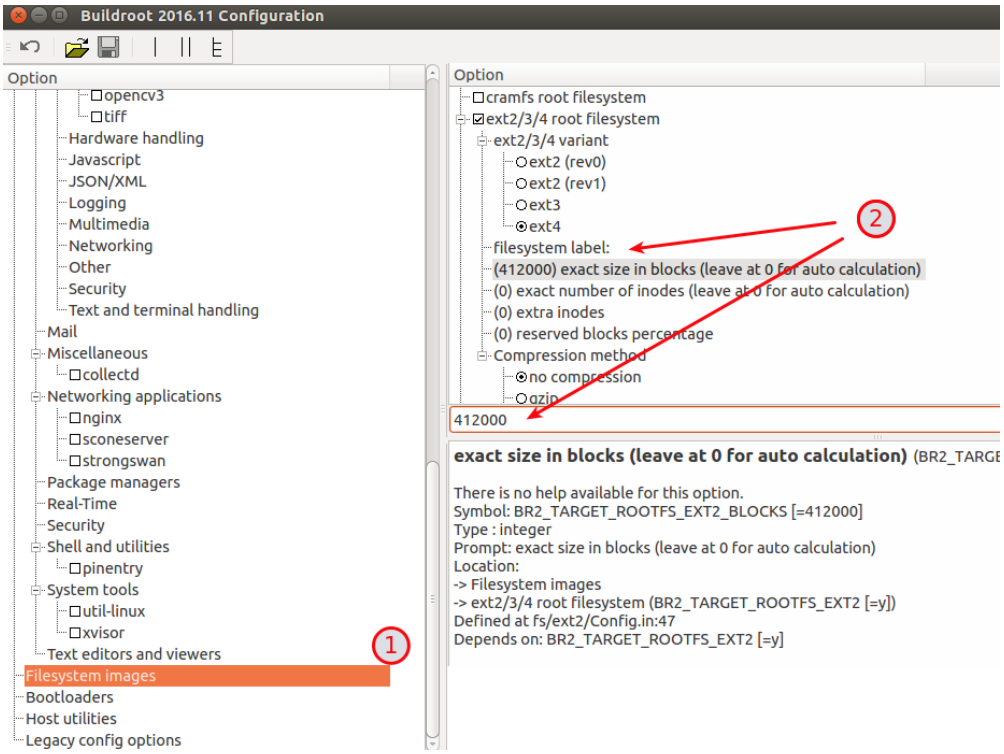


Fig. 152: Modifying the size of the generated image in Buildroot.

Select (Fig. 153) glibc library instead uclibc: “Toolchain / C library” and click “glibc”

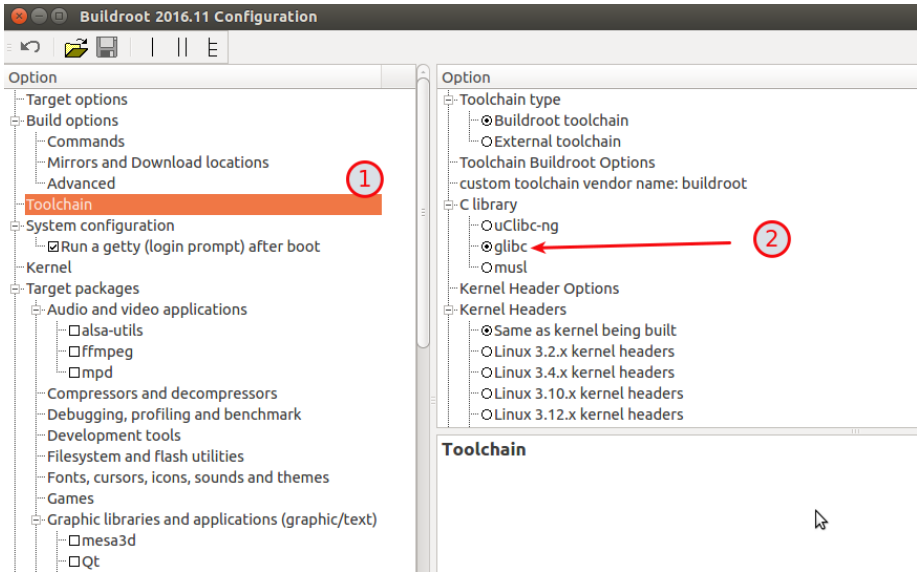


Fig. 153: Selecting glibc library in Buildroot.

Select Perl (Fig. 154): “Target Packages / Interpreter languages and scripting” option, click “perl”

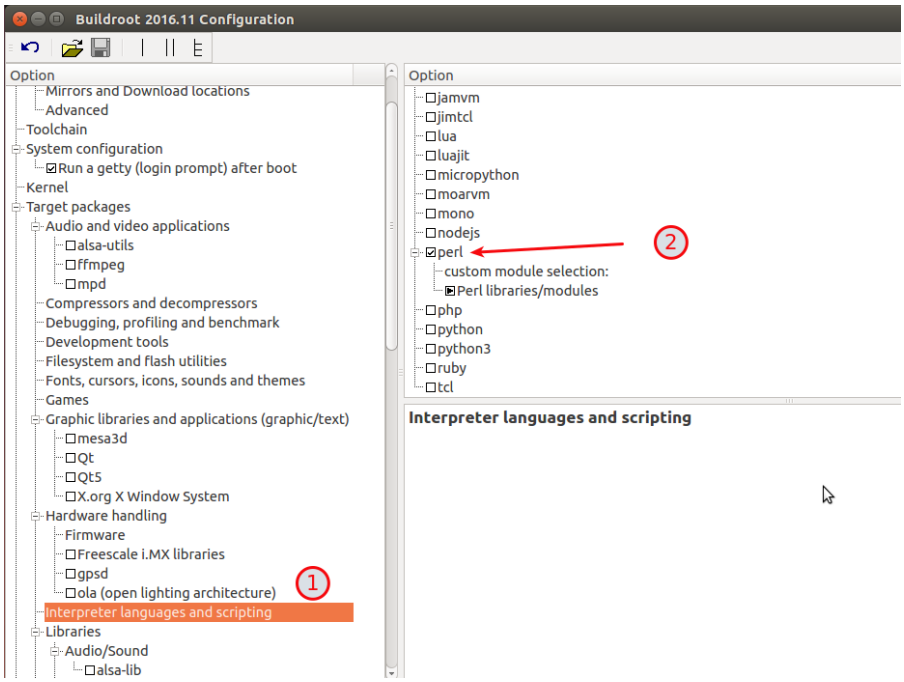


Fig. 154: Selecting Perl language in Buildroot.

Select SSH (Fig. 155): “Target packages / Networking applications” option, click “openssh”

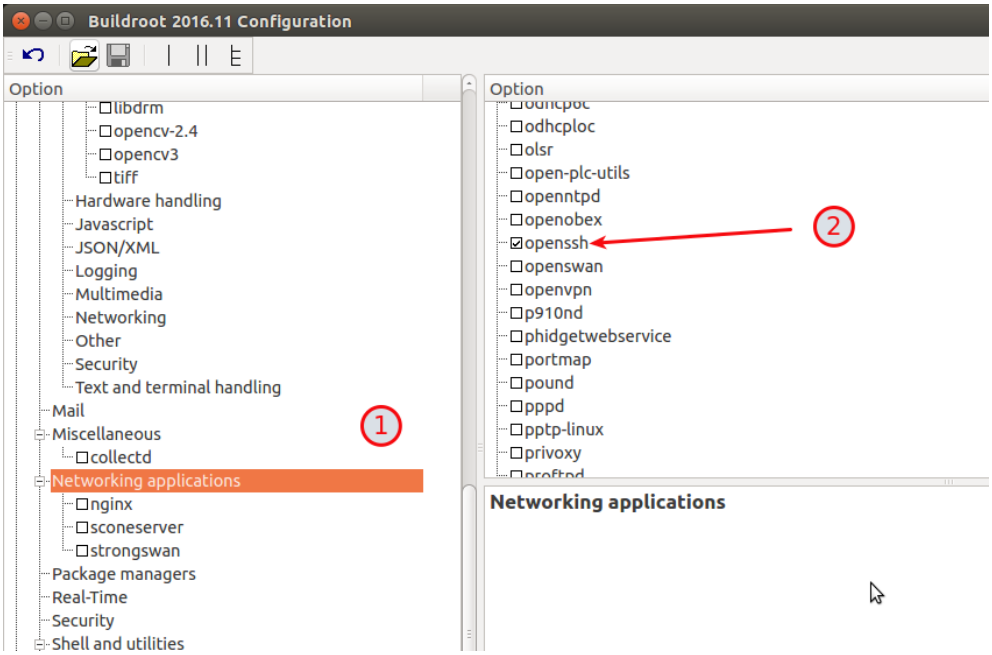


Fig. 155: Selecting openssh in Buildroot.

Select gdb (Fig. 156): “Toolchain” option, click “Build cross gdb for the host” and “gdb 7.10.x”

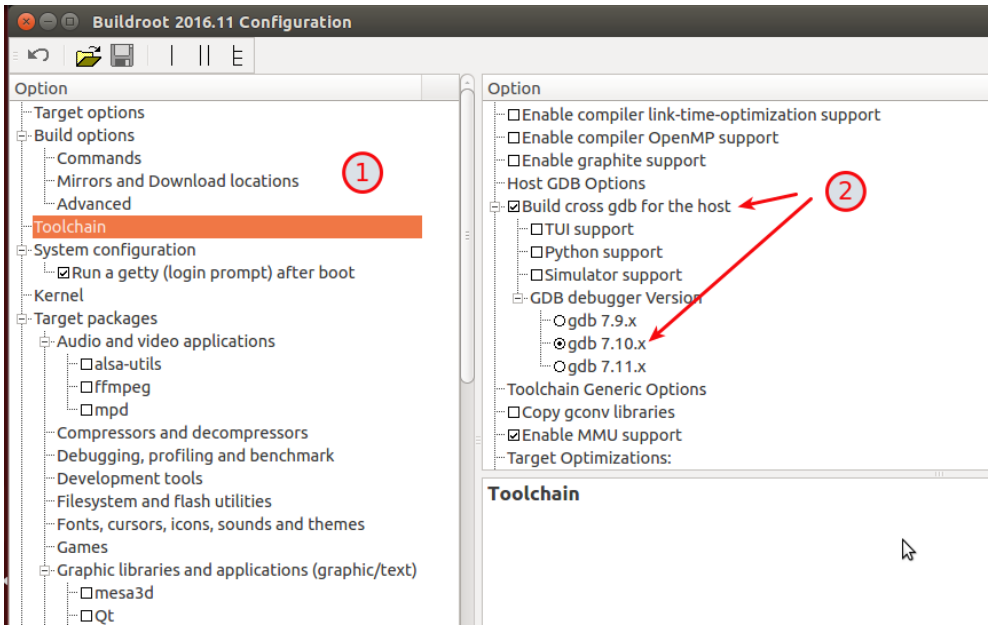


Fig. 156: Activating gdb debugging tool in Buildroot.

Click then on the floppy disk icon to save changes, and, after exiting the Buildroot environment, run the following commands to perform the compilation of the new image of our embedded Linux:

```
cd ~/Documents/buildroot-2016.11
make savedefconfig
make clean
make
```

As a result, a new image is created, its size 454MB. That image was tested in the rpi and worked fine, but, as we will see later, wifi connection needs to be configured.

We are going to see now how our changes affected the configuration file "**raspberrypi3_defconfig**", regarding to Raspberry Pi 3 default settings ("raspberrypi3_defconfig" original file). The crossed out lines are those that have disappeared with our new settings, and bold lines are the new ones.

```
BR2_arm=y
BR2_cortex_a7=y
~~BR2_ARM_EABIHF=y~~
BR2_ARM_FPU_NEON_VFPV4=y
BR2_TOOLCHAIN_BUILDROOT_GLIBC=y
BR2_PACKAGE_HOST_LINUX_HEADERS_CUSTOM_4_4=y
BR2_TOOLCHAIN_BUILDROOT_CXX=y
BR2_PACKAGE_HOST_GDB=y
BR2_SYSTEM_DHCP="eth0"
BR2_ROOTFS_POST_BUILD_SCRIPT="board/raspberrypi3/post-build.sh"
BR2_ROOTFS_POST_IMAGE_SCRIPT="board/raspberrypi3/post-image.sh"
BR2_ROOTFS_POST_SCRIPT_ARGS="--add-pi3-miniuart-bt-overlay"
BR2_LINUX_KERNEL=y
BR2_LINUX_KERNEL_CUSTOM_GIT=y
BR2_LINUX_KERNEL_CUSTOM_REPO_URL="https://github.com/raspberrypi/linux.git"
BR2_LINUX_KERNEL_CUSTOM_REPO_VERSION="9669a50a3a8e4f33b4fe138277bc4407e1eab9b
2"
BR2_LINUX_KERNEL_DEFCONFIG="bcm2709"
```

```
BR2_LINUX_KERNEL_DTS_SUPPORT=y
BR2_LINUX_KERNEL_INTREE_DTS_NAME="bcm2710-rpi-3-b"
BR2_PACKAGE_RPI_FIRMWARE=y
~~BR2_PACKAGE_RPI_FIRMWARE_INSTALL_DTB_OVERLAYS=y~~
BR2_PACKAGE_PERL=y
BR2_PACKAGE_OPENSSH=y
BR2_TARGET_ROOTFS_EXT2=y
BR2_TARGET_ROOTFS_EXT2_4=y
BR2_TARGET_ROOTFS_EXT2_BLOCKS=412000
# BR2_TARGET_ROOTFS_TAR is not set
BR2_PACKAGE_HOST_DOSFSTOOLS=y
BR2_PACKAGE_HOST_GENIMAGE=y
BR2_PACKAGE_HOST_MTOOLS=y
```

6.3 Necessary changes in Buildroot to achieve wifi connection

6.3.1 Sources (October, 2016):

https://rohitsw.wordpress.com/2016/12/17/building-a-linux-filesystem-on-raspberry-pi-3/
https://sites.google.com/site/walavalkarcoin/home/buildingalinuxfilesystemonraspberrypi3

It is still not possible to use the rpi integrated wifi, even with all the changes we made to default Buildroot settings.

Copy the image generated with Buildroot to the microsd card, boot the rpi up, and you will notice that wifi interface ("wlan0") is not listed (Fig. 157), only Ethernet ("eth0") is. Execute this command:

```
# ifconfig -a
```

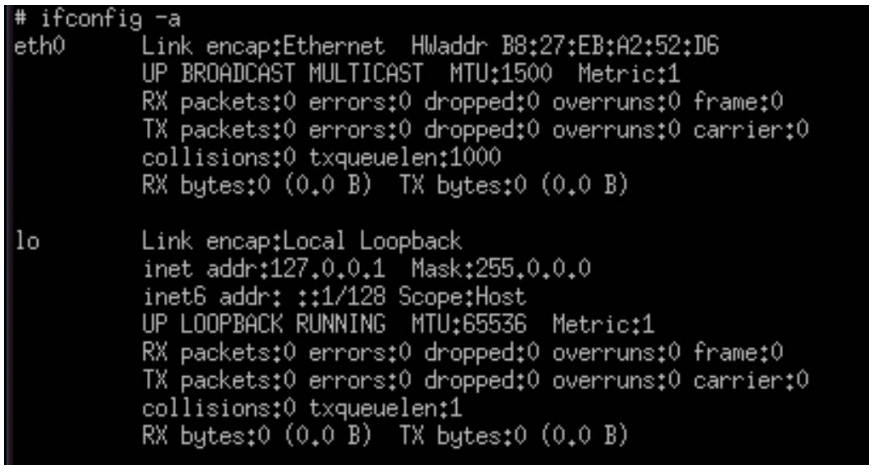


Fig. 157: ifconfig command showing missing wlan interface.

We will solve this by making some adjustments to our embedded system in Buildroot (see following chapters to know how).

6.3.2 Enabling devtmpfs option in Buildroot

Execute “make xconfig” and select "System Configurations" -> "/dev management" -> "Dynamic using Devtmpfs + mdev" option.

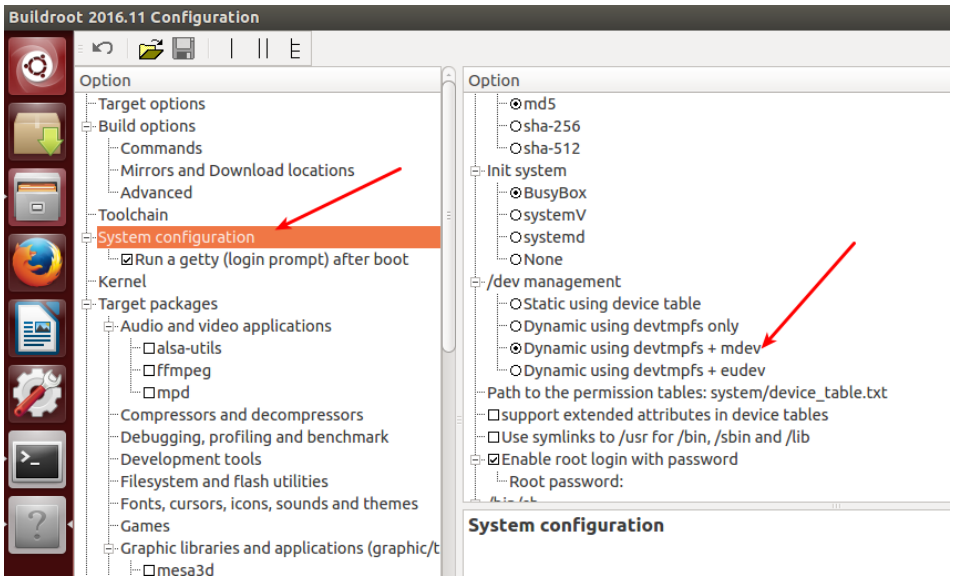


Fig. 158: Enabling devtmpfs in Buildroot.

Inside “mdev.conf” file are contained all the new settings that will be loaded after rpi booting process. We need to add the below lines to the post-build script, located at **/board/raspberrypi3/post-build.sh**:

```
cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf
```

Now run *make* and copy the resulting image to the microsd card. Boot the rpi up, and you will notice that wlan is still missing. It does not work because when driver loads, it can not find the necessary firmware, namely ***“brcmfmac43430-sdio.bin”***.

Execute this command to see more detailed info about this error:

```
# dmesg | grep brcm
```

Raspberry Pi 3 has wifi chipset built in, but firmware for its Broadcomm chipset is not available as a package in Buildroot. Apparently, this firmware should be included into “linux-firmware” Buildroot package, according to:
http://git.kernel.org/cgit/linux/kernel/git/firmware/linux-firmware.git/commit/?id=c4c07a8d1128d50a5c2885ceea1abbebaa82f820
https://git.busybox.net/buildroot/commit/?id=ad0162623327fadd65b50a6007a5dfc5c52bd0a1
But further testing confirmed that the firmware is not included.

6.3.3 Installing wifi firmware for rpi in Buildroot

What we are going to do now is add the wifi firmware ourselves as a package. We are going to create a new package in Buildroot manually, so that we can include it into our embedded Linux image.

Create a new folder named "*rpi-wifi-firmware"* inside "*package"* folder, and add two files to it: **rpi-wifi-firmware.mk** and **Config.in** (capital letter "C" is mandatory). This will add a new config option called "**rpi-wifi-firmware**" inside Buildroot menu "Target Packages" -> "Hardware Handling" -> "Firmware". We will see it when running "make xconfig".

Config.in

```
config BR2_PACKAGE_RPI_WIFI_FIRMWARE
       bool "rpi-wifi-firmware"
       help
         This package provides the wifi firmware for the Raspberry Pi
```

rpi-wifi-firmware.mk

```
RPI_WIFI_FIRMWARE_VERSION = master
RPI_WIFI_FIRMWARE_SITE = $(call github,RPi-Distro,firmware-
nonfree,$(RPI_WIFI_FIRMWARE_VERSION))
RPI_WIFI_FIRMWARE_LICENSE = Proprietary
RPI_WIFI_FIRMWARE_LICENSE_FILES = brcm80211/LICENSE

define RPI_WIFI_FIRMWARE_INSTALL_TARGET_CMDS
       $(INSTALL) -D -m 0644 $(@D)/brcm80211/brcm/brcmfmac43143.bin
$(TARGET_DIR)/lib/firmware/brcm/brcmfmac43143.bin
       $(INSTALL) -D -m 0644 $(@D)/brcm80211/brcm/brcmfmac43430-sdio.bin
$(TARGET_DIR)/lib/firmware/brcm/brcmfmac43430-sdio.bin
       $(INSTALL) -D -m 0644 $(@D)/brcm80211/brcm/brcmfmac43430-sdio.txt
$(TARGET_DIR)/lib/firmware/brcm/brcmfmac43430-sdio.txt
endef

$(eval $(generic-package))
```

Now we need to add this to the global package list, add the below line under the "Firmware" menu in "*package/Config.in"* file:

```
source "package/rpi-wifi-firmware/Config.in"
```

Resulting file is shown in Fig. 159:

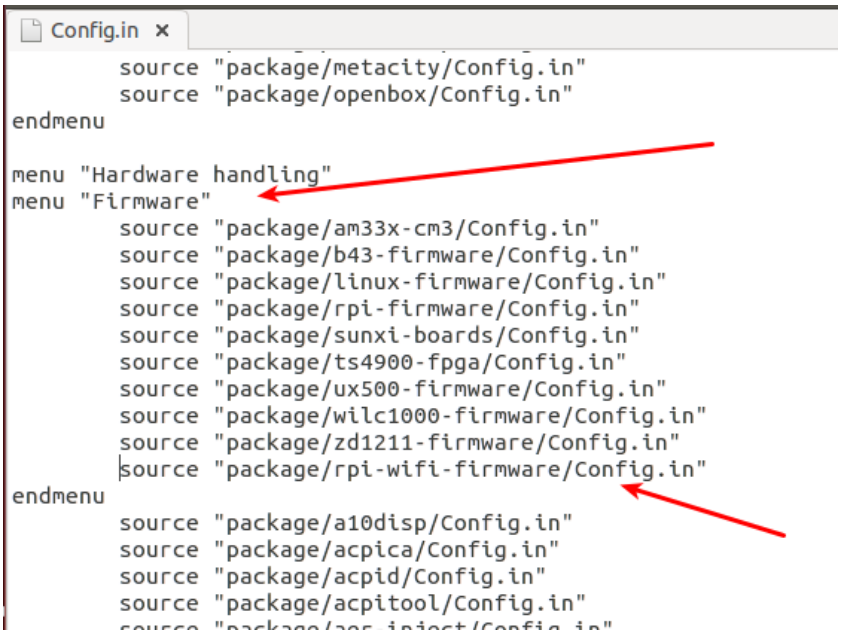


Fig. 159: Adding new wifi firmware package in "package/Config.in" file.

Execute "make xconfig", and select (Fig. 160) new rpi-wifi-firmware option:

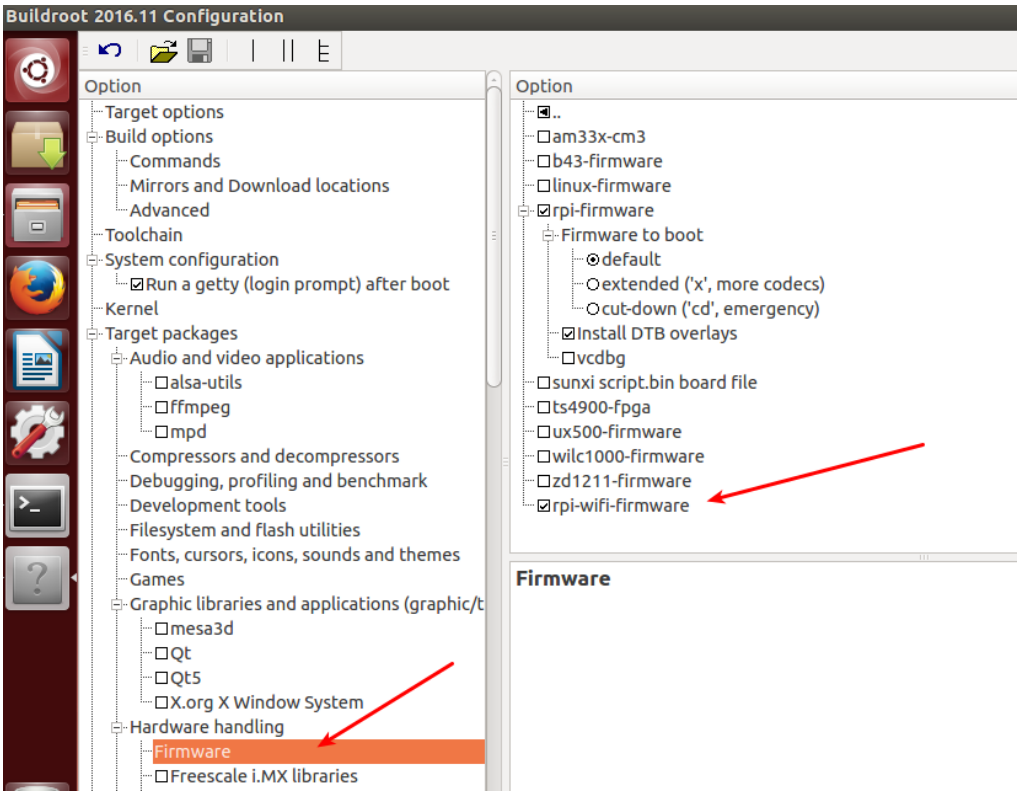


Fig. 160: Selecting new wifi firmware in Buildroot xconfig.

Execute "*make*", copy the Linux image to microsd card, boot the rpi, and (see Fig. 161) "wlan0" interface will be shown when executing the following command:

```
# ifconfig -a
```

```
Welcome to Buildroot
buildroot login: root
Password:
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr B8:27:EB:A2:52:D6
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr B8:27:EB:F7:07:83
          inet addr:138.4.219.34  Bcast:138.4.223.255  Mask:255.255.224.0
          inet6 addr: fe80::ba27:ebff:fef7:783/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:183 errors:0 dropped:135 overruns:0 frame:0
          TX packets:55 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:36808 (35.9 KiB)  TX bytes:6432 (6.2 KiB)

#
```

Fig. 161: ifconfig command showing wlan interface.

6.3.4 Connecting to wifi networks

Our wlan interface can be used into our rpi, but we will still not be able to connect to any wifi network. Our Linux distribution, due to the lack of a graphical interface, does not have a tool for searching and selecting a wifi network. We need to include the name and access key to the wifi network into a file.

First of all, execute “make xconfig”, and select (Fig. 162) "**wpa_supplicant**" in "Target Packages" -> "Networking Applications"

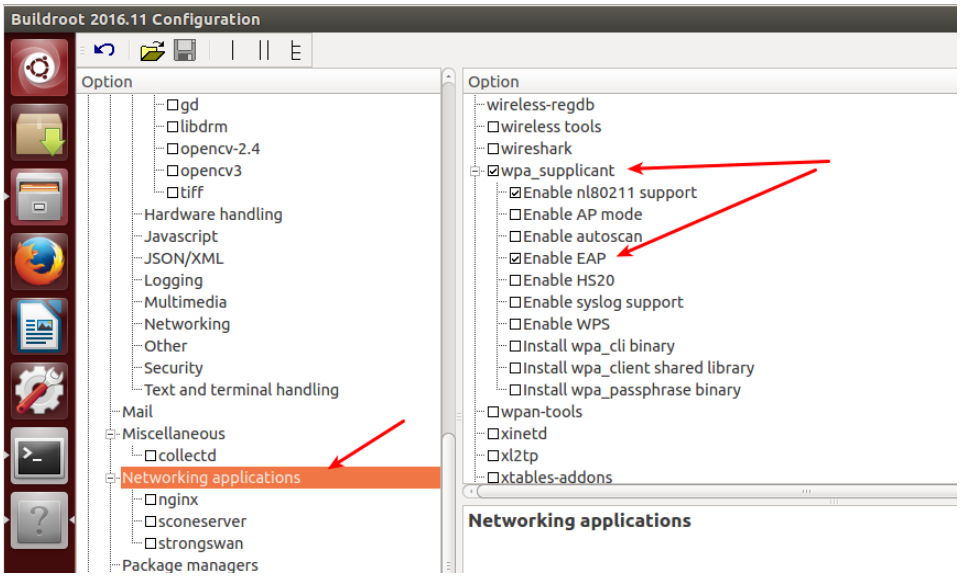


Fig. 162: Adding WPA_Supplicant package to Buildroot generated image.

Select also “**crda**”, “**iw**” and “**wireless-regdb**” packages, all located under “Networking Applications”.

Add the below lines into the post build script file **/board/raspberrypi3/post-build.sh:**

```
cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf ${TARGET_DIR}/etc/wpa_supplicant.conf
```

This will copy those two files to our Linux image.
Fig. 163 shows the contents of the resulting “post-buidl.sh” file:

post-build.sh

```
#!/bin/sh

set -u
set -e

# Add a console on tty1
if [ -e ${TARGET_DIR}/etc/inittab ]; then
    grep -qE '^tty1::' ${TARGET_DIR}/etc/inittab || \
        sed -i '/GENERIC_SERIAL/a\
tty1::respawn:/sbin/getty -L  tty1 0 vt100 # HDMI console' ${TARGET_DIR}/etc/inittab
fi

#puesto por mi para configurar wifi en rpi3
cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf
cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf ${TARGET_DIR}/etc/wpa_supplicant.conf
```

Fig. 163: post-build.sh file contents.

Now we need to create the two files into “**board/raspberrypi3”**folder. First file is called “**interfaces**”, and the other one “**wpa_supplicant.conf**”.

board/raspberrypi3/interfaces

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
       pre-up /etc/network/nfs_check
       wait-delay 15

auto wlan0
iface wlan0 inet dhcp
       pre-up wpa_supplicant -B -Dwext -iwlan0 -c/etc/wpa_supplicant.conf
       post-down killall -q wpa_supplicant
       wait-delay 15

iface default inet dhcp
```

board/raspberrypi3/wpa_supplicant.conf

```
network={
ssid="SSID"
psk="PASSWORD"
}
```

Just replace SSID for the wifi network name you want to connect (quotation marks needed) and replace PASSWORD for wifi network password (remember to put quotation marks again). Multiple networks can be declared one after the other. System will attempt to connect to the first wifi network, if it is not available will connect to the next, and so on.

Warning: no blank spaces or tabs should be used before “ssid” and “psk”, because errors will appear. It is preferable to create these files using **nano** instead of gedit editor.

Next execute “make” in Buildroot, copy the resulting image to microsd card and boot the rpi, wifi connection should be established.

Check using command “*ifconfig -a*” (Fig. 164) that “wlan0” interface is “UP” and “RUNNING”, and an IP address has been assigned to it.

```
# ifconfig -a
```

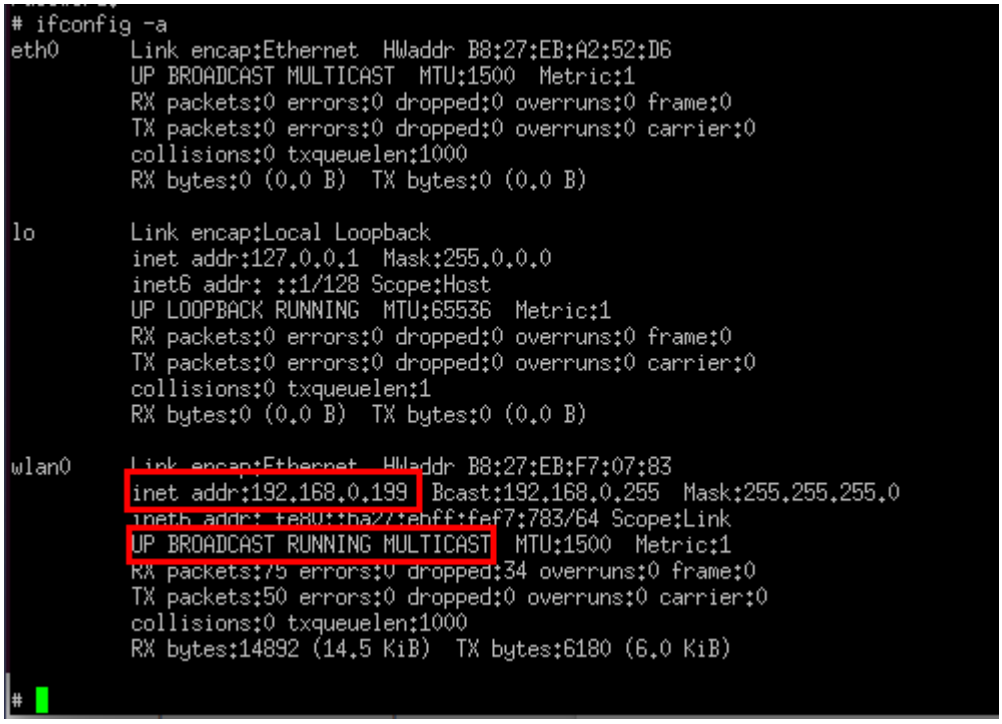


Fig. 164: Checking wifi interface in image generated using Buildroot.

Assigned IP address is 192.168.0.199, wlan0 interface is “UP” and “RUNNING”, that is correct.

We are going to see now a sample file "**wpa_supplicant.conf"** containing three networks (passwords have been omitted for security reasons). Last one of them shows the required settings to connect to Campus network "WIFIUPM". Student's own account data must be used in the fields "identity" and "password".

board/raspberrypi3/wpa_supplicant.conf

```
network={
ssid="CubotX17"
psk="xxxxxxxxx"
}
network={
ssid="vodafone46Ax"
psk="xxxxxxxxx"
}
network={
ssid="WIFIUPM"
key_mgmt=WPA-EAP
eap=PEAP
identity="xxxxxxx@alumnos.upm.es"
password="xxxxxxxxx"
}
```

6.4 Setting date and time using NTP

Bibliography:
http://www.pool.ntp.org/es/use.html
http://www.pool.ntp.org/zone/es
https://wiki.openwrt.org/doc/uci/system
http://souptonuts.sourceforge.net/README_Working_With_Time.html

If we want the rpi to obtain correct date and time from the Internet, it is necessary to add to our image in Buildroot a package called "NTP", from "Networking applications" section (Fig. 165).

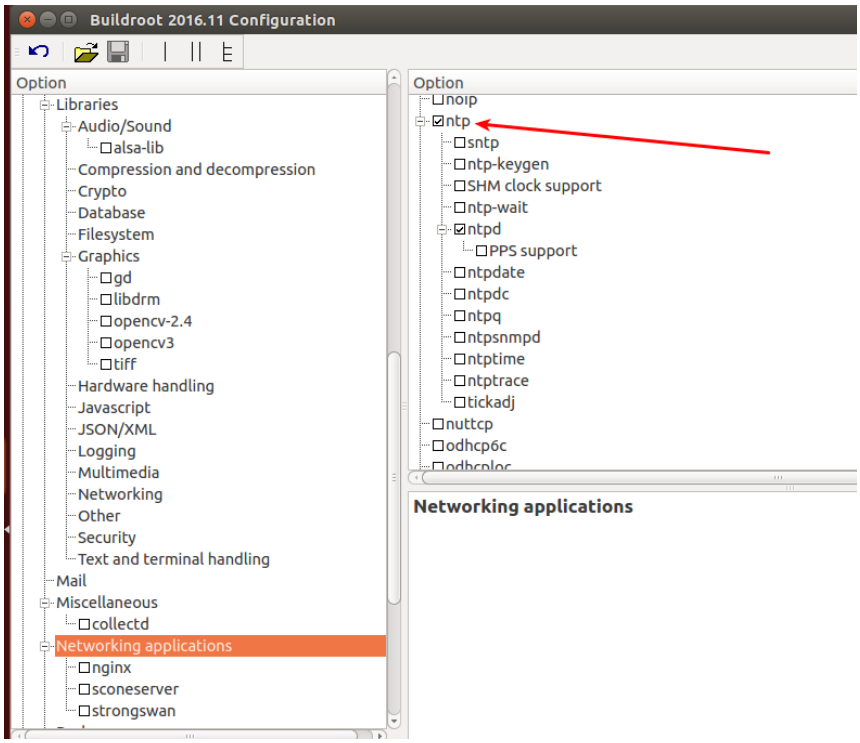


Fig. 165: Including NTP protocol in Buildroot.

6.4.1 Setting NTP after Buildroot image is built

First of all, we need to configure the NTP service, in order to get from the Internet date/time data. Servers close to our location must be used.

Edit "*/etc/ntp.conf*" file included in the resulting image (located in the 400MB second partition) so it uses servers that provide Spanish time zone (see Fig. 166).

```
server 3.es.pool.ntp.org iburst
server 0.europe.pool.ntp.org iburst
server 2.europe.pool.ntp.org iburst

# Allow only time queries, at a limited rate, sending KoD when in excess.
# Allow all local queries (IPv4, IPv6)
restrict default nomodify nopeer noquery limited kod
restrict 127.0.0.1
restrict [::1]
```

Fig. 166: ntp.conf file contents.

In addition, it is necessary to set the environment variable "TZ" to our time zone data. For example, data for Madrid (Spain) zone are these: CET-1CEST,M3.5.0,M10.5.0/3

Execute the following command:

```
# export TZ=CET-1CEST,M3.5.0,M10.5.0/3
```

Check date and time using command "*date*":

```
# date
```

Fig. 167: Checking date and time.

6.4.2 Setting NTP before Buildroot image is built

Previous settings have been done **after** our image was created using Buildroot. It would be better to include these settings in our image **before** it is created. So, we will define "TZ" environment variable into "***profile***" file as we did previously with EPICS variables, and we will create ***ntp.conf*** file too, so that it will be included into our image using ***post-build.sh*** file.

So, prior to Buildroot build process, create "***ntp.conf***" file in Ubuntu virtual machine (use preferably "nano ntp.conf" command). File should be located into "*buildroot-2016.11/board/raspberrypi3*" folder, and its contents are shown in the following box:

buildroot-2016.11/board/raspberrypi3/ntp.conf

```
server 3.es.pool.ntp.org iburst
server 0.europe.pool.ntp.org iburst
server 2.europe.pool.ntp.org iburst
# Allow only time queries, at a limited rate, sending KoD when in excess.
# Allow all local queries (IPv4, IPv6)
restrict default nomodify nopeer noquery limited kod
restrict 127.0.0.1
restrict [::1]
```

Create "***profile***" file into "*buildroot-2016.11/board/raspberrypi3*" folder too, with these contents:

buildroot-2016.11/board/raspberrypi3/profile

```
# Defining Madrid time zone
export TZ=CET-1CEST,M3.5.0,M10.5.0/3
```

Add the following line at the end of "*board/raspberrypi3/**post-build.sh***" file:

board/raspberrypi3/post-build.sh

```
cp board/raspberrypi3/ntp.conf ${TARGET_DIR}/etc/ntp.conf
```

After all the changes, “profile” file will look like this:

profile ×

```
#Defino la variable EPICS_BASE e incluyo la ruta ejecutable de EPICS en PATH
export EPICS_BASE=/base-3.14.12.6
export PATH=/bin:/sbin:/usr/bin:/usr/sbin:/$EPICS_BASE/bin/linux-arm

# Definimos la zona horaria Madrid
export TZ=CET-1CEST,M3.5.0,M10.5.0/3

if [ "$PS1" ]; then
        if [ "`id -u`" -eq 0 ]; then
                export PS1='# '
        else
                export PS1='$ '
        fi
fi

export PAGER='/bin/more '
export EDITOR='/bin/vi'

# Source configuration files from /etc/profile.d
for i in /etc/profile.d/*.sh ; do
        if [ -r "$i" ]; then
                . $i
        fi
        unset i
done
```

Fig. 168: Adding environment variables and time zone to profile file.

“post-build.sh” file will look like this:

post-build.sh ×

```
#!/bin/sh

set -u
set -e

# Add a console on tty1
if [ -e ${TARGET_DIR}/etc/inittab ]; then
    grep -qE '^tty1::' ${TARGET_DIR}/etc/inittab || \
        sed -i '/GENERIC_SERIAL/a\
tty1::respawn:/sbin/getty -L  tty1 0 vt100 # HDMI console' ${TARGET_DIR}/etc/inittab
fi

#puesto por mi para configurar wifi en rpi3
cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf
cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf ${TARGET_DIR}/etc/wpa_supplicant.conf
cp board/raspberrypi3/profile ${TARGET_DIR}/etc/profile
cp board/raspberrypi3/ntp.conf ${TARGET_DIR}/etc/ntp.conf
```

Fig. 169: Changes in post-build.sh file.

Execute “make” in Buildroot, copy the resulting image to the memory card, put it in the rpi, and boot it up.

Now you can check, using “env” and “date” commands, that environment variable “TZ” has been set right, and date and time are also correct (Fig. 170).

```
# env
# date
```

Fig. 170: Env and date commands.

7 INTEGRATING EPICS INTO RASPBERRY PI 3 EMBEDDED SYSTEM

7.1 Bibliography

"EPICS Application Developer's Guide" [RD3]
https://wiki-ext.aps.anl.gov/epics/index.php/How_To_cross_compile_EPICS_and_a_IOC_to_an_old_x86_Linux_system
http://nuclear.unh.edu/wiki/index.php?title=Cross-compiling_EPICS
https://blog.kitware.com/cross-compiling-for-raspberry-pi/
http://www.aps.anl.gov/epics/tech-talk/2006/msg01318.php
http://www.aps.anl.gov/epics/tech-talk/2014/msg00274.php
http://www.aps.anl.gov/epics/EpicsDocumentation/AppDevManuals/iocScm-3.13.2/creatingTops.html
About generating an IOC:
http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/node3.html
About EPICS_BASE directory structure:
https://wiki.gsi.de/foswiki/pub/Epics/EpicsInstallationsAtGsiBase/README.html

7.2 EPICS cross-compilation

We are going to carry out in the following paragraphs a cross-compilation of EPICS system. We are going to compile EPICS in our Ubuntu virtual machine (x86 microprocessor environment), but the result of the compilation will be run in our Raspberry Pi (ARM microprocessor environment).
Our Host will be the Ubuntu 14 environment, and the Target will be the Raspberry Pi.

7.2.1 Downloading and preparing

The latest stable release of EPICS is 3.14.12.6 (December 9, 2016), according to http://www.aps.anl.gov/epics/base/R3-14/12.php

Execute our Ubuntu 14.04.5 32 bits virtual machine, then create a folder for EPICS in "*Home/Documents/*", download EPICS Base into that folder (1.4MB) and unpack it, the result is shown at Fig. 171.

```
wget https://www.aps.anl.gov/epics/download/base/baseR3.14.12.6.tar.gz
tar xzf baseR3.14.12.6.tar.gz
```

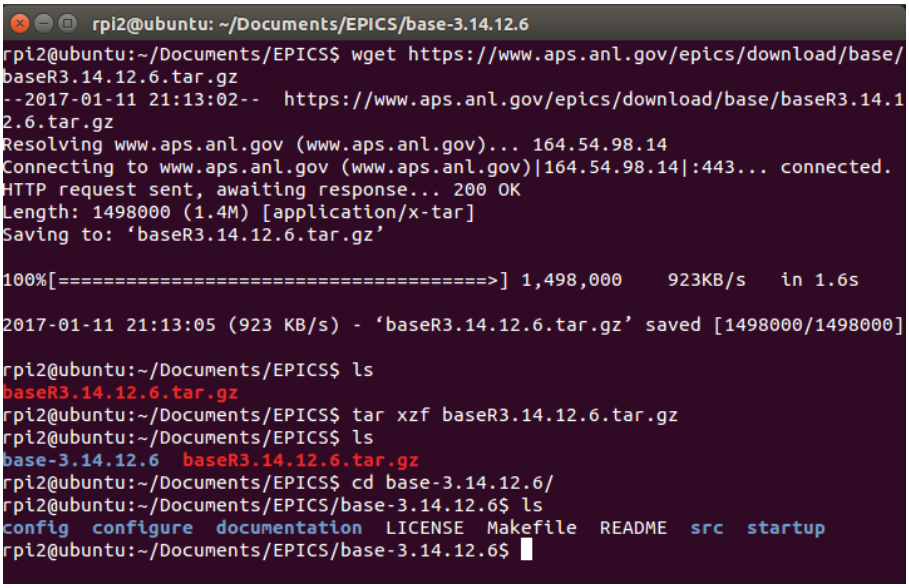


Fig. 171: Downloading and unpacking EPICS Base in Ubuntu.

A new folder called base-3.14.12.6 with several subdirectories is created:

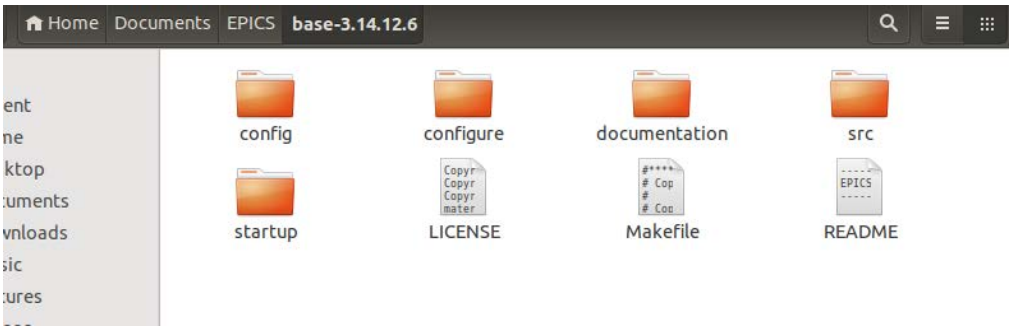


Fig. 172: EPICS Base directory structure in Ubuntu.

Edit (Fig. 173) "*CONFIG_SITE"* file inside *"configure"* folder

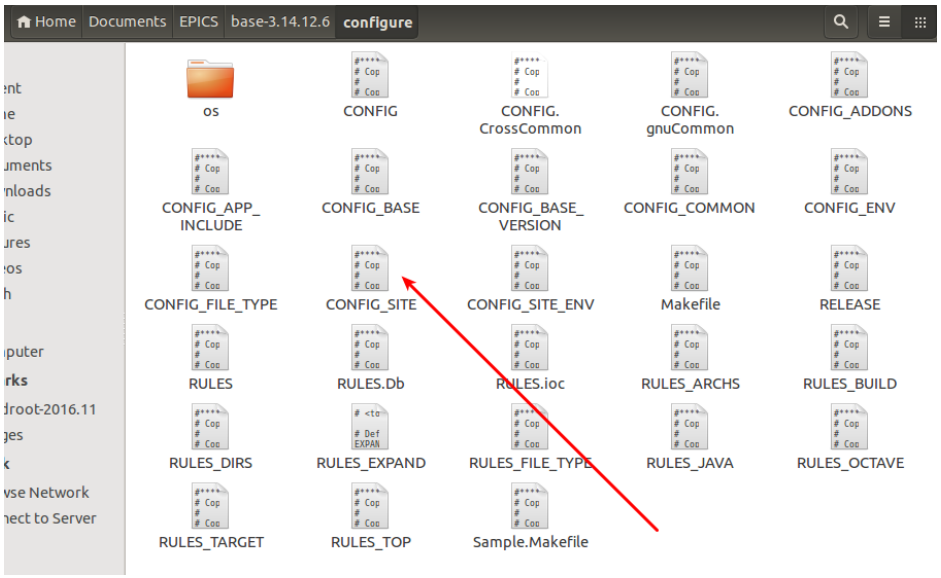


Fig. 173: CONFIG_SITE file location.

Look for *CROSS_COMPILER_TARGET_ARCHS* line (it is empty by default) and fill it with:

```
CROSS_COMPILER_TARGET_ARCHS=linux-arm
```

This is so because our target destination is the rpi.

Look for *CROSS_COMPILER_HOST_ARCHS* line, and fill it with:

```
CROSS_COMPILER_HOST_ARCHS=linux-x86
```

This is so because our host is a 32 bits operating system (Ubuntu 14 virtual machine). The following lines must be changed too:

```
SHARED_LIBRARIES=NO
STATIC_BUILD=YES
```

Save the file.

Fig. 174 shows the contents of “CONFIG_SITE” file:

CONFIG_SITE ×

```
# configure/os/CONFIG_SITE.<host>.Common files will
# override
#
CROSS_COMPILER_TARGET_ARCHS=linux-arm
#CROSS_COMPILER_TARGET_ARCHS=vxWorks-ppc32

#  If only a subset of the host architectures perform
#  the build for the CROSS_COMPILER_TARGET_ARCHS
#  uncomment the following line and specify them.
#
CROSS_COMPILER_HOST_ARCHS=linux-x86

#  The 'make runtests' and 'make tapfiles' build targets normally only run
#  self-tests for the EPICS_HOST_ARCH architecture. If the host can execute
#  the self-test programs for any other cross-built architectures such as
#  a -debug architecture, those architectures can be named here.
#
CROSS_COMPILER_RUNTEST_ARCHS=

#  Build shared libraries?
#        must be either YES or NO
#     NOTE: os/CONFIG.$(EPICS_HOST_ARCH).$(EPICS_HOST_ARCH) files and
#     os/CONFIG_SITE.$(EPICS_HOST_ARCH).$(EPICS_HOST_ARCH) files may override
#
#     NOTE Windows: YES results in a DLL.  Valid settings are
#        SHARED_LIBRARIES=YES  and  STATIC_BUILD=NO
#        SHARED_LIBRARIES=NO   and  STATIC_BUILD=YES
#
SHARED_LIBRARIES=NO

#  Build client objects statically ?
#        must be either YES or NO
#
STATIC_BUILD=YES

#  Should header dependancy files be automatically generated
```

Fig. 174: Changes made into CONFIG_SITE file.

Edit now "*CONFIG_SITE.linux-x86.linux-arm*" file into "*/configure/os"* folder:

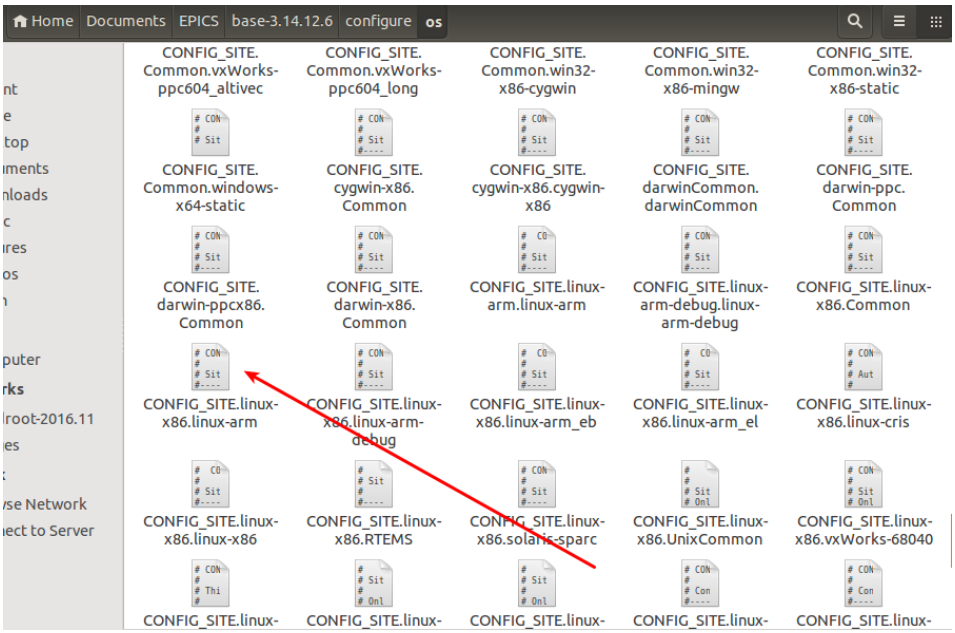


Fig. 175: CONFIG_SITE.linux-x86.linux-arm file location.

Look for GNU_DIR line and add the following route (it can be different in your system):

```
GNU_DIR = /home/rpi2/Documents/buildroot-2016.11/output/host/usr
```

That way, Buildroot will be the compiler to be used. Add the following line too:

```
GNU_TARGET = arm-buildroot-linux-gnueabihf
```

That will select that option for the target.

NOTE: There are no mention to SHARED_LIBRARIES and STATIC_BUILD in this file because they have been already included into CONFIG_SITE file into *configure* folder.

A screenshot showing all changes made into "*CONFIG_SITE.linux-x86.linux-arm"* file is shown in Fig. 176:

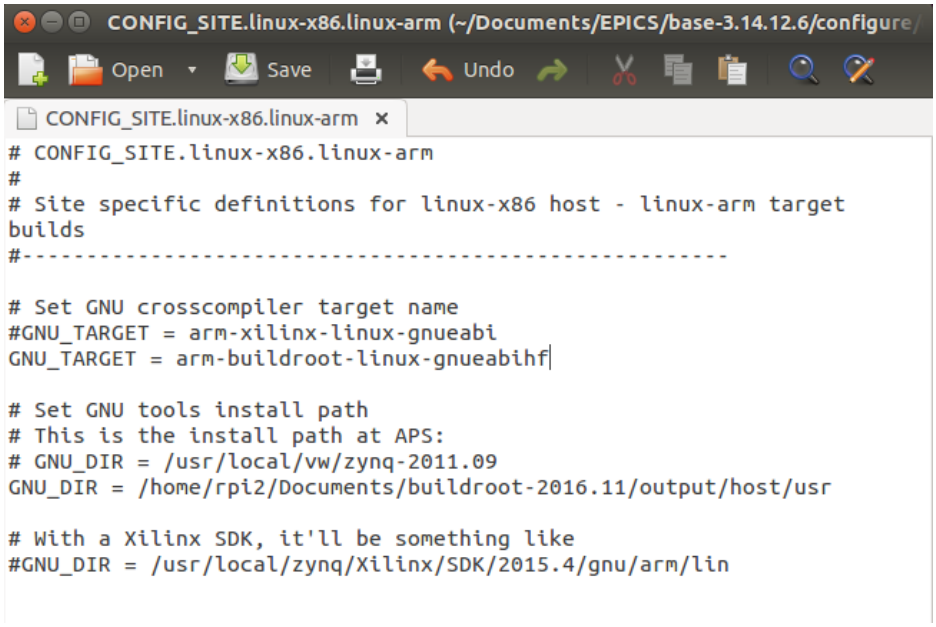


Fig. 176: CONFIG_SITE.linux-x86.linux-arm file modifications.

7.2.2 EPICS cross-compilation

Now we are going to compile EPICS. From *EPICS/base-3.14.12.6* folder, execute the command "*make*".

Fig. 177 shows the contents of the folder before the *make* process.

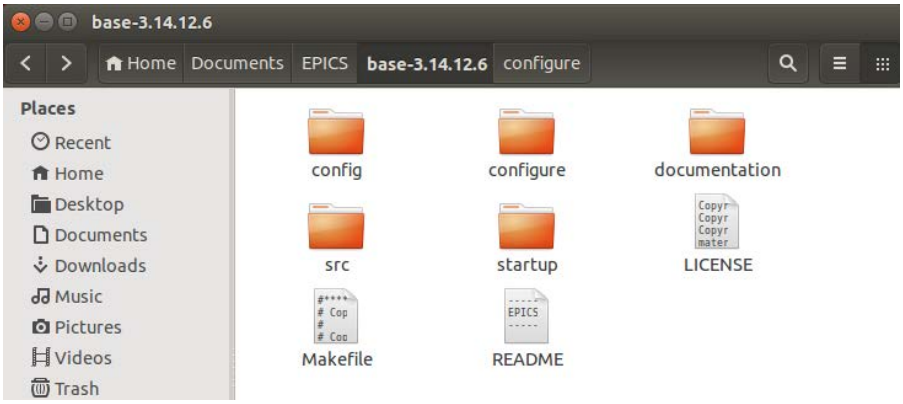


Fig. 177: EPICS Base folder contents before compilation.

An error is shown (see Fig. 178), "*readline.h*" library is not found:

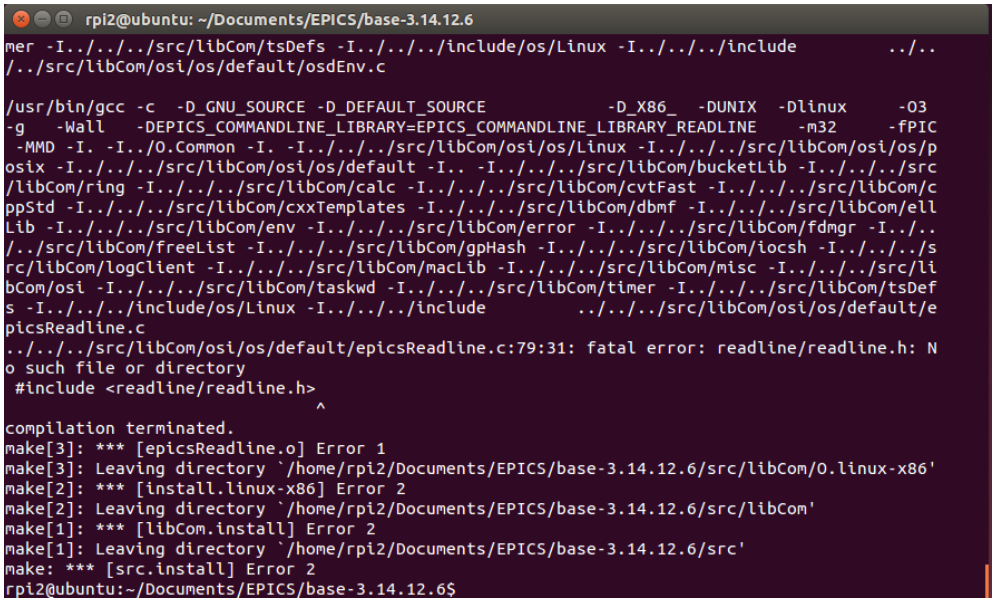


Fig. 178: Compilation error due to readline.h library missing.

Execute command "*make clean uninstall"* is needed to start from scratch the compilation process.

Install "*readline"* development library, called "*libreadline6-dev*", using Synaptic (see Fig. 179)

Another way of installing the library is using this command:

```
sudo apt-get install libreadline6-dev
```

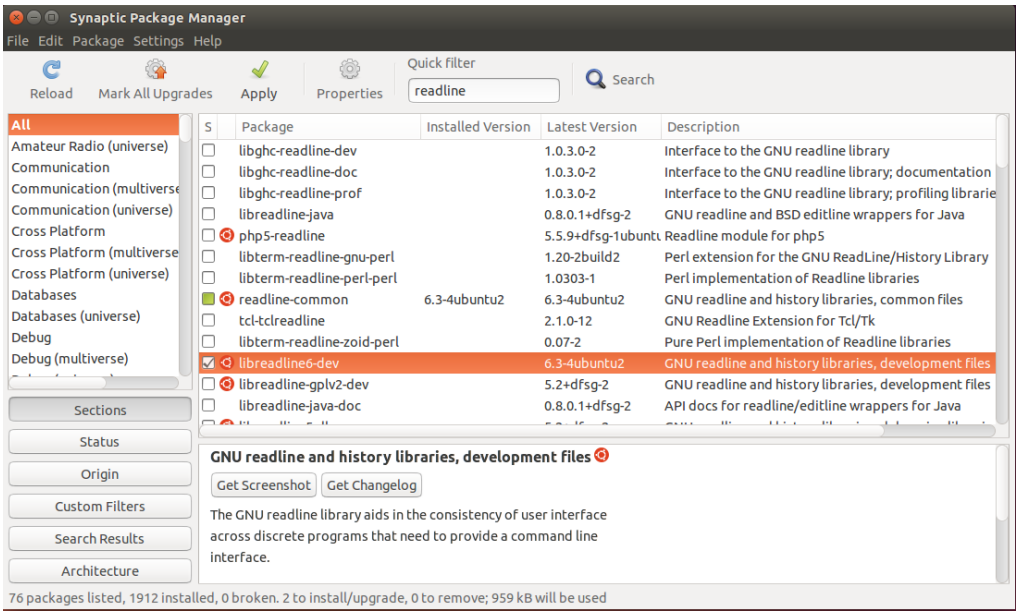


Fig. 179: Installing "libreadline6-dev" library from Synaptic.

Start from scratch again using "*make clean uninstall*" and "*make*", no errors are shown this time.

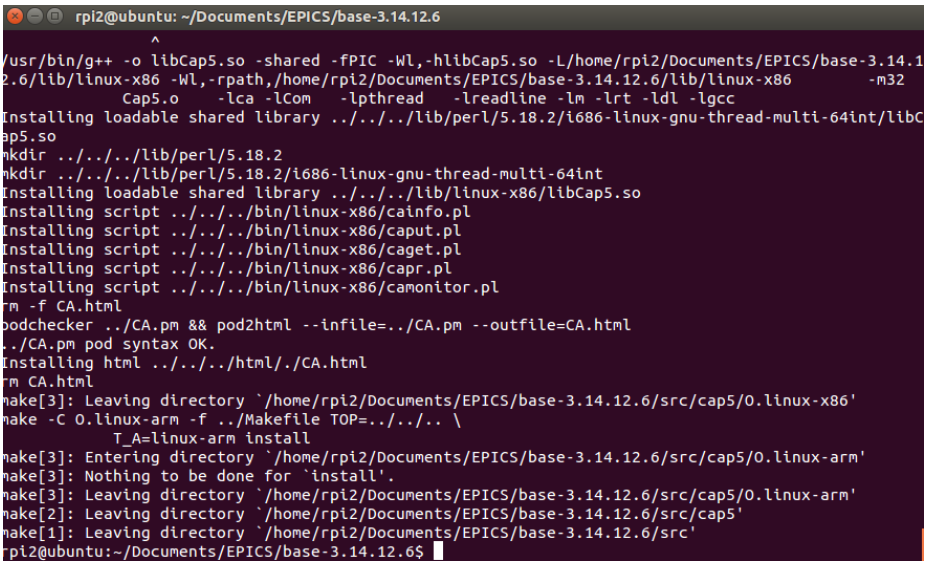


Fig. 180: EPICS Base cross-compilation result.

Warning: If the process fails indicating that a library is missing, that library must be installed in our Ubuntu virtual machine, using Synaptic or "sudo apt-get" command.

As a result of the compilation, there are now two folders inside "bin" (Fig. 181). One folder for the target (linux-arm) and the other for the host (linux-x86).

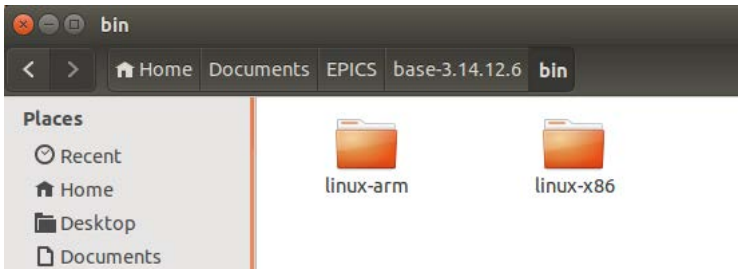


Fig. 181: Two folders as a result of EPICS Base cross-compilation.

7.2.3 Checking the cross-compilation

Execute the following command to verify that the cross-compilation process has been successful:

```
file bin/linux-arm/softIoc
```

And compare it to:

```
file bin/linux-x86/softIoc
```

We can see that one file has been compiled for ARM microprocessor architecture, and the other for Intel 80386 (x86) architecture:

```
rpi2@ubuntu: ~/Documents/EPICS/base-3.14.12.6
rpi2@ubuntu:~/Documents/EPICS/base-3.14.12.6$ ls
bin     configure  dbd            html     lib      Makefile  src      templates
config  db         documentation  include  LICENSE  README    startup
rpi2@ubuntu:~/Documents/EPICS/base-3.14.12.6$ file bin/linux-arm/softIoc
bin/linux-arm/softIoc: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 4.4.0, not stripped
rpi2@ubuntu:~/Documents/EPICS/base-3.14.12.6$ file bin/linux-x86/softIoc
bin/linux-x86/softIoc: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamically linked (use
s shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=1a0d4f4466513890aaa1e0e7a7f02d2c776117ec, not strippe
d
rpi2@ubuntu:~/Documents/EPICS/base-3.14.12.6$
```

Fig. 182: Checking cross-compilation result.

7.3 Preparing Buildroot to include EPICS

Let's create a new folder, whose contents will be the compilation result. We can put it within buildroot-2016.11 folder, into "*board/raspberrypi3"* and its name will be "*rpi3ov2017*"

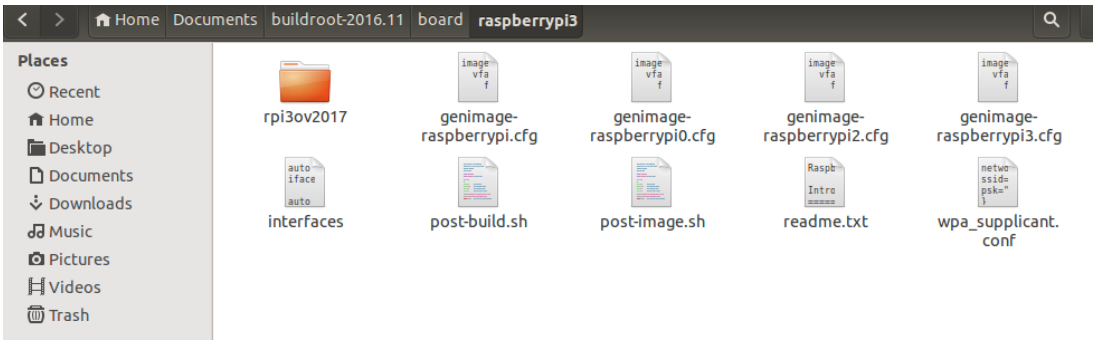
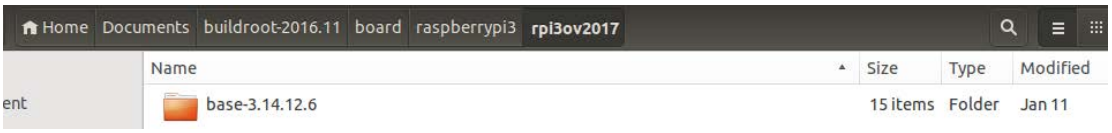


Fig. 183: Creating rpi3ov2017 folder.

Copy "*base-3.14.12.6*" (the result of EPICS compilation, 192MB) folder into "*rpi3ov2017*".



Now we need to execute "*make xconfig"* to include the "*rpi3ov2017"* folder into our Buildroot compilation. The contents of our custom folder, "*rpi3ov2017"*, will be included in the resulting "iso" image, no need to copy that folder to the microsd afterwards.

Look for "System Configuration" option, and then click "Root filesystem overlay directories". Type now the path "*board/raspberrypi3/rpi3ov2017"*, as shown in Fig. 184.

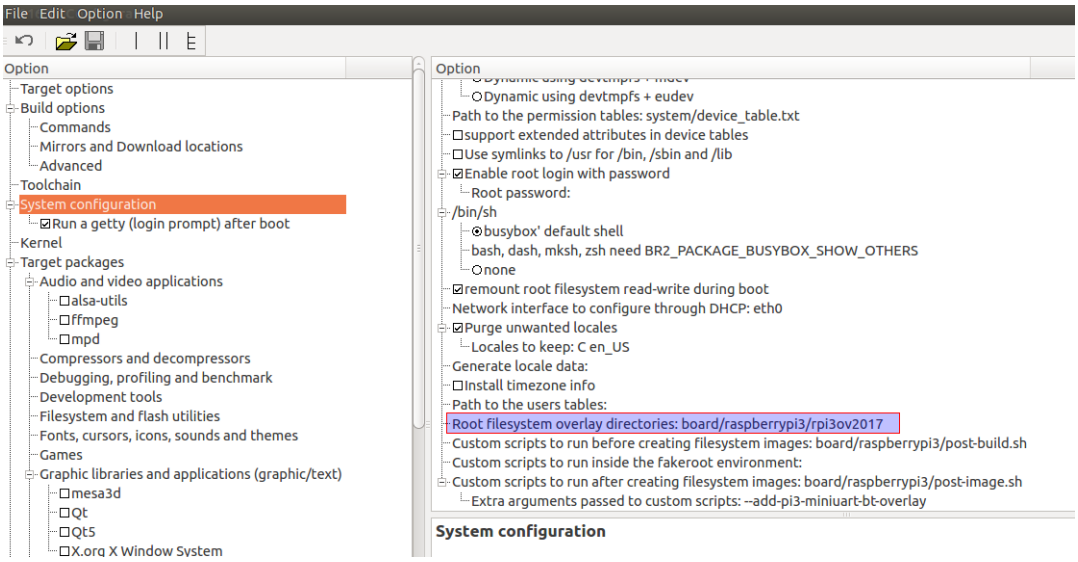


Fig. 184: Buildroot settings to include a folder into the Linux image.

The "*board/raspberrypi3/rpi3ov2017"* folder has now an EPICS folder inside, called "base-3.14.12.6".

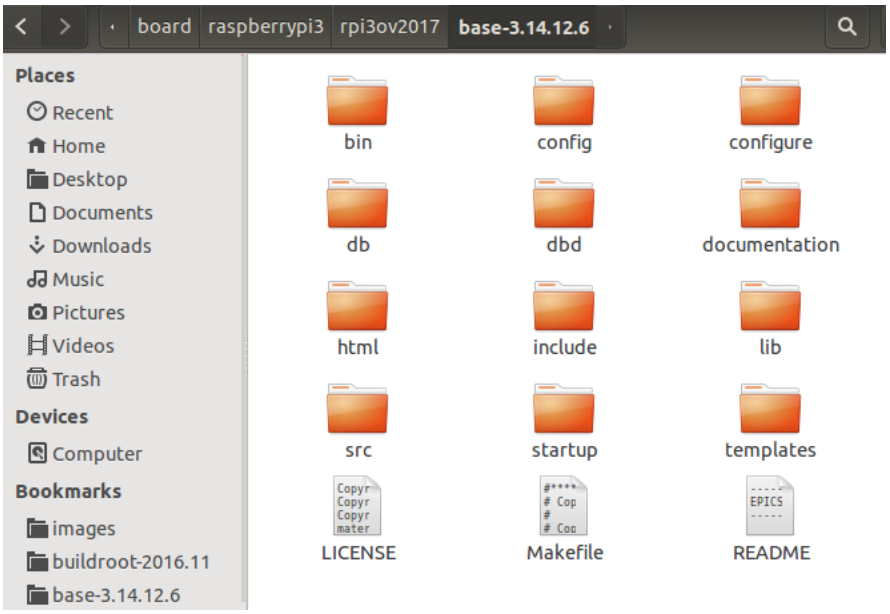


Fig. 185: EPICS Base inside Buildroot rpi3ov2017 folder.

7.4 Creating an IOC EPICS example

Next we are going to create an IOC EPICS example, using all the information located at [RD3].
Chapter two of that document provides an explanation and tutorial to create an example IOC application that will be executed in our Raspberry Pi.

First we need to create a folder called "*myexample*" into "EPICS" folder in our computer.
Then we move to that folder "*myexample".*

Execute the following command (maybe the path will be different in your system, full path to an already built copy of EPICS base must be given):

```
/home/rpi2/Documents/EPICS/base-3.14.12.6/bin/linux-x86/makeBaseApp.pl -t example myexample
```

This directory structure will appear as a result:

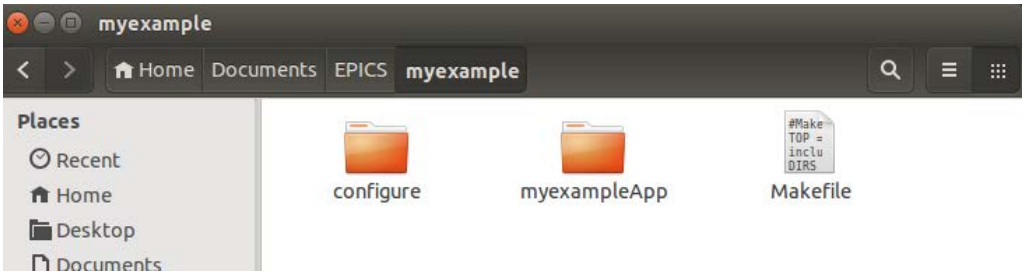


Fig. 186: Directory structure after executing makeBaseApp.pl.

Now execute:

```
/home/rpi2/Documents/EPICS/base-3.14.12.6/bin/linux-x86/makeBaseApp.pl -i -t example myexample
```

A question is shown about the IOC architecture (Fig. 187) “arm” or “x86”:

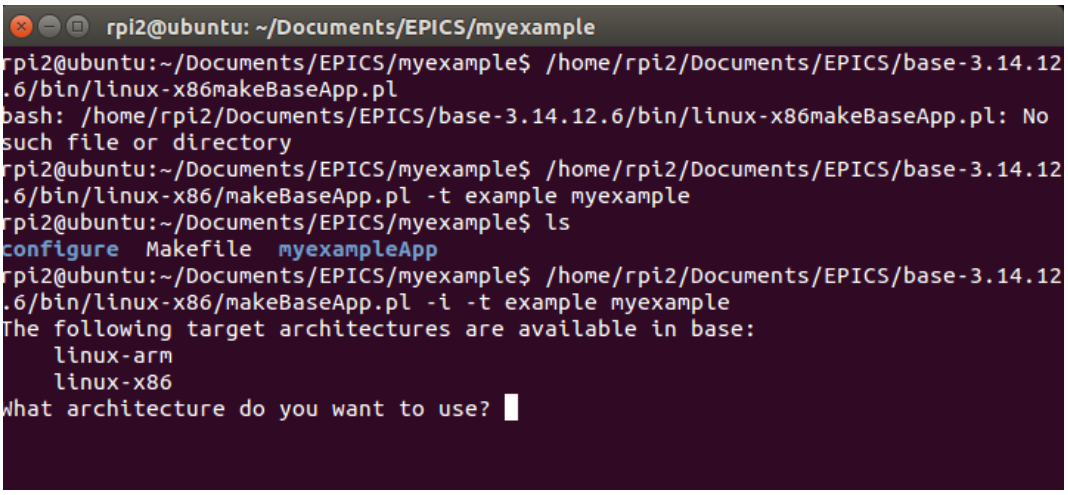


Fig. 187: Choosing an architecture during example IOC creation.

Type “linux-arm”.

IOC boot name is required too (Fig. 188):

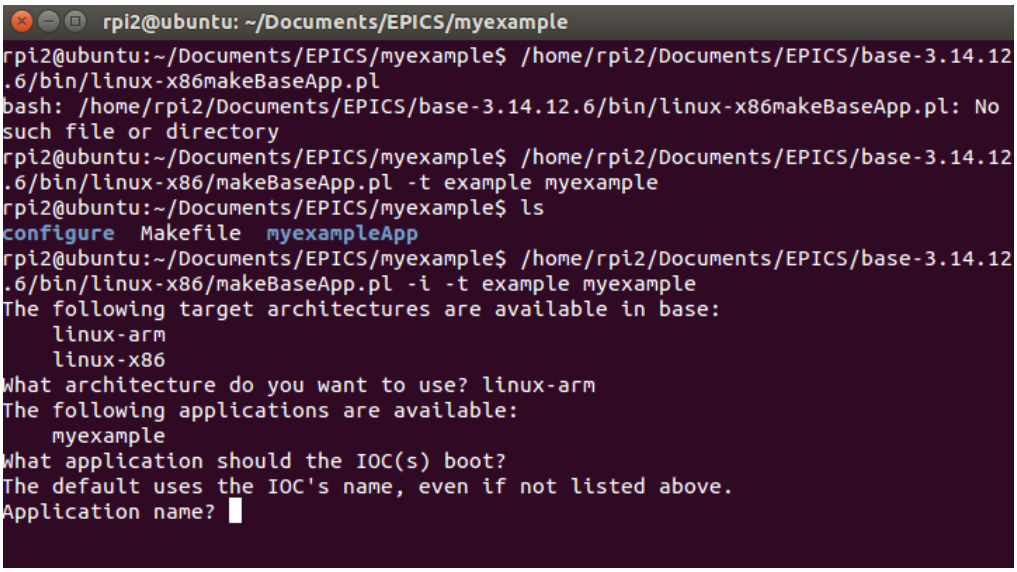


Fig. 188: Choosing a name for example IOC.

Type “myexample” (without quotation marks) and press “Intro”. When the process finished, directory structure will look like Fig. 189.

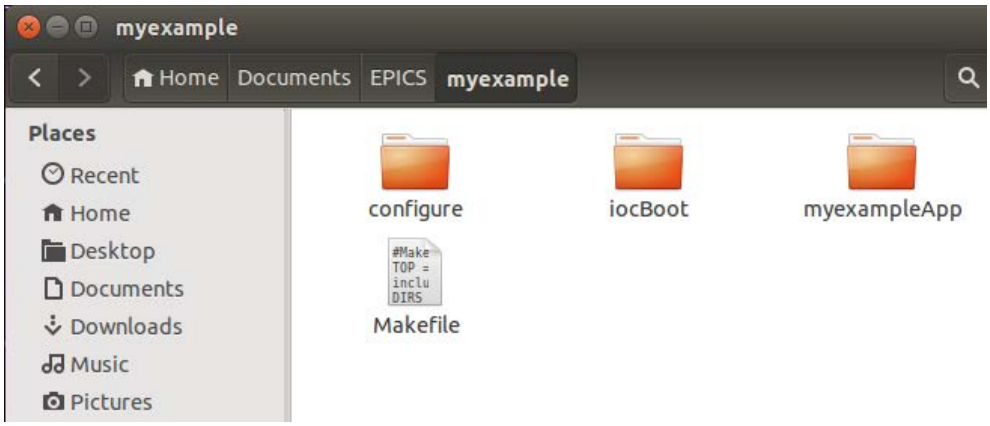


Fig. 189: myexample folder contents before example IOC building process.

Execute "*make*" (always in directory "*myexample*") and if there are no errors, result will look as Fig. 190.

```
rpi2@ubuntu: ~/Documents/EPICS/myexample
            T_A=linux-x86 install
make[3]: Entering directory `/home/rpi2/Documents/EPICS/myexample/myexampleApp/D
b/O.linux-x86'
Installing ../../../db/dbExample1.db
mkdir ../../../db
Installing ../../../db/dbExample2.db
Installing ../../../db/dbSubExample.db
Installing substitution file ../../../db/user.substitutions
Installing substitution file ../../../db/userHost.substitutions
make[3]: Leaving directory `/home/rpi2/Documents/EPICS/myexample/myexampleApp/Db
/O.linux-x86'
make -C O.linux-arm -f ../Makefile TOP=../../.. \
            T_A=linux-arm install
make[3]: Entering directory `/home/rpi2/Documents/EPICS/myexample/myexampleApp/D
b/O.linux-arm'
make[3]: Nothing to be done for `install'.
make[3]: Leaving directory `/home/rpi2/Documents/EPICS/myexample/myexampleApp/Db
/O.linux-arm'
make[2]: Leaving directory `/home/rpi2/Documents/EPICS/myexample/myexampleApp/Db
'
make[1]: Leaving directory `/home/rpi2/Documents/EPICS/myexample/myexampleApp'
make -C ./iocBoot install
make[1]: Entering directory `/home/rpi2/Documents/EPICS/myexample/iocBoot'
make -C ./iocmyexample install
make[2]: Entering directory `/home/rpi2/Documents/EPICS/myexample/iocBoot/iocmye
xample'
perl /home/rpi2/Documents/EPICS/base-3.14.12.6/bin/linux-x86/convertRelease.pl -
a linux-arm -t /home/rpi2/Documents/EPICS/myexample envPaths
make[2]: Leaving directory `/home/rpi2/Documents/EPICS/myexample/iocBoot/iocmyex
ample'
make[1]: Leaving directory `/home/rpi2/Documents/EPICS/myexample/iocBoot'
rpi2@ubuntu:~/Documents/EPICS/myexample$
```

Fig. 190: Building the example IOC.

The contents of "*myexample*" folder are shown in Fig. 191.

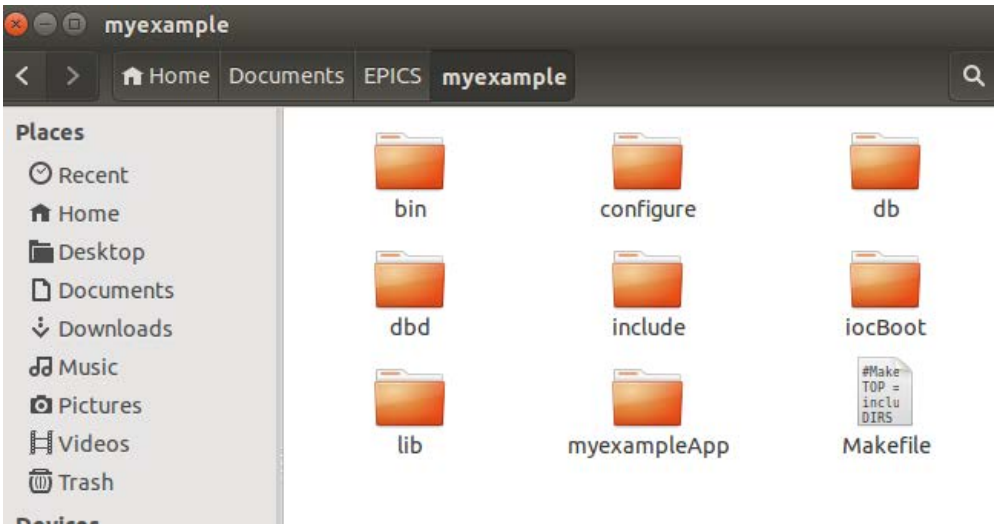


Fig. 191: myexample folder after building example IOC.

Copy "*myexample*" folder to "*rpi3ov2017*" folder, within "*Documents/buildroot-2016.11/board/raspberrypi3"*

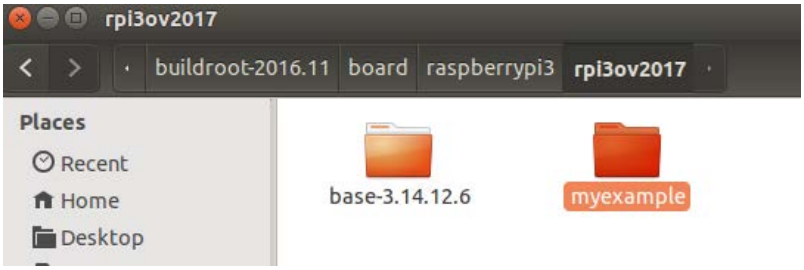


Fig. 192: Folder containing example IOC copied into Buildroot overlay folder.

After all this process, both folders containing EPICS Base and the example IOC have been prepared into Buildroot settings.

7.4.1 Setting IOC environment variables

Before the example IOC can be executed, we need to configure some environment.
Edit the file "*myexample/iocBoot/iocmyexample/envPaths*", whose contents were before any change was applied (Fig. 193):

```
epicsEnvSet("ARCH","linux-arm")
epicsEnvSet("IOC","iocmyexample")
epicsEnvSet("TOP","/home/rpi2/Documents/EPICS/myexample")
epicsEnvSet("EPICS_BASE","/home/rpi2/Documents/EPICS/base-3.14.12.6")
```

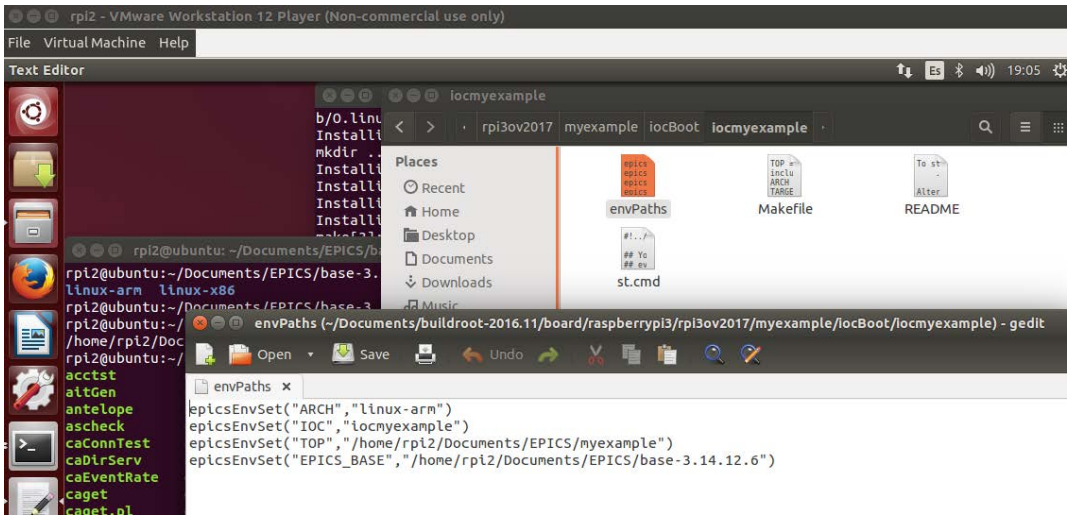


Fig. 193: File envPaths contents.

TOP variable defines the path to our application ("*myexample*"), and EPICS_BASE variable defines the path to EPICS Base ("*base-3.14.12.6*").
Now, let's change the content using the instructions described in "EPICS app developer guide" [RD3], page 119 (see following screenshot)

The envPaths file is automatically generated in the IOC's boot directory and defines several environment variables that are useful later in the startup script. The definitions shown below are always provided; additional entries will be created for each support module referenced in the application's `configure/RELEASE` file:

```
epicsEnvSet("ARCH","linux-x86")
epicsEnvSet("IOC","iocname")
epicsEnvSet("TOP","/path/to/application")
epicsEnvSet("EPICS_BASE","/path/to/base")
```

119

The file "*myexample/iocBoot/iocmyexample/envPaths"* contents should look like this (remember that it must match the directory structure of our example):

```
epicsEnvSet("ARCH","linux-arm")
epicsEnvSet("IOC","iocmyexample")
epicsEnvSet("TOP","/myexample")
epicsEnvSet("EPICS_BASE","/base-3.14.12.6")
```

7.5 Using Buildroot to integrate EPICS into our Linux image

In directory "buildroot-2016.11" execute "make" and check that folder "*rpi3ov2017"* has been included as desired, see Fig. 194:

```
mkdir -p /home/rpi2/Documents/buildroot-2016.11/output/target/etc
( \
                echo "NAME=Buildroot"; \
                echo "VERSION=2016.11"; \
                echo "ID=buildroot"; \
                echo "VERSION_ID=2016.11"; \
                echo "PRETTY_NAME=\"Buildroot 2016.11\"" \
        ) >  /home/rpi2/Documents/buildroot-2016.11/output/target/etc/os-release
>>>   Copying overlay board/raspberrypi3/rpi3ov2017
>>>   Executing post-build script board/raspberrypi3/post-build.sh
>>>   Generating root filesystem image rootfs.ext2
rm -f /home/rpi2/Documents/buildroot-2016.11/output/build/_fakeroot.fs
```

```
rpi2@ubuntu: ~/Documents/buildroot-2016.11
tune2fs 1.43.3 (04-Sep-2016)
Setting maximal mount count to -1
Setting interval between checks to 0 seconds
/usr/bin/install -m 0644 support/misc/target-dir-warning.txt /home/rpi2/Document
s/buildroot-2016.11/output/target/THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
ln -sf rootfs.ext2 /home/rpi2/Documents/buildroot-2016.11/output/images/rootfs.e
xt4
>>>   Executing post-image script board/raspberrypi3/post-image.sh
Version: Linux version 4.4.21-v7 (rpi2@ubuntu) (gcc version 5.4.0 (Buildroot 201
6.11) ) #1 SMP Tue Dec 20 23:15:13 CET 2016
DT: y
DDT: y
283x: n
vfat(boot.vfat): adding file 'bcm2710-rpi-3-b.dtb' as 'bcm2710-rpi-3-b.dtb' ...
vfat(boot.vfat): adding file 'rpi-firmware/bootcode.bin' as 'rpi-firmware/bootco
de.bin' ...
vfat(boot.vfat): adding file 'rpi-firmware/cmdline.txt' as 'rpi-firmware/cmdline
.txt' ...
vfat(boot.vfat): adding file 'rpi-firmware/config.txt' as 'rpi-firmware/config.t
xt' ...
vfat(boot.vfat): adding file 'rpi-firmware/fixup.dat' as 'rpi-firmware/fixup.dat
' ...
vfat(boot.vfat): adding file 'rpi-firmware/start.elf' as 'rpi-firmware/start.elf
' ...
vfat(boot.vfat): adding file 'rpi-firmware/overlays' as 'rpi-firmware/overlays'
...
vfat(boot.vfat): adding file 'kernel-marked/zImage' as 'kernel-marked/zImage' ..
.
hdimage(sdcard.img): adding partition 'boot' (in MBR) from 'boot.vfat' ...
hdimage(sdcard.img): adding partition 'rootfs' (in MBR) from 'rootfs.ext4' ...
hdimage(sdcard.img): writing MBR
rpi2@ubuntu:~/Documents/buildroot-2016.11$
```

Fig. 194: Building image containing EPICS with Buildroot.

As a result (Fig. 195) an image is created ("sdcard.img" file), that should be copied to our microsd card.

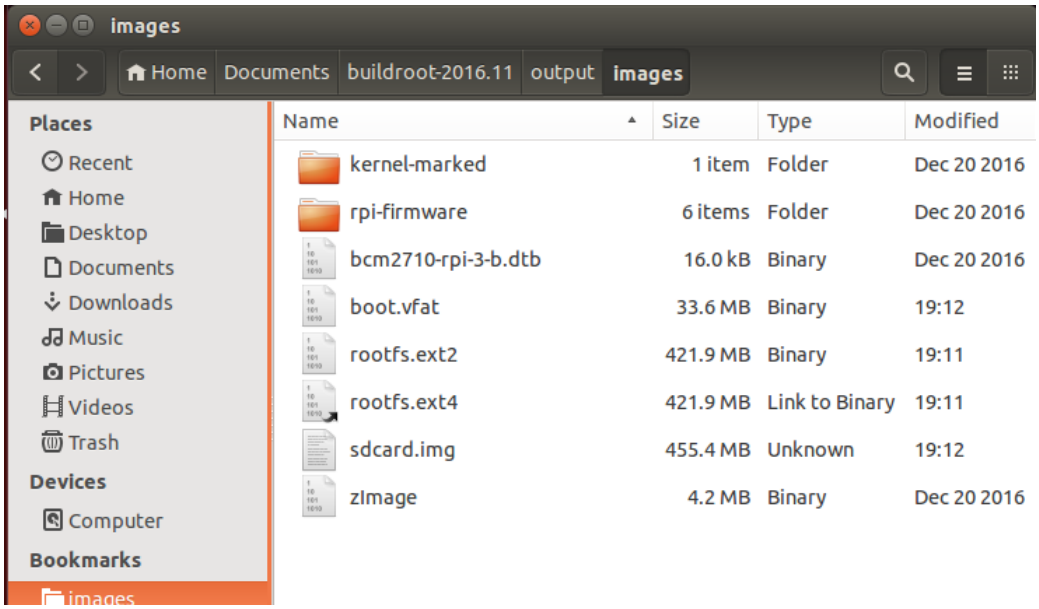


Fig. 195: Buildroot folder containing Linux image file.

Copy the image to microsd card: right click in “sdcard.img”, choose “Open with Disk Image Writer”. Select destination “SD Card Reader” and click “Start restoring”.

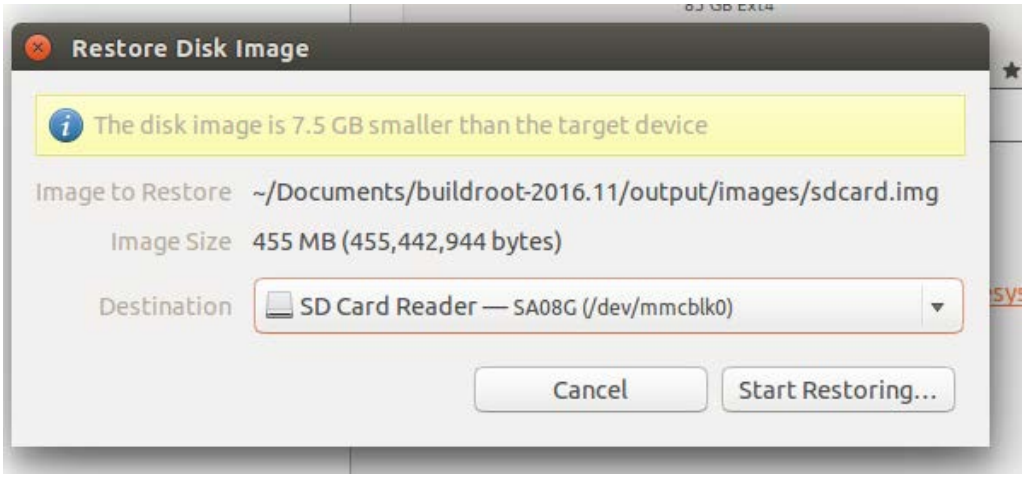


Fig. 196: Dumping the “.iso” image into microsd card.

A directory structure is created in 422MB volume (second partition of the microsd card). Both folders inside “rpi3ov2017” appears as well (see Fig. 197).

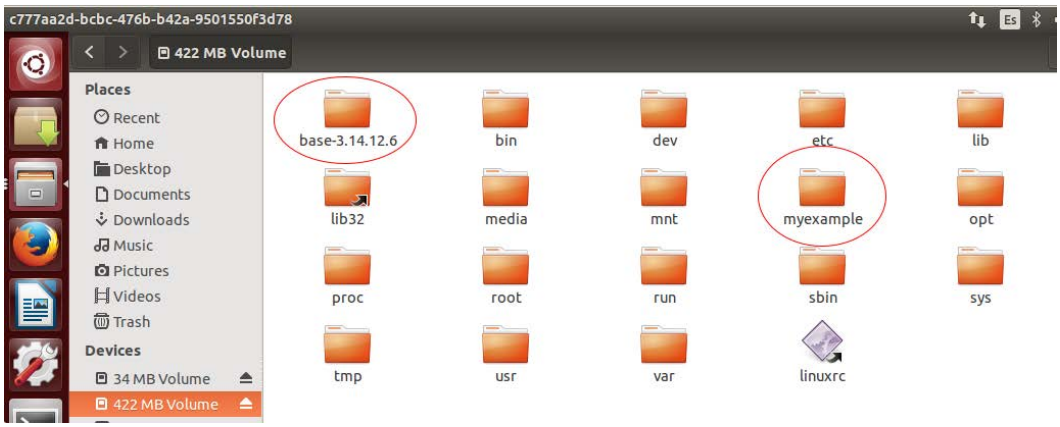


Fig. 197: Second partition including EPICS folders.

7.6 Running EPICS softIOC on Raspberry Pi

The following steps have to be executed from the Raspberry Pi. We are going to run the example we generated before, simulate multiple EPICS records and process variables (PVs), and try to read their values.

Place the microsd card into the rpi and boot it up.

Change the active directory to "*iocBoot/oicmyexample"*, using this command:

```
cd /myexample/iocBoot/iocmyexample
```

Run the example "st.cmd":

```
../../bin/linux-arm/myexample ./st.cmd
```

After the IOC is started, EPICS prompt will appear: "epics>" (see Fig. 198)

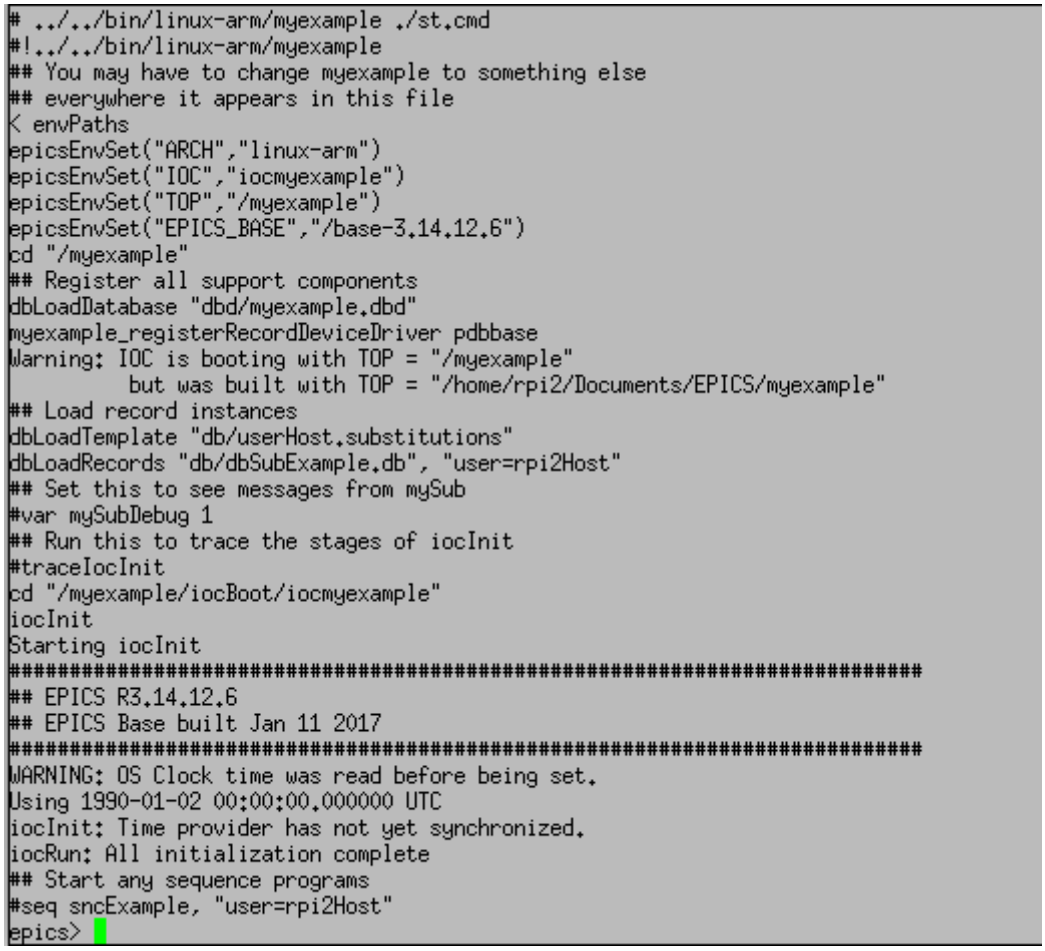


Fig. 198: Running EPICS IOC on Raspberry Pi.

Warning: Maybe a “Warning” like the one shown in previous screen will appear. It means that clock has not been set, so Buildroot settings must be reviewed. The package “NTP” must be included, and an additional settings must be changed. More information is available here

To print the names of records in the run time database, we can use command “*dbl*”:

```
epics> dbl
```

A list of all variables (PVs) created by our example is shown:

```
rpi2Host:ai1
rpi2Host:ai2
rpi2Host:ai3
rpi2Host:aiExample
rpi2Host:aiExample1
rpi2Host:aiExample2
rpi2Host:aiExample3
rpi2Host:aSubExample
```

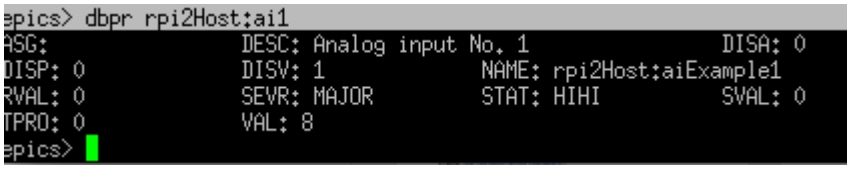
```
rpi2Host:calc1
rpi2Host:calc2
rpi2Host:calc3
rpi2Host:calcExample
rpi2Host:calcExample1
rpi2Host:calcExample2
rpi2Host:calcExample3
rpi2Host:compressExample
rpi2Host:subExample
rpi2Host:xxxExample
epics>
```

As we can see, our host name is “rpi2Host”.

To print all fields of a record, (“ai1”, for example) use “*dbpr”* command, followed by the record name (host:record), i.e. “rpi2Host:ai1”

```
epics> dbpr rpi2Host:ai1
```

This is the result:



Executing the same command several times, we will see that the value of the PV (VAL field) changes from 0 to 9, so the IOC is running OK.

8 MONITORING PVS CREATED BY EXAMPLE SOFTIOC

8.1 Monitoring PVs by connecting to rpi through SSH

Due to previous work, we are running an EPICS server (softIOC) on the rpi. This server is giving values to several process variables (PVs). Now we are going to read the values of these PVs accessing via SSH to the rpi. So, we are going to use the rpi both as client and server.

There will be an EPICS server running in the rpi (using Putty to connect) and we will monitor it through a SSH connection to rpi too (if you prefer to use several SSH connections instead of Putty, you can do it, of course).

To view in detail the necessary settings to establish SSH connection, see annex "Configuring SSH".

Open a terminal window in Ubuntu 14 virtual machine, and type:

```
ssh root@192.168.0.199
```

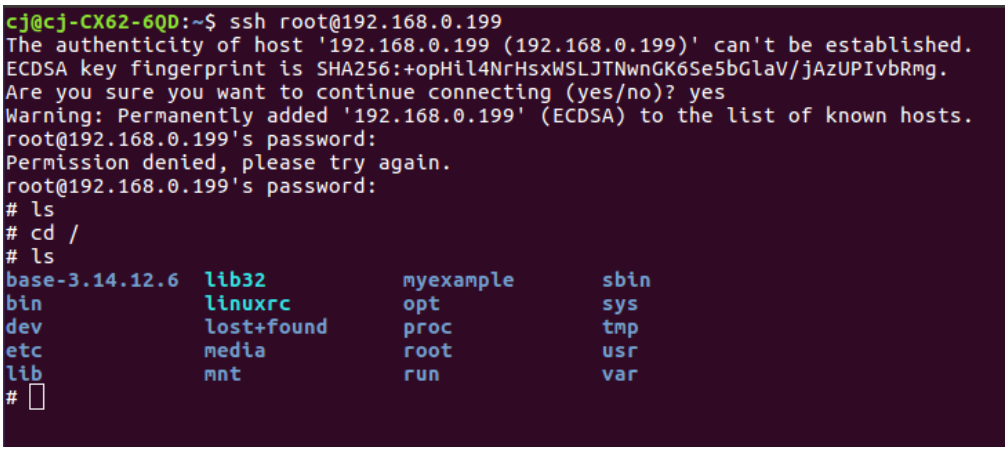


Fig. 199: Connecting to Raspberry Pi from pc using SSH.

So we are connecting using "root" as user and (see Fig. 199) to IP address 192.168.0.199. Change this address for the IP in your system if necessary.

Remember to define the environment variables in every SSH connection to the rpi, using these commands:

```
export EPICS_BASE=/base-3.14.12.6
export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
```

Environment variables (Fig. 200) will be set like these:

```
# export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
# env
USER=root
SHLVL=1
OLDPWD=/bin
HOME=/root
PAGER=/bin/more
PS1=#
LOGNAME=root
TERM=vt100
EPICS_BASE=/base-3.14.12.6
PATH=/bin:/sbin:/usr/bin:/usr/sbin://base-3.14.12.6/bin/linux-arm
SHELL=/bin/sh
PWD=/
EDITOR=/bin/vi
```

Fig. 200: Changing environment variables in Raspberry Pi.

Warning: it is important to configure these variables. If not, commands for monitoring or changing PVs values will not be executed. If that is the case, an error "*-sh: caget: not found*" will be shown when a command is to be executed ("*caget*" for example). See annex "Solving PATH issue".

Obtain the value of PV "ai1", typing from the established SSH session this command:

```
# caget rpi2Host:ai1
```

If we repeat it several times (Fig. 201), we will see how the value (VAL) of the PV ("ai1") changes, which confirms that we are reading the values of the PV the right way.

```
# caget rpi2Host:ai1
rpi2Host:ai1                   8
#
# caget rpi2Host:ai1
rpi2Host:ai1                   0
# caget rpi2Host:ai1
rpi2Host:ai1                   2
# caget rpi2Host:ai1
rpi2Host:ai1                   4
# caget rpi2Host:ai1
rpi2Host:ai1                   5
# caget rpi2Host:ai1
rpi2Host:ai1                   6
#
```

Fig. 201: Reading a PV using caget.

We can also run the command "camonitor" (Fig. 202) to monitor PV's value continuously:

```
# camonitor rpi2Host:ai1
```

```
# camonitor rpi2Host:ai1
rpi2Host:ai1                   2017-01-25 10:08:41.273747 9 HIHI MAJOR
rpi2Host:ai1                   2017-01-25 10:08:42.273731 0 LOLO MAJOR
rpi2Host:ai1                   2017-01-25 10:08:43.273737 1 LOLO MAJOR
rpi2Host:ai1                   2017-01-25 10:08:44.273722 2 LOLO MAJOR
rpi2Host:ai1                   2017-01-25 10:08:45.273735 3 LOW MINOR
rpi2Host:ai1                   2017-01-25 10:08:46.273725 4 LOW MINOR
rpi2Host:ai1                   2017-01-25 10:08:47.273725 5
rpi2Host:ai1                   2017-01-25 10:08:48.273725 6 HIGH MINOR
rpi2Host:ai1                   2017-01-25 10:08:49.273741 7 HIGH MINOR
rpi2Host:ai1                   2017-01-25 10:08:50.273726 8 HIHI MAJOR
rpi2Host:ai1                   2017-01-25 10:08:51.273740 9 HIHI MAJOR
rpi2Host:ai1                   2017-01-25 10:08:52.273726 0 LOLO MAJOR
rpi2Host:ai1                   2017-01-25 10:08:53.273742 1 LOLO MAJOR
rpi2Host:ai1                   2017-01-25 10:08:54.273723 2 LOLO MAJOR
^C
#
```

Fig. 202: Monitoring PV's value using camonitor.

Next screenshot (Fig. 203) shows EPICS server running on a remote session using SSH (on the left) and another connection using Putty and USB FTDI cable (to the right). Session on the right shows the use of *caget* to read the value of a PV: "*rpi2Host:ai1*"

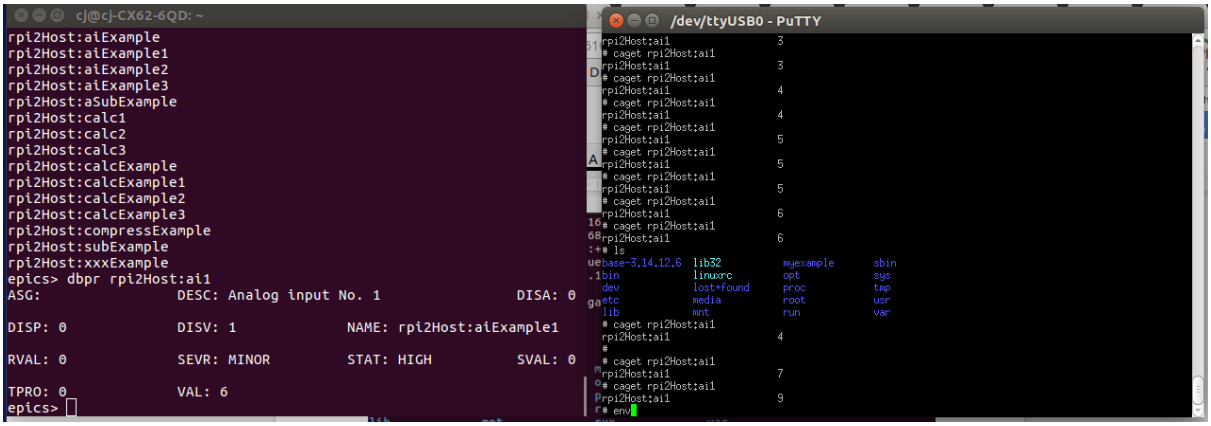


Fig. 203: Simultaneous connection to Raspberry Pi using SSH and Putty.

8.1.1 Summary of the commands to run on rpi:

Commands to run on the rpi to boot EPICS server:

```
# export EPICS_BASE=/base-3.14.12.6
# export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
# cd /myexample/iocBoot/iocmyexample
# ../../bin/linux-arm/myexample ./st.cmd
```

Commands to run from pc to use SSH and monitor EPICS PVs (change IP if necessary in the first command):

```
$ ssh root@192.168.0.199
# export EPICS_BASE=/base-3.14.12.6
# export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
# caget rpi2Host:ai1
```

8.2 Monitoring PVs from Ubuntu virtual machine

Previous step 8.1 showed us how to monitor PV's values, generating and reading the values from the same rpi (rpi working as both server and client). We are going to monitor now those values from our Ubuntu virtual machine, where as you remember, EPICS is also installed. When we did the cross-compilation process, two set of binaries were created in two separate folders, one for "linux-arm" architecture, and the other for "linux-x86". We will use the contents of the last folder.

Connect the computer to the rpi using a Putty session, boot up the rpi, and run the IOC on the rpi (Fig. 204), executing these commands:

```
# export EPICS_BASE=/base-3.14.12.6
# export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
# cd /myexample/iocBoot/iocmyexample
# ../../bin/linux-arm/myexample ./st.cmd
```

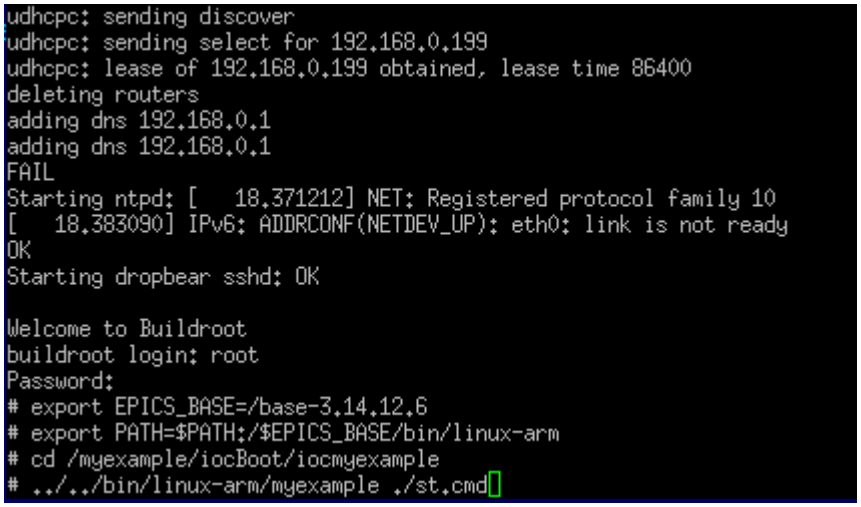


Fig. 204: Running example IOC on Raspberry Pi.

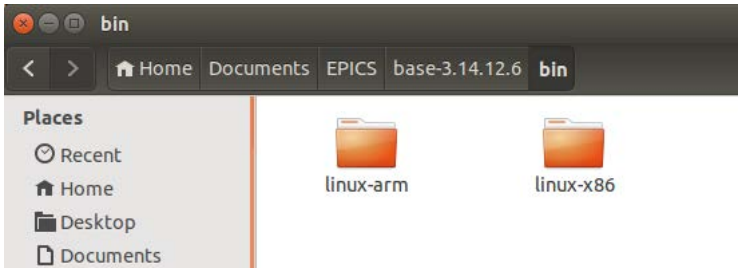
IOC server is up and running on the rpi:

```
#!../../bin/linux-arm/myexample
## You may have to change myexample to something else
## everywhere it appears in this file
< envPaths
epicsEnvSet("ARCH","linux-arm")
epicsEnvSet("IOC","iocmyexample")
epicsEnvSet("TOP","/myexample")
epicsEnvSet("EPICS_BASE","/base-3.14.12.6")
cd "/myexample"
## Register all support components
dbLoadDatabase "dbd/myexample.dbd"
myexample_registerRecordDeviceDriver pdbbase
Warning: IOC is booting with TOP = "/myexample"
         but was built with TOP = "/home/rpi2/Documents/EPICS/myexample"
## Load record instances
dbLoadTemplate "db/userHost.substitutions"
dbLoadRecords "db/dbSubExample.db", "user=rpi2Host"
## Set this to see messages from mySub
#var mySubDebug 1
## Run this to trace the stages of iocInit
#traceIocInit
cd "/myexample/iocBoot/iocmyexample"
iocInit
Starting iocInit
############################################################################
## EPICS R3.14.12.6
## EPICS Base built Jan 11 2017
############################################################################
iocRun: All initialization complete
## Start any sequence programs
#seq sncExample, "user=rpi2Host"
epics>
```

Fig. 205: SoftIOC server running on Raspberry Pi.

We are going to use now the rpi as an EPICS IOC server, and Ubuntu 14 virtual machine as an EPICS client.

That value of the environment variable "PATH" has to be different now, because we will run EPICS on Ubuntu, not on the rpi. Remember that in the cross-compilation process, two folders were created inside "bin" directory, one for "arm" and the other for "x86". To monitor the PVs from Ubuntu, we have to use the binaries (caget, camonitor) included into that "linux-x86" folder. The former PATH used the binaries from "linux-arm" folder.



From Ubuntu 14 virtual machine, set the environment variables EPICS_BASE and PATH to specify the path where EPICS (base and binaries) is located into our Ubuntu virtual machine:

```
$ export EPICS_BASE=~/Documents/EPICS/base-3.14.12.6
$ export PATH=$PATH:/$EPICS_BASE/bin/linux-x86
```

Check that the values have been set using "echo" command (see Fig. 206):

```
$ echo $EPICS_BASE
$ echo $PATH
```

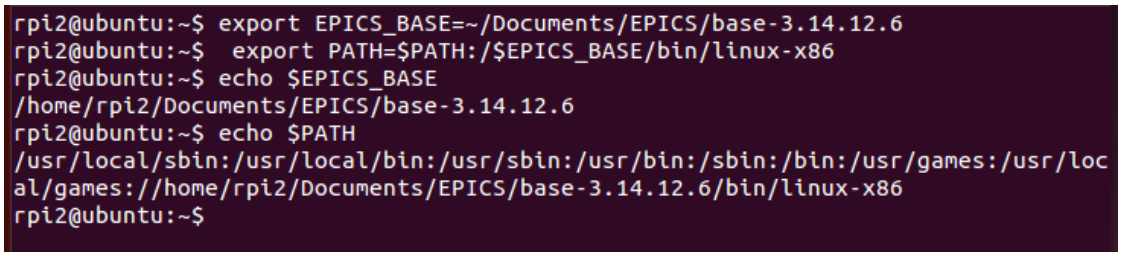


Fig. 206: Setting EPICS environment variables in client (Ubuntu).

If we try to obtain the values of PVs, an error is shown, PV is not found (Fig. 207):

```
$ caget rpi2Host:ai1
$ camonitor rpi2Host:ai1
```

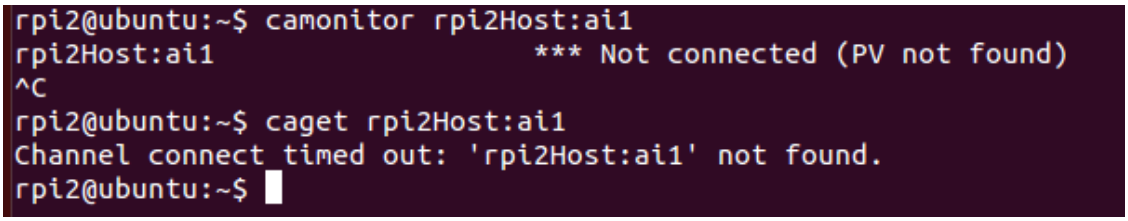


Fig. 207: Error reading PVs from client (Ubuntu).

This error is due to the fact that right now, the rpi and the computer have different IP ranges, and EPICS is not able to find a PV if it is running on a different subnet (for security reasons).

If we look at the IP address assigned to Ubuntu virtual machine (using "*ifconfig*" command), it is 192.168.120.132, nothing to do with the IP assigned to the rpi: 192.168.0.199.
This can be solved setting the value of a certain EPICS environment variable. We will set the value of that variable to the IP address of the EPICS IOC server (192.168.0.199 in our example) using these commands:

```
$ export EPICS_CA_AUTO_ADDR_LIST=NO
$ export EPICS_CA_ADDR_LIST=192.168.0.199
```

So, once you have executed your Ubuntu 14 virtual machine, run these commands:

```
$ export EPICS_BASE=~/Documents/EPICS/base-3.14.12.6
$ export PATH=$PATH:/$EPICS_BASE/bin/linux-x86
$ export EPICS_CA_AUTO_ADDR_LIST=NO
$ export EPICS_CA_ADDR_LIST=192.168.0.199
```

Check the environment variables using "*echo*" command (Fig. 208):

```
$ echo $EPICS_BASE
$ echo $PATH
$ echo $ EPICS_CA_AUTO_ADDR_LIST
$ echo $ EPICS_CA_ADDR_LIST
```

```
rpi2@ubuntu:~$ echo $EPICS_BASE
/home/rpi2/Documents/EPICS/base-3.14.12.6
rpi2@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games://home/rpi2/Documents/EPICS/base-3.14.12.6/bin/linux-x86
rpi2@ubuntu:~$ echo $EPICS_CA_AUTO_ADDR_LIST
NO
rpi2@ubuntu:~$ echo $EPICS_CA_ADDR_LIST
192.168.0.199
rpi2@ubuntu:~$
```

Fig. 208: Setting and checking environment variable EPICS_CA_ADDR_LIST.

It is time to monitor IOC PVs running on the rpi, but from the virtual machine running on the computer. Use "camonitor" for that purpose in Ubuntu terminal window (Fig. 209):

```
$ camonitor rpi2Host:ai1
```

```
rpi2@ubuntu:~$ camonitor rpi2Host:ai1
rpi2Host:ai1                   2017-02-05 13:07:12.996101 5
rpi2Host:ai1                   2017-02-05 13:07:13.996110 6 HIGH MINOR
rpi2Host:ai1                   2017-02-05 13:07:14.996106 7 HIGH MINOR
rpi2Host:ai1                   2017-02-05 13:07:15.996102 8 HIHI MAJOR
rpi2Host:ai1                   2017-02-05 13:07:16.996107 9 HIHI MAJOR
rpi2Host:ai1                   2017-02-05 13:07:17.996101 0 LOLO MAJOR
rpi2Host:ai1                   2017-02-05 13:07:18.996107 1 LOLO MAJOR
rpi2Host:ai1                   2017-02-05 13:07:19.996100 2 LOLO MAJOR
^C
rpi2@ubuntu:~$
```

Fig. 209: Monitoring PV from client (Ubuntu).

PV values can be read without a problem, as we can see.

8.3 Steps to include environment variables and PATH after Buildroot build process

Idea from: http://lists.busybox.net/pipermail/buildroot/2014-January/086603.html

> If you want to export variables globally, consider adding it to
> /etc/profile (or better create a /etc/profile/<whatever>.sh)
>
> --
> Bye, Peter Korsgaard

Every time we boot up the rpi, to be able to run the EPICS example IOC, we must:

1.- To manually enter the path to EPICS base in an environment variable.
2.- To include into PATH environment variable the path to where linux-arm binaries can be found.

We achieve those requirements by executing these commands in the rpi:

```
# export EPICS_BASE=/base-3.14.12.6
# export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
```

If we want to do this automatically, let's put those commands into *"/etc/profile"* file, located in the second partition (422MB) that was generated in the build process using Buildroot. Extract the card from the rpi, put it in your computer.
For testing purposes, edit the file */etc/profile*, whose original content is shown in Fig. 210.

```
export PATH=/bin:/sbin:/usr/bin:/usr/sbin

if [ "$PS1" ]; then
        if [ "`id -u`" -eq 0 ]; then
                export PS1='# '
        else
                export PS1='$ '
        fi
fi

export PAGER='/bin/more '
export EDITOR='/bin/vi'

# Source configuration files from /etc/profile.d
for i in /etc/profile.d/*.sh ; do
        if [ -r "$i" ]; then
                . $i
        fi
        unset i
done
```

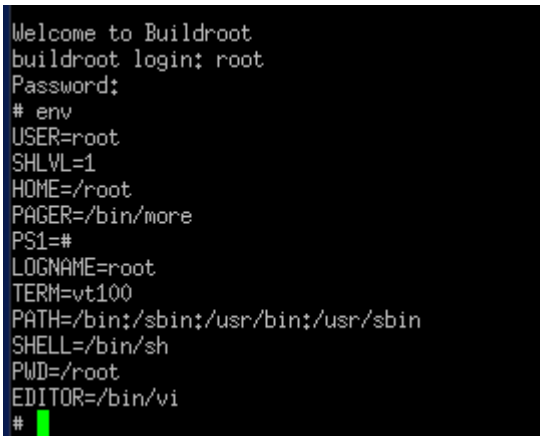
Fig. 210: Original content of /etc/profile file.

Remove the first line and add these two (use "*sudo nano profile*", for example):

```
export EPICS_BASE=/base-3.14.12.6
export PATH=/bin:/sbin:/usr/bin:/usr/sbin:/$EPICS_BASE/bin/linux-arm
```

Save the file, put the card into the rpi, and boot it up.

These were the original environment variables before the changes (type "*env")* after boot up the rpi:



And these are now the environment variables after the changes (Fig. 211). Variables "EPICS_BASE" and the path to "bin/linux-arm" are already defined:

Fig. 211: EPICS environment variables after booting Raspberry Pi.

To run the example IOC now, only two commands must be entered, instead of four:

```
# cd /myexample/iocBoot/iocmyexample
# ../../bin/linux-arm/myexample ./st.cmd
```

NOTE: if we use dynamic libraries instead of static ones, environment variable LD_LIBRARY_PATH must be set too.

8.4 Steps to include environment variables and PATH before Buildroot build process

In the previous step, we built the image using Buildroot and then, afterwards, we set the environment variables. It would be more efficient if those environment variables and PATH were set during the build process. That way they will be automatically generated during the compilation of our Linux image.

The idea is to create a file "*/etc/profile*" that, using script "*post-build.sh*", will be copied automatically into our Linux image.

It is the same process that we used before to add "*interfaces*" and "*wpa_supplicant.conf*" files to configure wifi en the rpi.

Create a new file called *"profile"* inside "*buildroot-2016.11/board/raspberrypi3*" folder. The first lines of the file define the path to EPICS_BASE and the path to EPICS binaries for "linux-arm" architecture:

buildroot-2016.11/board/raspberrypi3/profile

```
#Define EPICS_BASE and binaries PATH
export EPICS_BASE=/base-3.14.12.6
export PATH=/bin:/sbin:/usr/bin:/usr/sbin:/$EPICS_BASE/bin/linux-arm

if [ "$PS1" ]; then
     if [ "`id -u`" -eq 0 ]; then
           export PS1='# '
     else
           export PS1='$ '

     fi
fi

export PAGER='/bin/more '
export EDITOR='/bin/vi'

# Source configuration files from /etc/profile.d
for i in /etc/profile.d/*.sh ; do
     if [ -r "$i" ]; then
           . $i
     fi
     unset i
done
```

This is the content of "*buildroot-2016.11/board/raspberrypi3"* folder (see Fig. 212).

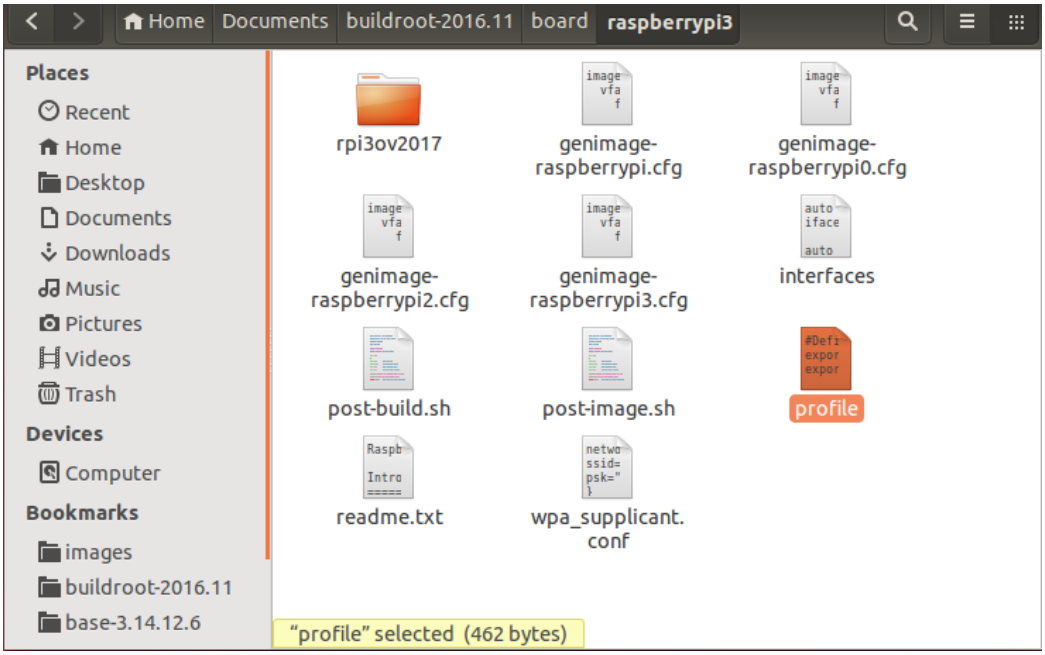


Fig. 212: Contents of buildroot-2016.11/board/raspberrypi3 folder.

Let's edit now the file "*board/raspberrypi3/post-build.sh*" and add this line at the end of the file. It will copy the file "*profile*" to the resulting Linux image:

```
cp board/raspberrypi3/profile ${TARGET_DIR}/etc/profile
```

This is the "*post-build.sh"* file after all the changes:

post-build.sh ×

```
#!/bin/sh

set -u
set -e

# Add a console on tty1
if [ -e ${TARGET_DIR}/etc/inittab ]; then
    grep -qE '^tty1::' ${TARGET_DIR}/etc/inittab || \
        sed -i '/GENERIC_SERIAL/a\
tty1::respawn:/sbin/getty -L  tty1 0 vt100 # HDMI console' ${TARGET_DIR}/etc/
inittab
fi

#puesto por mi para configurar wifi en rpi3
cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf
cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf ${TARGET_DIR}/etc/wpa_supplicant.conf
cp board/raspberrypi3/profile ${TARGET_DIR}/etc/profile
```

Fig. 213: Contents of post-build.sh.

Execute "*make*" in Buildroot.

Dump the image file into the microsd card, put it into the rpi, boot the rpi, and start a Putty session.

First of all, check (see Fig. 214) using "*env*" command that environment variables EPICS_BASE and PATH are correct. PATH should include the reference to "/bin/linux-arm".

```
# env
USER=root
SHLVL=1
HOME=/root
PAGER=/bin/more
PS1=#
LOGNAME=root
TERM=vt100
EPICS_BASE=/base-3.14.12.6
PATH=/bin:/sbin:/usr/bin:/usr/sbin://base-3.14.12.6/bin/linux-arm
SHELL=/bin/sh
PWD=/root
EDITOR=/bin/vi
#
```

Fig. 214: Checking environment variables after Raspberry Pi boot.

8.5 Monitoring PVs using Control System Studio (CSS)

8.5.1 Control System Studio description

Control System Studio (CSS) is a collection of Eclipse-based tools, which allow you to monitor control systems on a large scale. It is the product of collaboration between different laboratories and universities all over the world. Its home page is: http://controlsystemstudio.org/

Within the client/server EPICS architecture, it falls into the client category, or OPI.

There are tools for managing alarms, archiving data, several operator interfaces and tools for system diagnostics.

It allows to read and display visually the values of the different process variables (PVs) of our EPICS systems.

Unlike the "caget" or "camonitor" commands used in the previous sections, that showed us the values of the PVs in text mode, CSS give us access to a complete graphical environment.

Fig. 215: Control System Studio Logo.

8.5.2 Installing and configuring Control System Studio

Different versions of Control System Studio can be downloaded from (Fig. 216):
https://ics-web.sns.ornl.gov/css/products.html

Compare CSS Products

CSS is a collection of Eclipse Plug-ins which then get combined into a *Product*.

We offer

- A Basic EPICS product where the startup screen guides you through the EPICS Channel Access settings.
- A product pre-configured for the SNS Office network with support for the SNS ELog, Channel Access Gateway, Oracle Database.
- The SNS Control Room uses the same SNS Office product with optional features like Alarm System GUI added from the update site.
- Web OPI, an online version of BOY. Installed into for example Tomcat, it allows users to access their BOY display files from most web browsers phones.

Refer to the this Comparison on the Source Forge CSS Wiki for the content of each product, or check the sites of other CSS collaborators for their p

Some products are available as binaries for various operating systems. The source code down-load includes the sources for all products offered by SourceForge repository includes these plus sources used by other CSS collaborators.

Version	Basic EPICS	SNS Office	Web OPI	Comments
Nightly	For the adventurous, https://ics-web.sns.ornl.gov/css/nightly updates each night if the source code has changed. Guaranteed to include the latest bugs. If even more adventurous, use the "Help", "Install.." menu to add the display builder from `http://ics-web.sns.ornl.gov/css/display.builder` See https://github.com/kasemir/org.csstudio.display.builder/blob/master/Readme.md			
4.1.1	MS Windows MS Windows, 64 bit Mac OS X 64 bit RH Linux x86 RH Linux x86, 64 bit	MS Windows MS Windows, 64 bit Mac OS X 64 bit RH Linux x86 RH Linux x86, 64 bit		Github Sources as of 2015-07-13
4.1.0	MS Windows MS Windows, 64 bit	MS Windows MS Windows, 64 bit		

Fig. 216: Control System Studio (CSS) download web page.

If we choose the wrong version, and try to run a 64 bit program into our 32 bits virtual machine, an error message will appear stating that the program will not be executed. Use the following command to check if your CSS version is 32 or 64 bits:

```
$ file css
```

```
rpi1@ubuntu:~/Documents/EPICS/basic-epics-4.1.1$ file css
css: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linked (u
ses shared libs), for GNU/Linux 2.4.0, not stripped
rpi1@ubuntu:~/Documents/EPICS/basic-epics-4.1.1$
```

In this case, the version is intended for 64 bits (x86-64).

Download BASE version 4.1.1 32 bits (130MB), “basic-epics-4.1.1-linux.gtk.x86.zip” file, and unpack its contents to EPICS folder.

Fig. 217: Unpacking CSS to EPICS folder.

A folder named “basic-epics-4.1.1” appears, and its contents are:

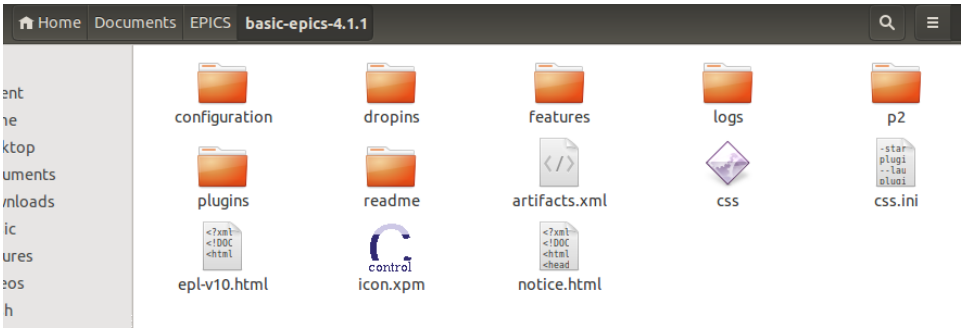


Fig. 218: Contents of CSS folder.

Check that CSS file is a 32 bits version, using:

```
$ file css
```

```
rpi1@ubuntu:~/Documents/EPICS/basic-epics-4.1.1$ file css
css: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamically link
ed (uses shared libs), for GNU/Linux 2.2.5, not stripped
rpi1@ubuntu:~/Documents/EPICS/basic-epics-4.1.1$
```

To execute CSS, execute this command:

```
./css &
```

A Java related error is shown:

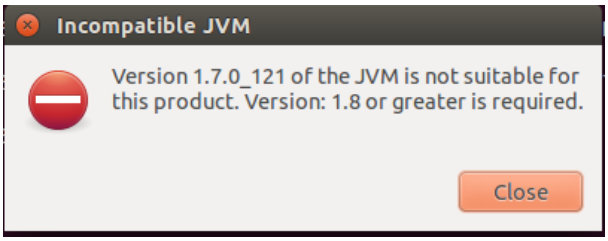


Fig. 219: Can not run CSS, wrong Java version.

CSS 4.1.1 needs Java JRE version 1.8, but Ubuntu 14.04 installs by default Java JRE version 1.7. Annex **Installing Java JRE 1.8 in Ubuntu 14.04 32 bits** shows in detail how to solve this issue.

As a temporary option, we are going to install another CSS version, 3.2.16. This version works fine with Java 1.7, and CSS menus and functionalities are the same in both versions.

Download the installation file from the same site that hosts CSS 4.1.1 version: (https://ics-web.sns.ornl.gov/css/products.html), the file to download is: "**epics_css_3.2.16-linux.gtk.x86.zip**"

Unpack it to EPICS folder, creating a new folder named "*CSS_EPICS_3.2.16*".

Check permissions of the file "css" (Fig. 220) using this command:

```
ls -lasg
```

```
rpi1@ubuntu:~/Documents/EPICS/CSS_EPICS_3.2.16$ file css
css: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), dynamically link
ed (uses shared libs), for GNU/Linux 2.2.5, not stripped
rpi1@ubuntu:~/Documents/EPICS/CSS_EPICS_3.2.16$ ls -lasg
total 504
  4 drwxr-xr-x  9 rpi1   4096 Apr  9  2014 .
  4 drwxrwxr-x  6 rpi1   4096 Feb  8 17:38 ..
  4 drwxr-xr-x  2 rpi1   4096 Apr  9  2014 about_files
  4 -rw-r--r--  1 rpi1    577 Jan 11  2012 about.html
 68 -rw-r--r--  1 rpi1  66606 Apr  9  2014 artifacts.xml
  4 drwxr-xr-x  4 rpi1   4096 Apr  9  2014 configuration
 64 -rwxr-xr-x  1 rpi1  62814 Jan 11  2012 css
  4 -rw-r--r--  1 rpi1    358 Apr  9  2014 css.ini
  4 drwxr-xr-x  2 rpi1   4096 Apr  9  2014 dropins
  4 -rw-r--r--  1 rpi1     59 Feb  8  2012 .eclipseproduct
 20 -rw-r--r--  1 rpi1  16536 Feb  8  2012 epl-v10.html
  4 drwxr-xr-x 16 rpi1   4096 Apr  9  2014 features
  4 -rw-r--r--  1 rpi1   2592 Apr  9  2014 icon.xpm
260 -rwxr-xr-x  1 rpi1 266168 Jan 11  2012 libcairo-swt.so
 12 -rw-r--r--  1 rpi1   8951 Feb  8  2012 notice.html
  4 drwxr-xr-x  4 rpi1   4096 Apr  9  2014 p2
 32 drwxr-xr-x 13 rpi1  28672 Apr  9  2014 plugins
  4 drwxr-xr-x  2 rpi1   4096 Apr  9  2014 readme
rpi1@ubuntu:~/Documents/EPICS/CSS_EPICS_3.2.16$
```

Fig. 220: CSS file permissions.

Execute the graphical environment Control System Studio CSS using this command:

```
./css &
```

There is no need to set any EPICS environment variables, it is necessary only to set the IP address of the Raspberry Pi where our EPICS server is running.

Warning: If a permission related error is shown when executing the previous command, “css” file permissions must be modified, using this command:

chmod +x css

8.5.3 Configuring CSS

Remember that we executed CSS program using the command:

```
./css &
```

The CSS environment loads, and it asks us to define a folder for our workspace:

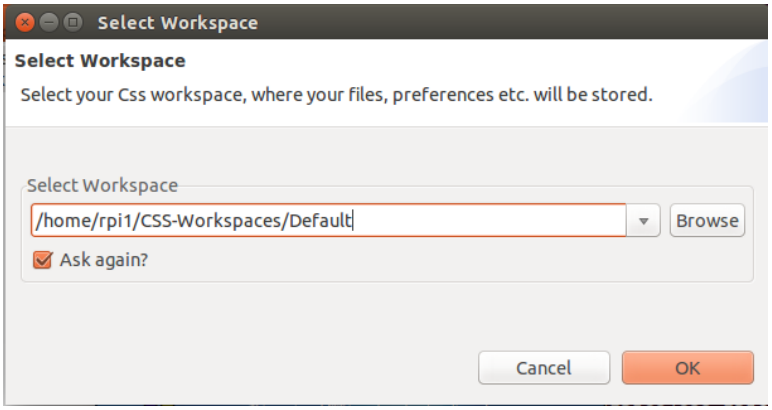


Fig. 221: Defining CSS workspace.

Create a folder named “workspacecss” into “Documents/EPICS” folder.

Click “OK” and a welcome screen appears, close it.

Fig. 222: CSS welcome screen.

We are going to use as a reference the instructions on:
http://www.smolloy.com/2015/12/control-system-studio-epics-with-an-arduino/

First open the OPI Editor Windows:

Window menu–>Open Perspective–>Other, choose “OPI Editor” (Fig. 223).

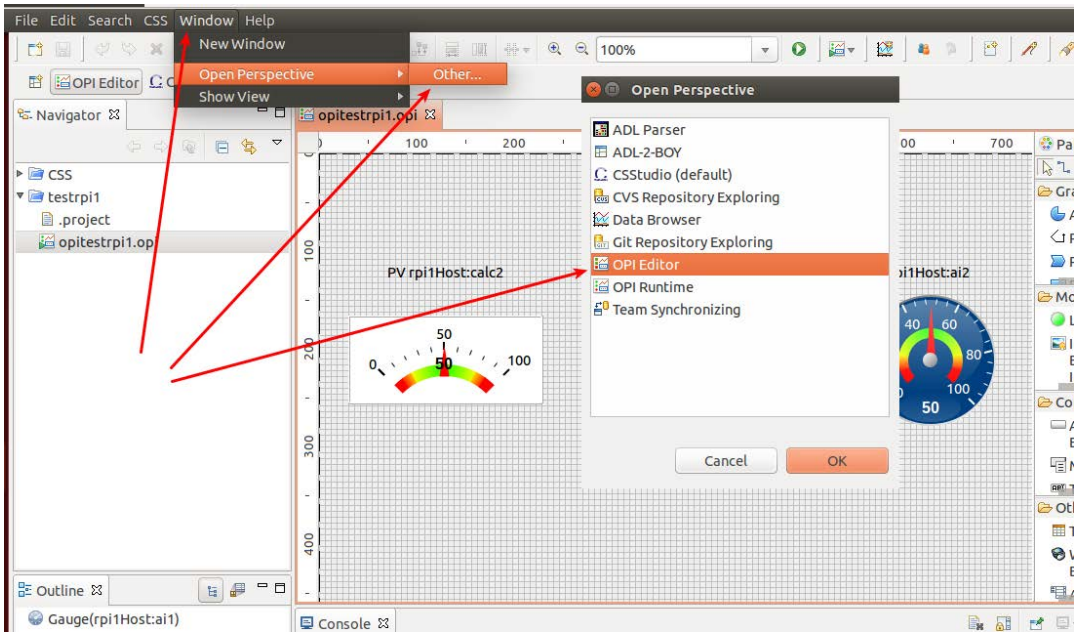


Fig. 223: Choosing OPI editor in CSS.

Time to create a new project: to do this, go to the left panel, “Navigator”, right click in “CSS” and choose “New”–>”Project”.

Then select “General”–>”Project”, click “Next” (Fig. 224). Type the name of your project, and click “Finish”.

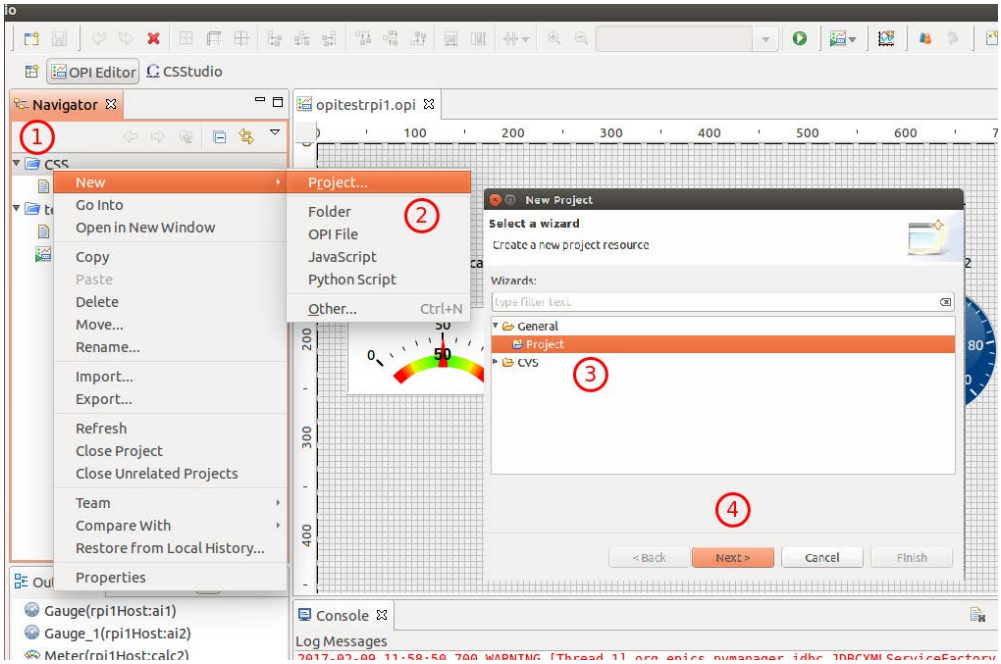


Fig. 224: Creating a new project in CSS.

Now we need to create a space to put our indicators, gauges, etc. An OPI file is the place to do that.

Move to the Navigator window, right click on the name of your project (“testrpi1” in the example here). Click “New” and then on “OPI File”. Type a name (“opitestrpi1” in the example), and click “Finish” (see Fig. 225).

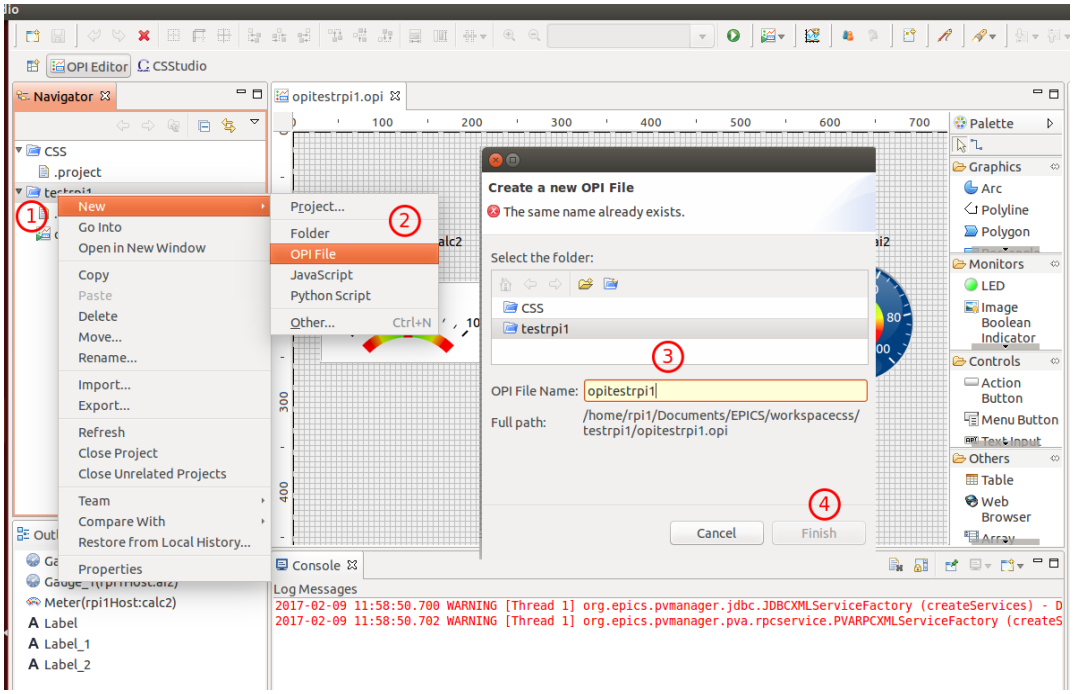


Fig. 225: Creating and OPI file in CSS.

A blank workspace is available now, called “opitestrpi1.opi”. We can click now on any of the indicators located at the right hand side, under “Palette” and “Monitors” category (Fig. 226). For example put a “Gauge”. Next screenshot shows three indicators monitoring three PVs, two of the type “Gauge” and another of the type “Meter”.

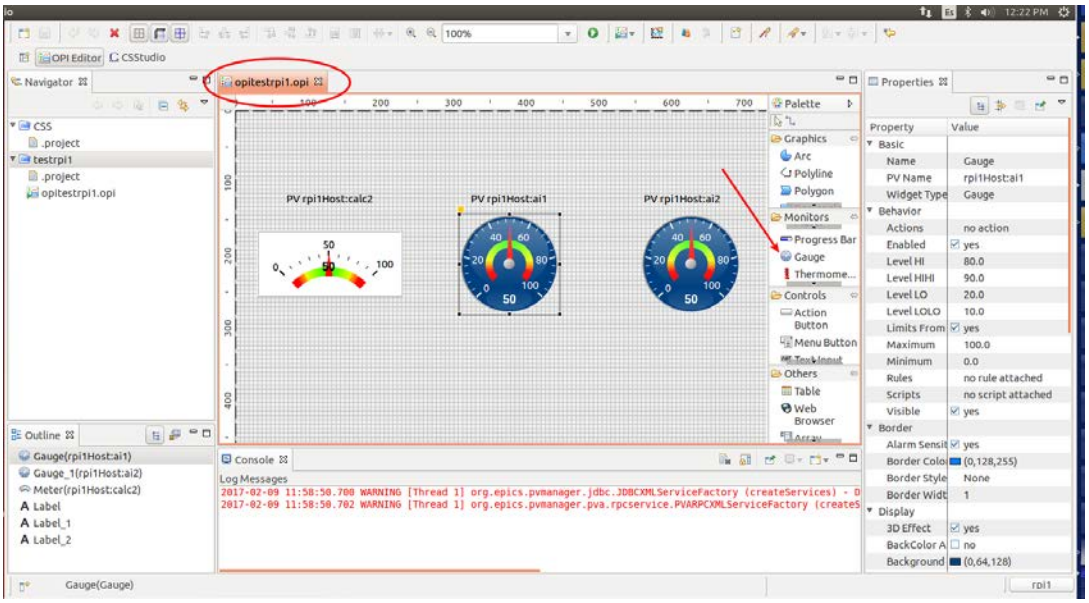


Fig. 226: Adding indicators to the workspace in CSS.

To assign a PV to an indicator and monitor its value, click on the small square at the top left of the indicator (Fig. 227), and type the name of the PV (in our example “rpi1Host:ai1”).

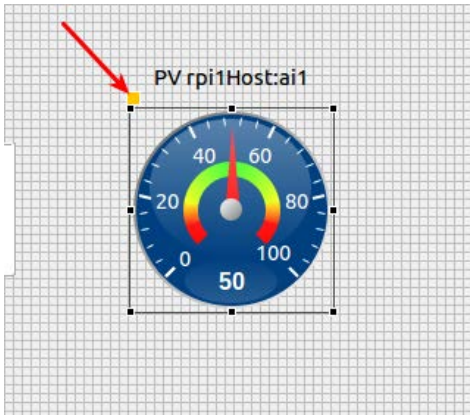


Fig. 227: Assigning a PV to an indicator in CSS.

CSS needs to know the IP address of the EPICS server in order to access the PVs. We have to configure this value into CSS settings. Let’s go to the menu at the top “Edit/Preferences/CSS Core/EPICS”, and uncheck “auto_addr_list” (Fig. 228).
In the field above, “addr_list”, type the IP address of the rpi. Remember that we had to do this because Ubuntu 14 virtual machine and the rpi had IPs in a different range (192.168.120.131 and 192.168.0.199). Click “Apply” and then “OK”.

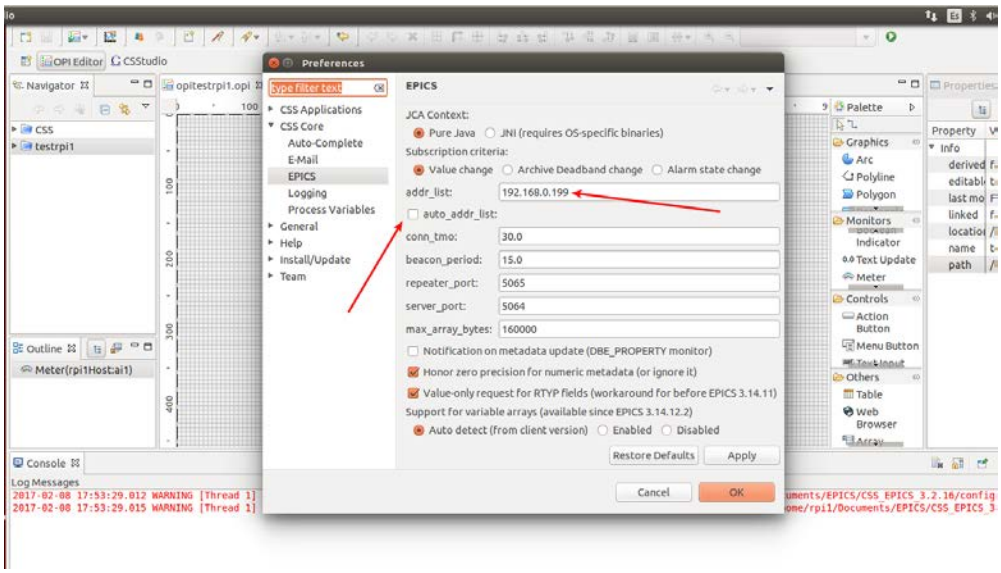
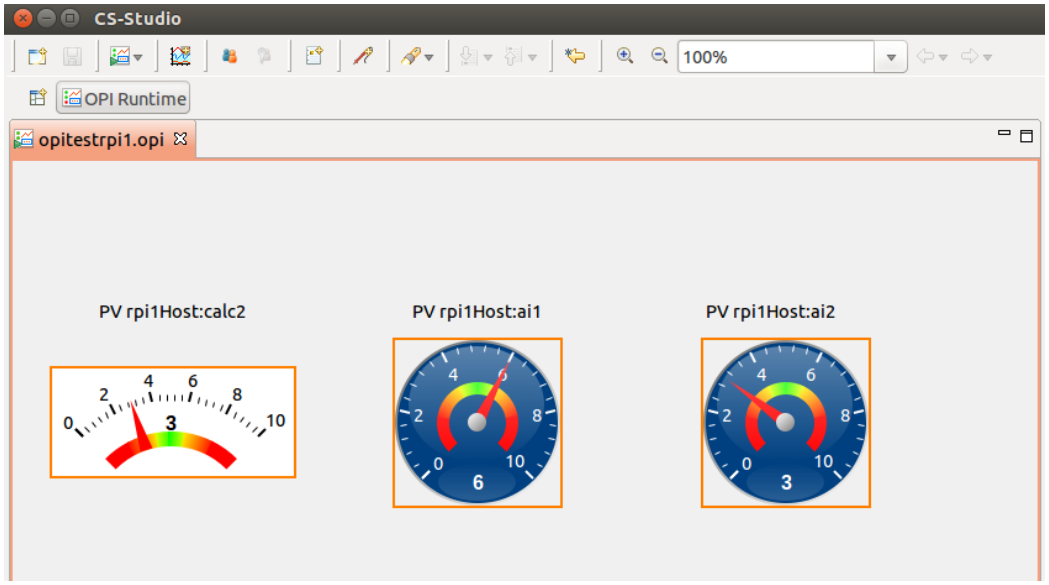


Fig. 228: Setting EPICS server address in CSS.

8.5.4 Testing CSS

To start monitoring the PVs, click “Play” icon at the top  , and see how the indicators change every one or two seconds. They are reading the PVs values, so it is working (Fig. 229).



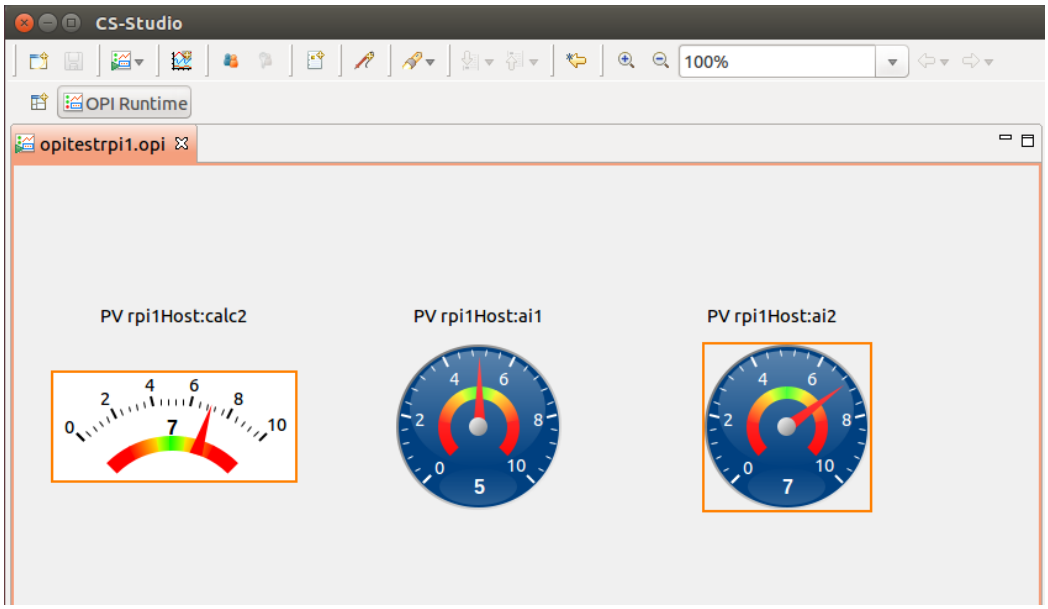


Fig. 229: Testing PVs indicators in CSS.

Remember to save all changes by accessing “File” / “Save” menu, or clicking on the disk icon in the top left menu.

In case of any error, or if the PV appears as “Disconnected”, go to “File” menu and click “Restart CS-Studio” option.

9 ANNEX: INSTALLING JAVA JRE 1.8 IN UBUNTU 14.04 32BITS

Process adapted from:
http://www.ubuntu-guia.com/2012/04/instalar-oracle-java-7-en-ubuntu-1204.html

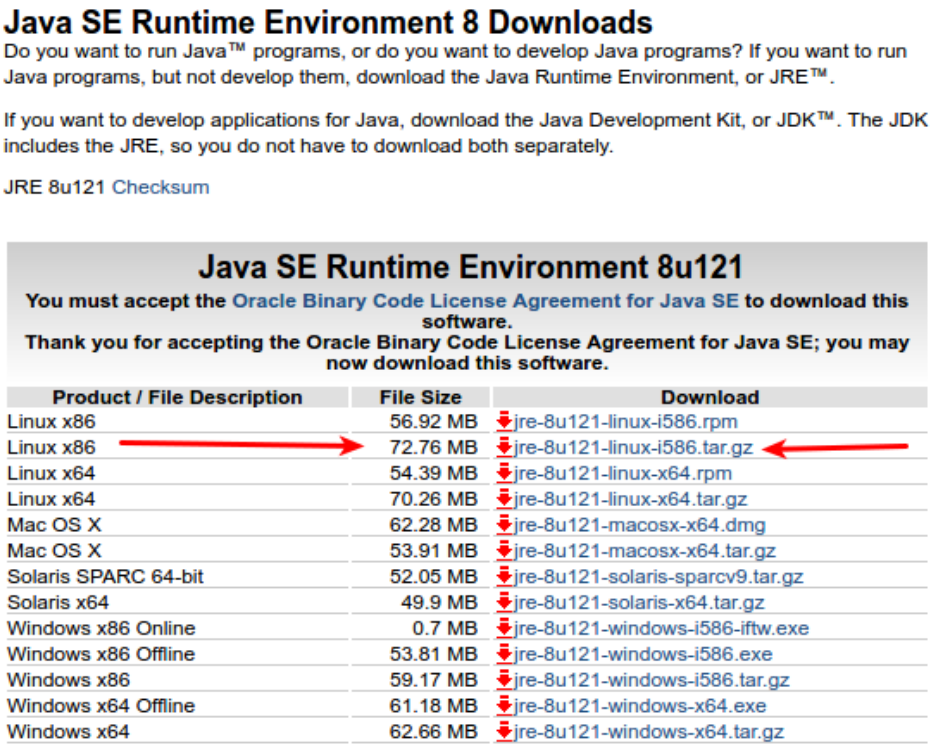
Typically, the process to install Java JRE in Linux is simple, only one command is needed:

```
$ sudo apt-get install default-jre
```

However, we are running Ubuntu 14.04 in our virtual machine, which installs by default Java JRE 1.7. Unlike Ubuntu 14.10 or Ubuntu 16.04, in which Java JRE 1.8 is installed.
We need to perform an additional process if we want to install 1.8 version of Java JRE, because Control System Studio (version 4) needs that version.

First, load VMware virtual machine running Ubuntu 14.04.

Download Linux x86 file **"jre-8u121-linux-i586.tar.gz"** from:
http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html
The web page is shown in Fig. 230, remember we are using a Ubuntu 32 bits virtual machine.



Java SE Runtime Environment 8 Downloads

Do you want to run Java™ programs, or do you want to develop Java programs? If you want to run Java programs, but not develop them, download the Java Runtime Environment, or JRE™.

If you want to develop applications for Java, download the Java Development Kit, or JDK™. The JDK includes the JRE, so you do not have to download both separately.

JRE 8u121 Checksum

Java SE Runtime Environment 8u121

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	56.92 MB	jre-8u121-linux-i586.rpm
Linux x86	72.76 MB	jre-8u121-linux-i586.tar.gz
Linux x64	54.39 MB	jre-8u121-linux-x64.rpm
Linux x64	70.26 MB	jre-8u121-linux-x64.tar.gz
Mac OS X	62.28 MB	jre-8u121-macosx-x64.dmg
Mac OS X	53.91 MB	jre-8u121-macosx-x64.tar.gz
Solaris SPARC 64-bit	52.05 MB	jre-8u121-solaris-sparcv9.tar.gz
Solaris x64	49.9 MB	jre-8u121-solaris-x64.tar.gz
Windows x86 Online	0.7 MB	jre-8u121-windows-i586-iftw.exe
Windows x86 Offline	53.81 MB	jre-8u121-windows-i586.exe
Windows x86	59.17 MB	jre-8u121-windows-i586.tar.gz
Windows x64 Offline	61.18 MB	jre-8u121-windows-x64.exe
Windows x64	62.66 MB	jre-8u121-windows-x64.tar.gz

Fig. 230: Downloading JavaSE Runtime Environment (JRE).

Move downloaded file to *"Documents"* folder

Open a terminal windows in Documents, and unzip the file ("jre1.8.0_121" folder is created):

```
tar -xvf jre-8u121-linux-i586.tar.gz
```

Create a "/jvm/jre1.8.0_121" folder into "/usr/lib" using this command:

```
sudo mkdir -p /usr/lib/jvm/jre1.8.0_121
```

Move unzipped folder from Documents to the new directory:

```
sudo mv jre1.8.0_121/* /usr/lib/jvm/jre1.8.0_121/
```

Execute the following command to install Java:

```
sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jre1.8.0_121/bin/java 0
```

Select your desired Java version if there is more than one available:

```
sudo update-alternatives --config java
```

Press (Fig. 231) the number corresponding to your desired Java version (number two in our example, Java 1.8.0_121)

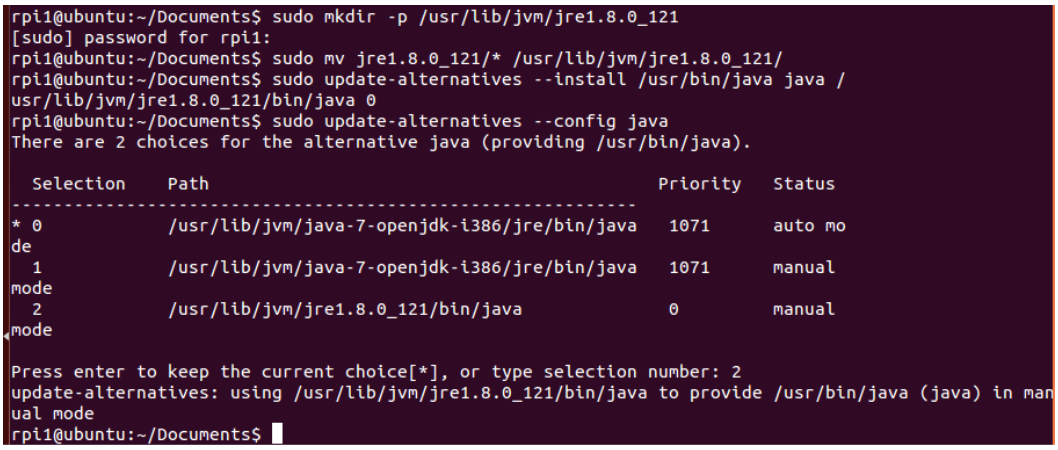


Fig. 231: Picking Java version to use.

Check Java version in use with this command:

```
java -version
```

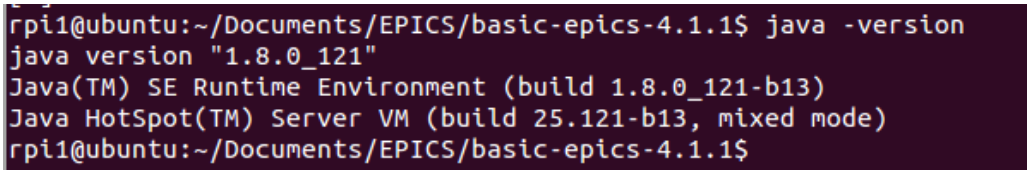


Fig. 232: Checking Java version in use.

It shows 1.8, so we can already use Control System Studio 4.1.1 version.

10 ANNEX: CONFIGURING SSH

To be able to remotely control the rpi, the most common option is SSH protocol. Remote control allow us to avoid connecting a keyboard and monitor to the rpi ("headless" mode). To do this, it is necessary to install an SSH server on the rpi. The most common server is "openSSH", but there are others like "dropbear".

NOTE: We had several issues using "*openSSH*", so we have used "*dropbear*" SSH server instead after removing "openSSH" from Buildroot packages.

These are the error messages using "openSSH" (Fig. 233), a log is available at */var/log* folder, in "*messages"* file.

```
# less /var/log/messages
```

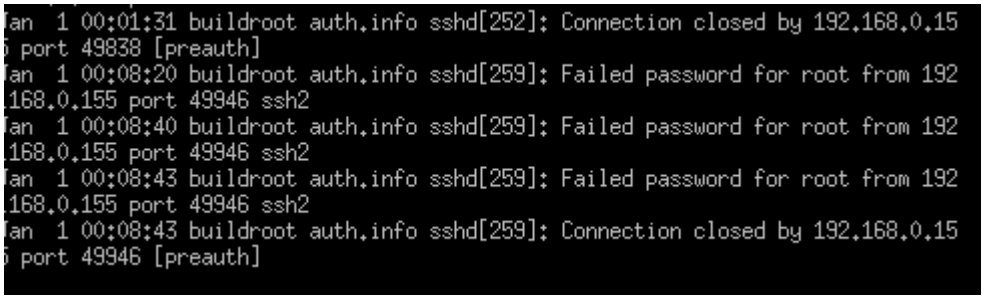


Fig. 233: Error messages using OpenSSH to connect to Raspberry Pi.

A password error is shown. This error appeared even after we defined a password for "root" user (mandatory before an SSH is established)
Let's start from scratch before making changes in Buildroot:

```
$ make clean
$ make xconfig
```

Remove "*openssh"* package and select "*dropbear"* package (select "NTP" package too, for date and time purposes).

Execute "*make"* in Buildroot to generate our Linux image.

Copy the image to microsd card, run Putty, and check using "ps -A" command (Fig. 234) that SSH server "dropbear" is running:

```
# ps -A
```

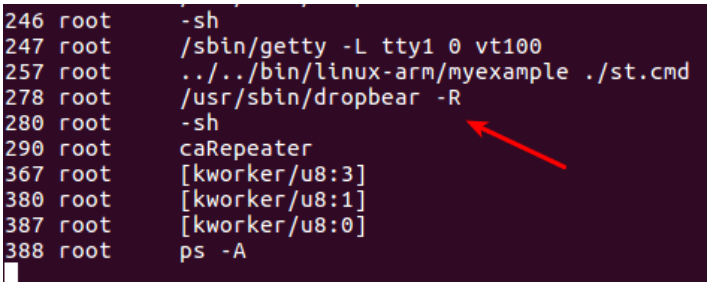


Fig. 234: Checking SSH server Dropbear is up and running.

First we need to define a password for "root" user, because by default there is none. If we want to use a SSH connection, this step must be done. Let's use "passwd" command:

```
# passwd
```

You can use "raspberry" as a password:

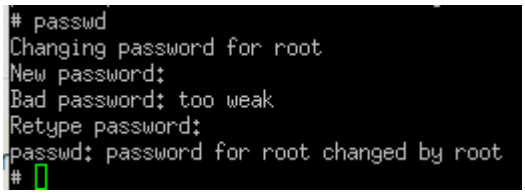


Fig. 235: Defining root user password.

You need to know also the IP address of the rpi. It is found in kernel messages shown at boot time (Fig. 236) and also using "ifconfig" command:

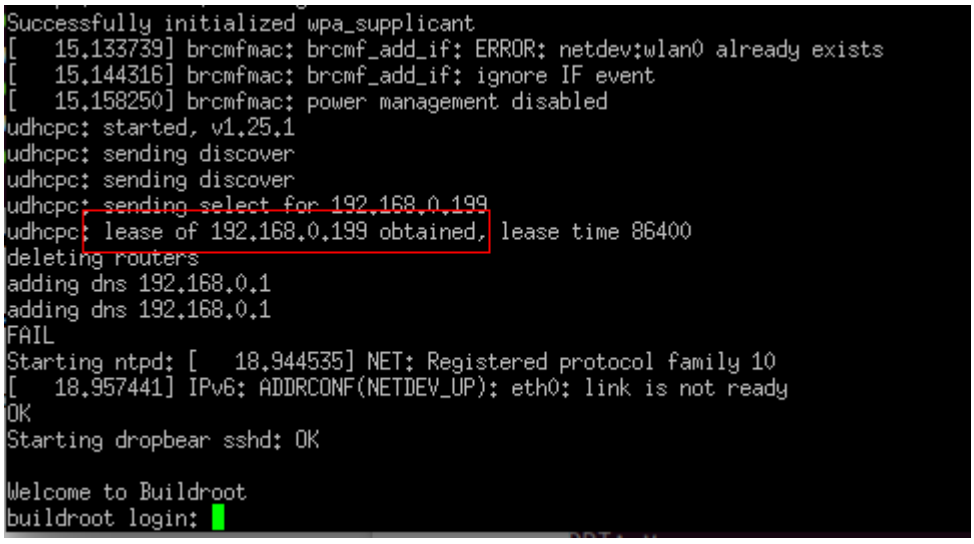


Fig. 236: Finding out Raspberry Pi IP address.

To establish a SSH connection, you have to execute the following command from a computer connected to same network as the rpi:

```
$ ssh root@192.168.0.199
```

Simply type “yes” after a warning message about authenticity appears, type the password defined in the previous step, and if everything goes according to plan, a prompt “#” will be shown.

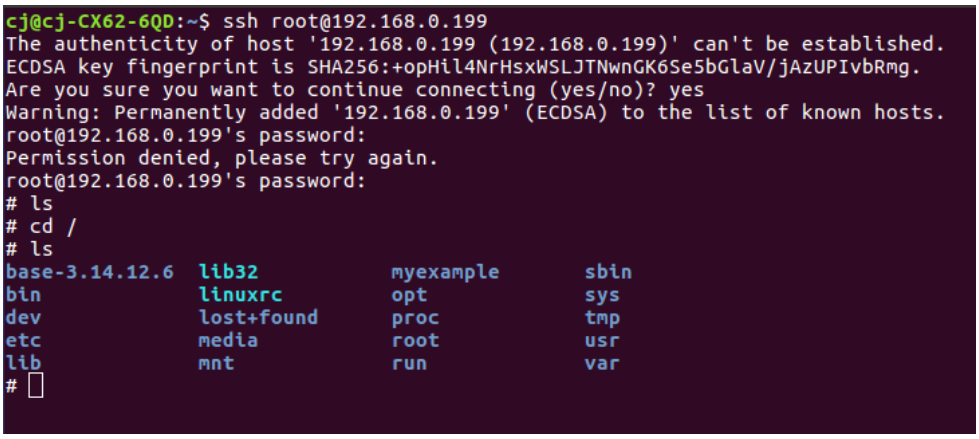
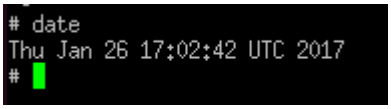


Fig. 237: Connecting a pc to Raspberry Pi using SSH.

If NTP was installed, you can check using “date” command if date and time are correct.



11 ANNEX: SOLVING PATH ISSUE

A problem arose during the tests. It was a PATH related issue, because system could not find EPICS commands *"caget"* and *"camonitor"*. The only way to make it work, was to put the symbols "./" at the beginning of every command, and that should not be necessary (Fig. 238).

```
# ./$EPICS_BASE/bin/linux-arm/caget rpi2Host:ai1
```

Fig. 238: Error executing caget due to undefined PATH in EPICS.

If we execute the commands from *"base-3.14.12.6/bin/linux-arm"* folder, we need again to put symbols "./" in front of *"caget"* (error will appear if we do not put them).

```
# cd base-3.14.12.6/bin/linux-arm
# ./caget rpi2Host:ai1
```

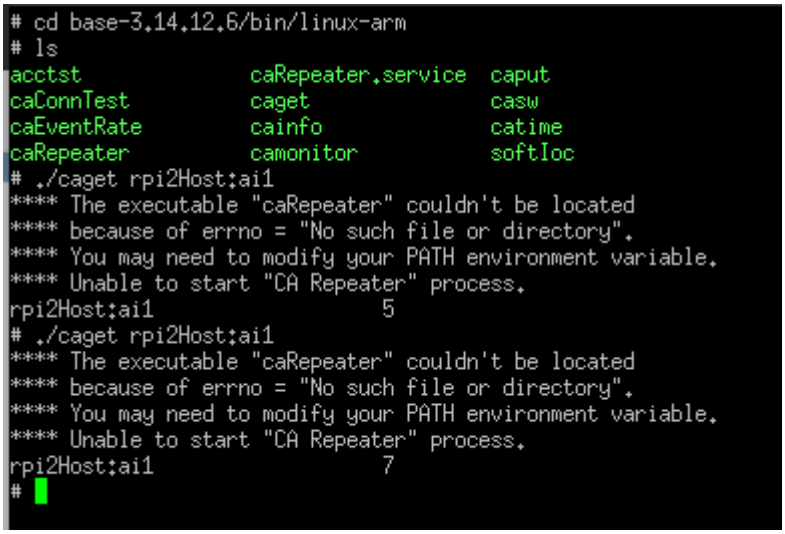


Fig. 239: Executing caget command from bin folder.

Here is a tip for returning to the original PATH of the system (maybe necessary after a lot of tests):

```
# export PATH=/bin:/sbin:/usr/bin:/usr/sbin
```

Executing “caget” command with that PATH, results in “not found” error:

```
# caget
```

```
-sh: caget: not found
```

But, if we define PATH this way, including “linux-arm” base folder:

```
# export PATH=$PATH:/base-3.14.12.6/bin/linux-arm
```

We see now (Fig. 240) how executing “caget” command does not return a “not found” error, because the system has already found the command.

Fig. 240: Correct execution of caget after defining EPICS PATH.

Now we are going to use environment variable EPICS_BASE to add that path.

Start from scratch defining default PATH:

```
# export PATH=/bin:/sbin:/usr/bin:/usr/sbin
```

And then:

```
# export EPICS_BASE=/base-3.14.12.6
# export PATH=$PATH:/$EPICS_BASE/bin/linux-arm
```

Using "env" command we can check that the environment variables are correct, and running "*caget*" or any other EPICS command is possible (Fig. 241), no need to use "./".

Fig. 241: Another way to define EPICS PATH.

12 ANNEX: LIST OF PACKAGES TO INSTALL INTO UBUNTU 14.04 VIRTUAL MACHINE

1. g++
2. dselect
3. git
4. gdbserver
5. u-boot-tools
6. qt4-dev-tools
7. qt4-qmake
8. eclipse
9. eclipse-cdt
10. gparted
11. putty
12. nautilus-open-terminal
13. synaptic
14. ncurses-dev
15. graphviz
16. python-matplotlib
17. Gnome-session-fallback
18. libreadline6-dev

Remember to update Ubuntu first:

```
sudo apt-get update
sudo apt-get upgrade
```

13 ANNEX: BUILDROOT SETTINGS

- “Filesystem images / exact size in blocks” -> Set value to 412000
- “Toolchain / C library” -> glibc
- “Toolchain” -> “Build cross gdb for the host” -> gdb 7.10.x
- "System Configurations / dev management" -> Dynamic using Devtmpfs + mdev
- “Target Packages / Interpreter languages and scripting” -> perl
- “Target packages / Networking applications” -> dropbear
- “Target Packages / Hardware Handling / Firmware” -> rpi-wifi-firmware
- "Target Packages / Networking Applications" -> wpa_supplicant
- "Target Packages / Networking Applications" -> wpa_supplicant ->Enable EAP
- "Target Packages / Networking Applications" -> crda
- "Target Packages / Networking Applications" -> iw
- "Target Packages / Networking Applications" -> wireless-regdb
- “Target packages / Networking applications” -> ntp
- “System Configuration / Root filesystem overlay directories” -> “board/raspberrypi3/rpi3ov2017”

This is the final content of defconfig file “raspberrypi3_defconfig”

```
BR2_arm=y
BR2_cortex_a7=y
BR2_ARM_FPU_NEON_VFPV4=y
BR2_TOOLCHAIN_BUILDROOT_GLIBC=y
BR2_PACKAGE_HOST_LINUX_HEADERS_CUSTOM_4_4=y
BR2_TOOLCHAIN_BUILDROOT_CXX=y
BR2_PACKAGE_HOST_GDB=y
BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_MDEV=y
BR2_SYSTEM_DHCP="eth0"
BR2_ROOTFS_OVERLAY="board/raspberrypi3/rpi3ov2017"
BR2_ROOTFS_POST_BUILD_SCRIPT="board/raspberrypi3/post-build.sh"
BR2_ROOTFS_POST_IMAGE_SCRIPT="board/raspberrypi3/post-image.sh"
BR2_ROOTFS_POST_SCRIPT_ARGS="--add-pi3-miniuart-bt-overlay"
BR2_LINUX_KERNEL=y
BR2_LINUX_KERNEL_CUSTOM_GIT=y
BR2_LINUX_KERNEL_CUSTOM_REPO_URL="https://github.com/raspberrypi/linux.git"
BR2_LINUX_KERNEL_CUSTOM_REPO_VERSION="9669a50a3a8e4f33b4fe138277bc4407e1eab9b2"
BR2_LINUX_KERNEL_DEFCONFIG="bcm2709"
BR2_LINUX_KERNEL_DTS_SUPPORT=y
BR2_LINUX_KERNEL_INTREE_DTS_NAME="bcm2710-rpi-3-b"
BR2_PACKAGE_RPI_FIRMWARE=y
BR2_PACKAGE_RPI_WIFI_FIRMWARE=y
BR2_PACKAGE_PERL=y
BR2_PACKAGE_CRDA=y
BR2_PACKAGE_DROPBEAR=y
BR2_PACKAGE_IW=y
BR2_PACKAGE_NTP=y
BR2_PACKAGE_WPA_SUPPLICANT=y
BR2_PACKAGE_WPA_SUPPLICANT_EAP=y
BR2_TARGET_ROOTFS_EXT2=y
BR2_TARGET_ROOTFS_EXT2_4=y
BR2_TARGET_ROOTFS_EXT2_BLOCKS=412000
# BR2_TARGET_ROOTFS_TAR is not set
BR2_PACKAGE_HOST_DOSFSTOOLS=y
BR2_PACKAGE_HOST_GENIMAGE=y
BR2_PACKAGE_HOST_MTOOLS=y
```

14 ANNEX: SUMMARY OF VIRTUAL MACHINE, LINUX, BUILDROOT AND EPICS SET-UP PROCESS

14.1 Installing and configuring Ubuntu 14.04 virtual machine

Download **VMware Workstation Player 12.5.1-4542065**

- File: VMware-Player-12.5.1-4542065.x86_64.bundle
- Link: http://www.vmware.com/products/workstation-for-linux.html

Download **Ubuntu 14.04.5 LTS (Trusty Tahr) 32 bits**

- File: ubuntu-14.04.5-desktop-i386.iso
- Link: https://www.ubuntu.com/download/alternative-downloads

Install VMware in Linux computer to host our virtual machine:

```
sudo sh VMware-Player-12.5.1-4542065.x86_64.bundle
```

Create Ubuntu 14 32 bits virtual machine

- Size: 80GB
- “split virtual into multiple files”
- RAM and number of processors in use according to your pc capabilities

VMware virtual machine settings:

- “Edit virtual machine settings->Hardware->Processors->Activate Virtualize Intel VT-x”
- Activate “Intel VT” in computer’s BIOS

Run Ubuntu 14 virtual machine and update it

```
sudo apt-get update
sudo apt-get upgrade
```

Ubuntu 14 settings

- Remove password after idle time
- Set time zone
- No login access
- Set keyboard to Spanish

14.2 Installing Buildroot

Install into Ubuntu virtual machine all the packages needed for Buildroot and other applications:

```
sudo apt-get install g++ dselect git gdbserver u-boot-tools qt4-dev-tools qt4-qmake eclipse
sudo apt-get install eclipse-cdt gparted putty nautilus-open-terminal synaptic ncurses-dev
sudo apt-get install graphviz python-matplotlib Gnome-session-fallback libreadline6-dev
```

Install and execute Buildroot 2016.11 for the first time in Ubuntu 14 virtual machine

```
cd ~/Documents
wget https://buildroot.org/downloads/buildroot-2016.11.tar.gz
tar xzf buildroot-2016.11.tar.gz
cd buildroot-2016.11
make raspberrypi3_defconfig
make
```

Copy resulting image to microsd card (mount point could be sdb, mmcblk0, etc.)

```
sudo umount /dev/mmcblk0p*
sudo dd if=output/images/sdcard.img of=/dev/mmcblk0
```

Or, right click “sdcard.img” file, choose “Open with Disk Image Writer”

FTDI Cable

- Connect it and find out identifier (ttyUSB0 in this case)

```
dmesg
sudo putty
```

- Serial settings: **/dev/ttyUSB0**, Speed: **115200**, Data bits: **8**, Stop bits: **1**, Parity: **none**, Flow control: **XON/XOFF**

Boot rpi and check (run commands “ps –A”,” top”, or “poweroff”)

14.3 Previous Buildroot settings for EPICS (glibc, wifi, perl, image size)

Configuring Buildroot:

```
cd ~/Documents/buildroot-2016.11
make xconfig
```

Buildroot options, added to “raspberrypi3_defconfig” default settings:

- “Filesystem images / exact size in blocks” -> set value 412000
- “Toolchain / C library” -> glibc
- “Target Packages / Interpreter languages and scripting” -> perl
- “Target packages / Networking applications” -> dropbear

- “Toolchain” -> “Build cross gdb for the host” -> gdb 7.10.x
- “System Configurations / dev management” -> Dynamic using Devtmpfs + mdev

Save Buildroot setup, click disk icon up left

Create “package/rpi-wifi-firmware“ folder

```
mkdir ~/Documents/buildroot-2016.11/package/rpi-wifi-firmware
```

Create “package/rpi-wifi-firmware/**Config.in**” file

```
sudo nano ~/Documents/buildroot-2016.11/package/rpi-wifi-firmware/Config.in
```

Content:

```
config BR2_PACKAGE_RPI_WIFI_FIRMWARE
       bool "rpi-wifi-firmware"
       help
         This package provides the wifi firmware for the Raspberry Pi
```

Create “package/rpi-wifi-firmware/**rpi-wifi-firmware.mk**” file

```
sudo nano ~/Documents/buildroot-2016.11/package/rpi-wifi-firmware/rpi-wifi-firmware.mk
```

Content (be careful about long lines):

```
RPI_WIFI_FIRMWARE_VERSION = master
RPI_WIFI_FIRMWARE_SITE = $(call github,RPi-Distro,firmware-nonfree,$(RPI_WIFI_FIRMWARE_VERSION))
RPI_WIFI_FIRMWARE_LICENSE = Proprietary
RPI_WIFI_FIRMWARE_LICENSE_FILES = brcm80211/LICENSE

define RPI_WIFI_FIRMWARE_INSTALL_TARGET_CMDS
       $(INSTALL) -D -m 0644 $(@D)/brcm80211/brcm/brcmfmac43143.bin $(TARGET_DIR)/lib/firmware/brcm/brcmfmac43143.bin
       $(INSTALL) -D -m 0644 $(@D)/brcm80211/brcm/brcmfmac43430-sdio.bin $(TARGET_DIR)/lib/firmware/brcm/brcmfmac43430-sdio.bin
       $(INSTALL) -D -m 0644 $(@D)/brcm80211/brcm/brcmfmac43430-sdio.txt $(TARGET_DIR)/lib/firmware/brcm/brcmfmac43430-sdio.txt
endef

$(eval $(generic-package))
```

Edit “package/**Config.in**” file

```
sudo nano ~/Documents/buildroot-2016.11/package/Config.in
```

Add this line to ”Hardware handling” option, “Firmware” section

```
source "package/rpi-wifi-firmware/Config.in"
```

Create “board/raspberrypi3/**interfaces**” file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/interfaces
```

Content:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
        pre-up /etc/network/nfs_check
        wait-delay 15

auto wlan0
iface wlan0 inet dhcp
        pre-up wpa_supplicant -B -Dwext -iwlan0 -c/etc/wpa_supplicant.conf
        post-down killall -q wpa_supplicant
        wait-delay 15

iface default inet dhcp
```

Create “board/raspberrypi3/ **wpa_supplicant.conf**” file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/wpa_supplicant.conf
```

Add wifi network data, type name into “ssid” and password into “psk”

```
network={
ssid="CubotX17CJ"
psk="xxxxxx"
}
network={
ssid="vodafone46A1"
psk="xxxxx"
}
network={
ssid="WIFIUPM"
key_mgmt=WPA-EAP
eap=PEAP
identity="xxxxx@alumnos.upm.es"
password="xxxxxxx"
}
```

Create “/board/raspberrypi3/**profile**” file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/profile
```

Add this content:

```
# Definimos la zona horaria Madrid
export TZ=CET-1CEST,M3.5.0,M10.5.0/3

if [ "$PS1" ]; then
    if [ "`id -u`" -eq 0 ]; then
          export PS1='# '
    else
          export PS1='$ '
    fi
fi

export PAGER='/bin/more '
export EDITOR='/bin/vi'

# Source configuration files from /etc/profile.d
for i in /etc/profile.d/*.sh ; do
    if [ -r "$i" ]; then
          . $i
    fi
    unset i
done
```

Create "/board/raspberrypi3/**ntp.conf**" file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/ntp.conf
```

Content:

```
server 3.es.pool.ntp.org iburst
server 0.europe.pool.ntp.org iburst
server 2.europe.pool.ntp.org iburst

# Allow only time queries, at a limited rate, sending KoD when in excess.
# Allow all local queries (IPv4, IPv6)
restrict default nomodify nopeer noquery limited kod
restrict 127.0.0.1
restrict [::1]
```

Edit "~/Documents/buildroot-2016.11/board/raspberrypi3/**post-build.sh**" file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/post-build.sh
```

Content to add at the end of the file:

```
#puesto por mi para configurar wifi en rpi3
cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
```

```
chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf
cp board/raspberrypi3/interfaces ${TARGET_DIR}/etc/network/interfaces
cp board/raspberrypi3/wpa_supplicant.conf
${TARGET_DIR}/etc/wpa_supplicant.conf
#en el archivo profile definimos variables de entorno para EPICS y NTP
cp board/raspberrypi3/profile ${TARGET_DIR}/etc/profile
#el archivo ntp.conf configura el NTP de fecha y hora
cp board/raspberrypi3/ntp.conf ${TARGET_DIR}/etc/ntp.conf
```

Activate required packages in Buildroot:

```
make xconfig
```

Select packages in **Buildroot:**

- "Target Packages / Hardware Handling / Firmware" -> rpi-wifi-firmware
- "Target Packages / Networking Applications" -> wpa_supplicant
- "Target Packages / Networking Applications" -> wpa_supplicant ->Enable EAP
- "Target Packages / Networking Applications" -> crda
- "Target Packages / Networking Applications" -> iw
- "Target Packages / Networking Applications" -> wireless-regdb
- "Target packages / Networking applications" -> ntp

Save Buildroot settings

Build:

```
cd ~/Documents/buildroot-2016.11
make savedefconfig
make clean
make
```

Copy image to microsd card
Run Putty and boot rpi, check wifi and date using "ifconfig" and "date" commands

14.4 EPICS download and compilation

Create EPICS folder in "Documents"

```
mkdir ~/Documents/EPICS
cd ~/Documents/EPICS
```

Download and unzip EPICS base in that folder

```
wget https://www.aps.anl.gov/epics/download/base/baseR3.14.12.6.tar.gz
tar xzf baseR3.14.12.6.tar.gz
```

Edit “EPICS/ base-3.14.12.6/configure/**CONFIG_SITE**” file

```
sudo nano ~/Documents/EPICS/base-3.14.12.6/configure/CONFIG_SITE
```

Edit the following lines:

```
CROSS_COMPILER_TARGET_ARCHS=linux-arm
CROSS_COMPILER_HOST_ARCHS=linux-x86
SHARED_LIBRARIES=NO
STATIC_BUILD=YES
```

Edit “EPICS/ base-3.14.12.6/configure/os/**CONFIG_SITE.linux-x86.linux-arm**” file

```
sudo nano ~/Documents/EPICS/base-3.14.12.6/configure/os/CONFIG_SITE.linux-x86.linux-arm
```

Edit the following lines:

```
GNU_DIR = ~/Documents/buildroot-2016.11/output/host/usr
GNU_TARGET = arm-buildroot-linux-gnueabihf
```

Build EPICS

```
cd ~/Documents/EPICS/base-3.14.12.6
make
```

14.5 Adding EPICS and softIOC example into image generated by Buildroot

Create “~/Documents/buildroot-2016.11/board/raspberrypi3/**rpi3ov2017**” folder

```
mkdir ~/Documents/buildroot-2016.11/board/raspberrypi3/rpi3ov2017
```

Copy folder containing EPICS built “~/Documents/EPICS/ base-3.14.12.6”, into “rpi3ov2017”

```
cp -r ~/Documents/EPICS/base-3.14.12.6 ~/Documents/buildroot-2016.11/board/raspberrypi3/rpi3ov2017
```

```
cd ~/Documents/buildroot-2016.11
make xconfig
```

Add EPICS built folder into **Buildroot**

- "System Configuration / Root filesystem overlay directories" -> "board/raspberrypi3/rpi3ov2017"

Save Buildroot settings

14.5.1 Creating example SoftIOC

Create "~/Documents/EPICS/**myexample**" folder

```
mkdir ~/Documents/EPICS/myexample
```

Create EPICS IOC (type linux-arm when asked for a name)

```
cd ~/Documents/EPICS/myexample
~/Documents/EPICS/base-3.14.12.6/bin/linux-x86/makeBaseApp.pl -t example myexample
~/Documents/EPICS/base-3.14.12.6/bin/linux-x86/makeBaseApp.pl -i -t example myexample
make
```

Copy "myexample" folder to "board/raspberrypi3/rpi3ov2017"

```
cp -r ~/Documents/EPICS/myexample ~/Documents/buildroot-2016.11/board/raspberrypi3/rpi3ov2017
```

Edit "/rpi3ov2017/myexample/iocBoot/iocmyexample/**envPaths**" file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/rpi3ov2017/myexample/iocBoot/iocmyexample/envPaths
```

Content:

```
epicsEnvSet("ARCH","linux-arm")
epicsEnvSet("IOC","iocmyexample")
epicsEnvSet("TOP","/myexample")
epicsEnvSet("EPICS_BASE","/base-3.14.12.6")
```

14.5.2 Including environment variables and PATH into Buildroot

Edit "~/Documents/buildroot-2016.11/board/raspberrypi3/**profile**" file

```
sudo nano ~/Documents/buildroot-2016.11/board/raspberrypi3/profile
```

Content to include at the beginning of the file:

```
#Defining EPICS_BASE variable and EPICS executable in PATH
export EPICS_BASE=/base-3.14.12.6
export PATH=/bin:/sbin:/usr/bin:/usr/sbin:/$EPICS_BASE/bin/linux-arm
```

Make **Buildroot**

```
cd ~/Documents/buildroot-2016.11
make
```

Copy image to microsd card

Run Putty and boot rpi

Execute EPICS softIOC from rpi:

```
cd /myexample/iocBoot/iocmyexample
../../bin/linux-arm/myexample ./st.cmd
```