Dpto de Telemática y Electrónica
Universidad Politécnica de Madrid

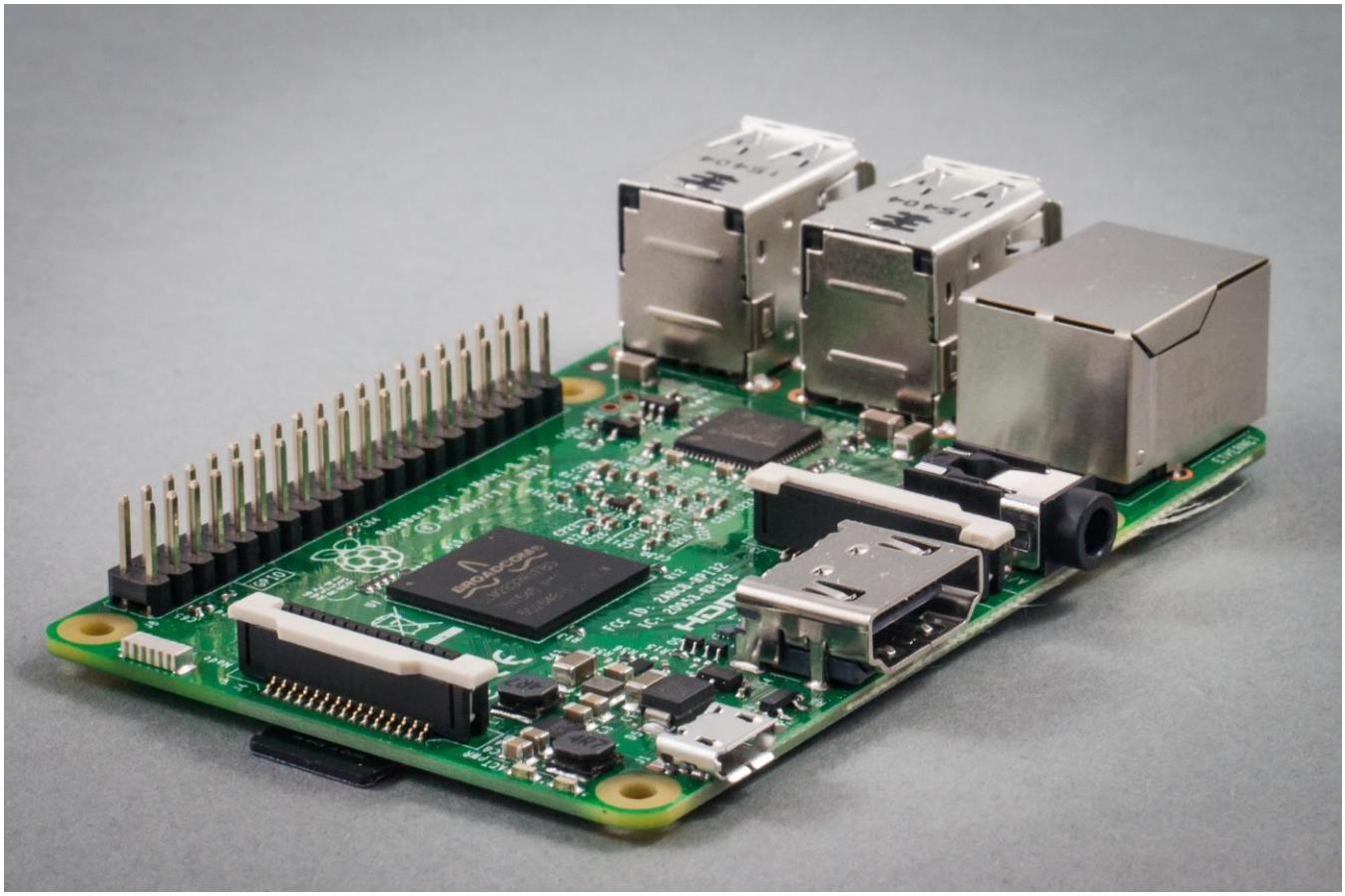
Embedded Linux Systems

Using Buildroot for building Embedded Linux Systems on Raspberry Pi 3 Model B

Sergio Esquembri

2018

ETSIS. Telecomunicación –Universidad Politécnica de Madrid



Acknoledgements

This document is based on a previous work using RPI Model B of my colleagues Dr. Mariano Ruiz and Dr. Francisco Javier Jímenez from the Department of Telematics and Electronics Engineering of the Technical University of Madrid.

Table of contents

1 SCOPE 5

1.1 Document Overview 5

1.2 Acronyms 5

2 REFERENCED DOCUMENTS 6

2.1 References 6

3 BUILDING LINUX USING BUILDROOT 7

3.1 Elements needed for the execution of these LABS. 7

3.2 Starting the VMware 7

3.3 Configuring Buildroot for RPI3. 11

3.4 Compiling buildroot. 15

3.5 Buildroot Output 16

3.6 Booting the Raspberry Pi. 17

3.7 Connecting the RPI to the network 23

3.7.1 Inspecting the configuration of the network interface automatically generated 23

3.7.2 Adding support to WIFI 23

4 USING INTEGRATED DEVELOPMENT ENVIRONMENT: ECLIPSE/CDT 25

4.1 Adding cross-compiling tools to PATH variable. 25

4.2 Cross-Compiling applications using Eclipse. 25

4.3 Automatic debugging using gdb and gdbserver. 33

5 PREPARING THE LINUX VIRTUAL MACHINE. 35

5.1 Download VMware Workstation Player. 35

5.2 Installing Ubuntu 14.04 LTS as virtual machine 35

5.3 Installing synaptic 35

5.4 Installing putty 36

5.5 Installing packages for supporting Buildroot. 37

5.6 Installing packages supporting Eclipse 37

Table of figures

Fig. 1: Main screen of VMware player with some VM available to be executed. 7
Fig. 2: Ubuntu Virtual Machine login screen. 8
Fig. 3 Buildroot home page. 8
Fig. 4: Example of Downloading buildroot source code. 9
Fig. 5: Buildroot folder (the folder name depends on the version downloaded). 9
Fig. 6: Dash home, Terminal application 10
Fig. 7: Buildroot setup screen. 11
Fig. 8: Successful compilation and installation of Buildroot 16
Fig. 9: Schematic representation of the Buildroot tool. Buildroot generates the root file system, the kernel image, the booloader and the toolchain. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/) 16
Fig. 10: images folder contains the binary files for our embedded system. 17
Fig. 11: RaspBerry-Pi (Version B) hardware with main elements identified. 18
Fig. 12: Raspberry-PI header terminal identification. 19
Fig. 13: Identification of the terminals in the USB-RS232 adapter 19
Fig. 14: Booting process for BCM2837 processor in the raspberry-pi. 20
Fig. 15: Putty program main window. 22
Fig. 16: Linux Running 22
Fig. 18: Summary of the different configurations for developing applications for embedded systems. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/) 25
Fig. 19: Cross compiling tools installed in the host computer 26
Fig. 20: Selection of the workspace for Eclipse. Use a folder in your account. 26
Fig. 21: Eclipse welcome window. 27
Fig. 22: Eclipse main window. 27
Fig. 23: Basic C project creation in Eclipse 28
Fig. 24: Cross-compiler prefix and path window. 28
Fig. 25: Hello world example. 29
Fig. 26: Tool Chain Editor should be configured to use Cross GCC. 29
Fig. 27: Cross tools locate on (path).The path shown in this figure is an example.Use always the path of your toolchain. 30
Fig. 28: Include search path. 30
Fig. 29: Libraries search path. 31
Fig. 30: Pop up window when executing "Connect to Server" 32
Fig. 31: Eclipse project compiled (Binaries has been generated). 32
Fig. 32: Run test program in Raspberry PI 33
Fig. 33: Creating a Debug Configuration 33
Fig. 34: Debug configuration including the path to locate the cross gdb tool. 34
Fig. 35: Synaptic program from Dash 36
Fig. 36: Synaptic windows 36

1 SCOPE

1.1 Document Overview

- This document describes the basic steps to develop and embedded Linux-based system using the Raspberry PI board. The document has been specifically written to use a Raspberry-PI development system based on the BCM2837 processor. All the software elements used have a GPL license.

[Time to complete the tutorial]: The time necessary to complete all the steps in this tutorial is approximately 8 hours.

Read carefully all the instructions before executing the practical part otherwise you will find errors and probably unpredicted errors. In parallel you need to review the slides available at Moodle site or at [RD1]

1.2 Acronyms

CPU	Central Processing Unit
EABI	Extended Application Binary Interface
EHCI	Enhanced Host Controller Interface
I/O	Input and Output
MMC	Multimedia card
NAND	Flash memory type for fast sequential read and write
PCI	Peripheral Component Interconnect – computer bus standard
PCI Express	Peripheral Component Interconnect Express
OS	Operating system
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

2 REFERENCED DOCUMENTS

2.1 References

[RD1] Embedded Linux system development.

Slides at https://moodle.upm.es/titulaciones/oficiales/course/<your course>

[RD2] Hallinan, C. Embedded Linux Primer. Second Edition. Prentice Hall. 2011.

[RD3] getting-started-with-ubuntu

[RD4] http://free-electrons.com/training/embedded-linux/

[RD5] Raspberry-Pi User Guide. Reference Manual.

www.myraspberry-pi.org/wp-content/.../Raspberry.Pi_.User_.Guide_.pdf

[RD6] http://www.uclibc.org/ uclib web site.

[RD7] http://www.gnu.org/software/binutils/ Binutils web site.

3 BUILDING LINUX USING BUILDROOT

3.1 Elements needed for the execution of these LABS.

In order to execute properly this lab you need the following elements:

1. VMware player version 5.0 or above. Available at www.wmware.com (free download and use). This software is already installed in the laboratory desktop computer.
2. A VMWare virtual machine with Ubuntu 12.04 and all the software packages installed is already available in the Desktop. This virtual machine is available for your personal use at the Department assistance office (preferred method). If you want to setup your virtual machine by yourself follow the instructions provided in the Annex I.
3. A Raspberry-Pi, the accessories and USB cable are available at the laboratory.
4. Basic knowledge of Linux commands.
   Available at http://www.ee.surrey.ac.uk/Teaching/Unix/ UNIX tutorial for beginners.

3.2 Starting the VMware

Start VMware player and open the RPI Virtual Machine. Wait until the welcome screen is displayed (see Fig. 1 and Fig. 2). Login as "*Ubuntu*" user using the password "rpi.2018". An Ubuntu tutorial is available at Moodle site.

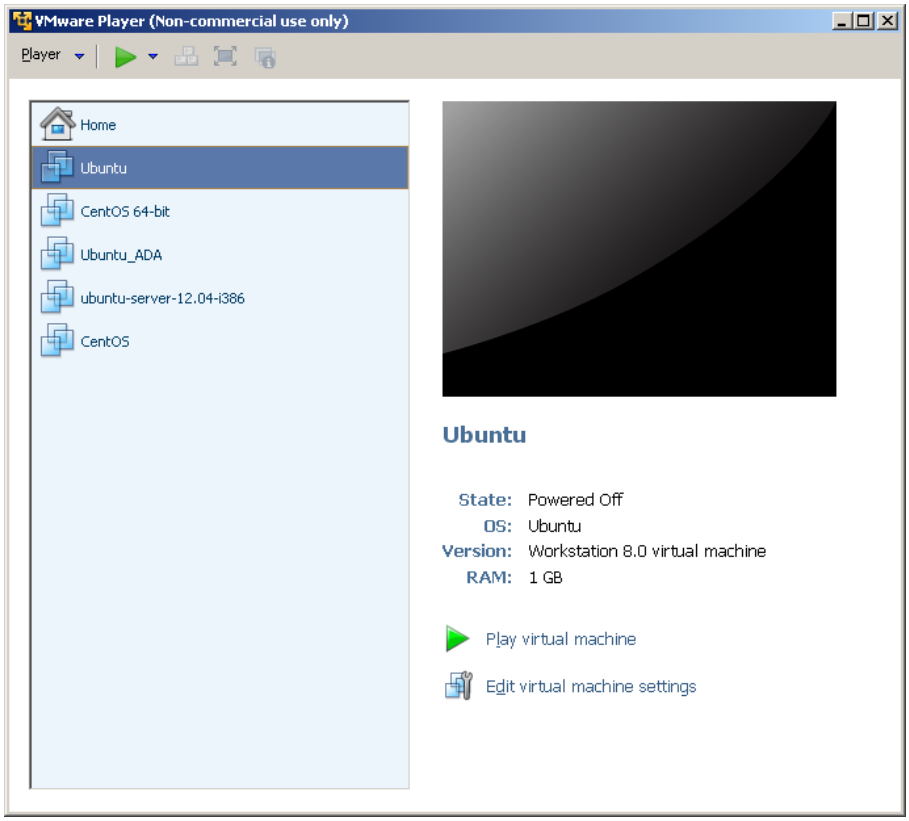


Fig. 1: Main screen of VMware player with some VM available to be executed.

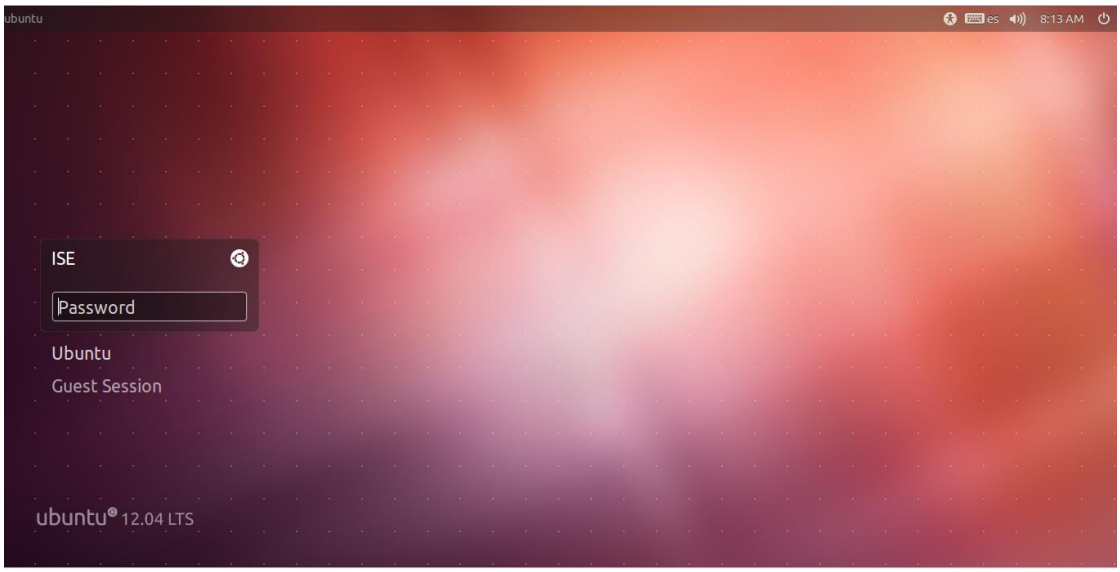


Fig. 2: Ubuntu Virtual Machine login screen.

Open the **Firefox** web browser and download from https://buildroot.org/ the version identified as buidlroot2018-02-1 (use the download link, see Fig. 3, and navigate searching for earlier releases if necessary, https://buildroot.org/downloads/). Save the file to the **Documents** folder in your account (Fig. 4).

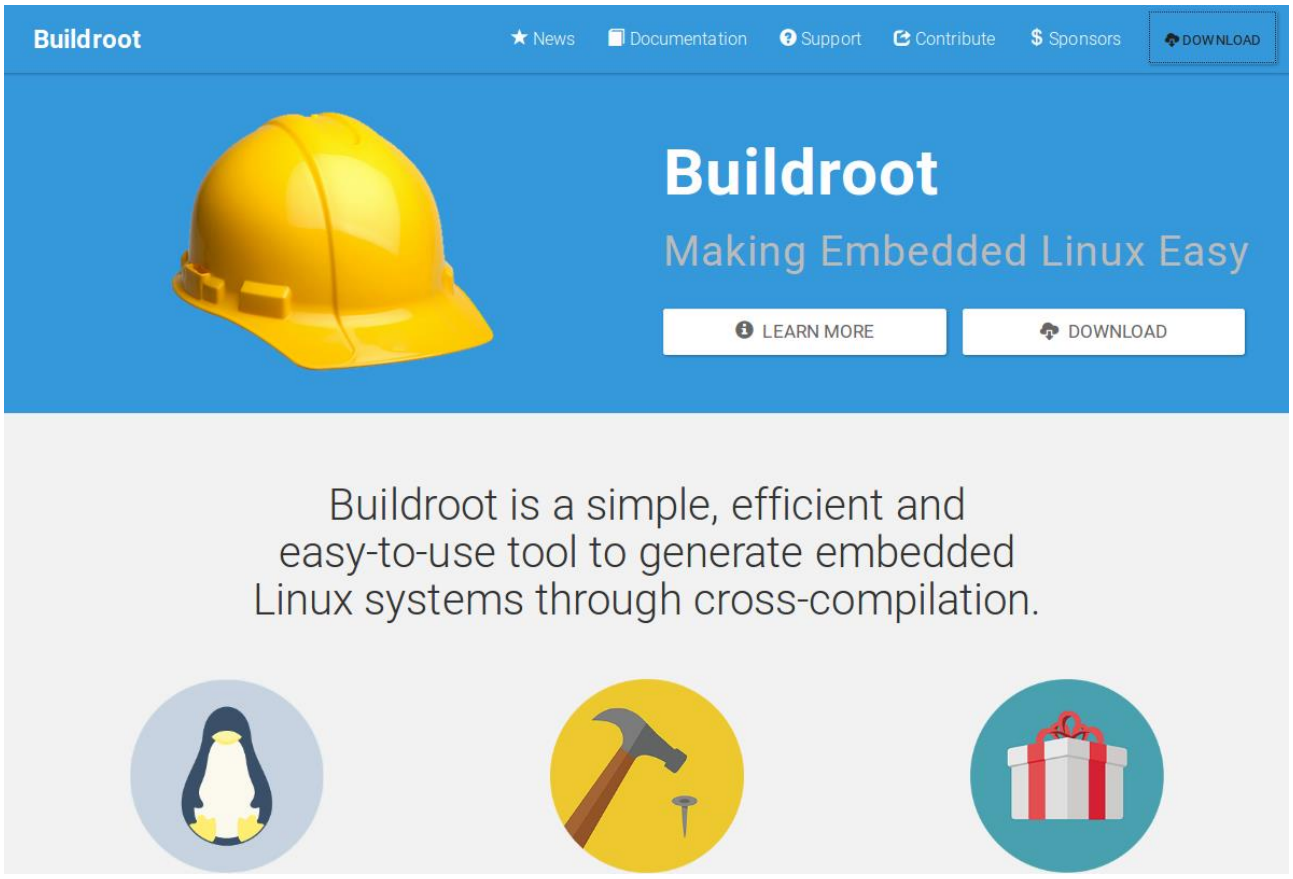


Fig. 3 Buildroot home page.

Buildroot is a tool to generate embedded Linux systems in our PC and then this Linux will be installed in the target.

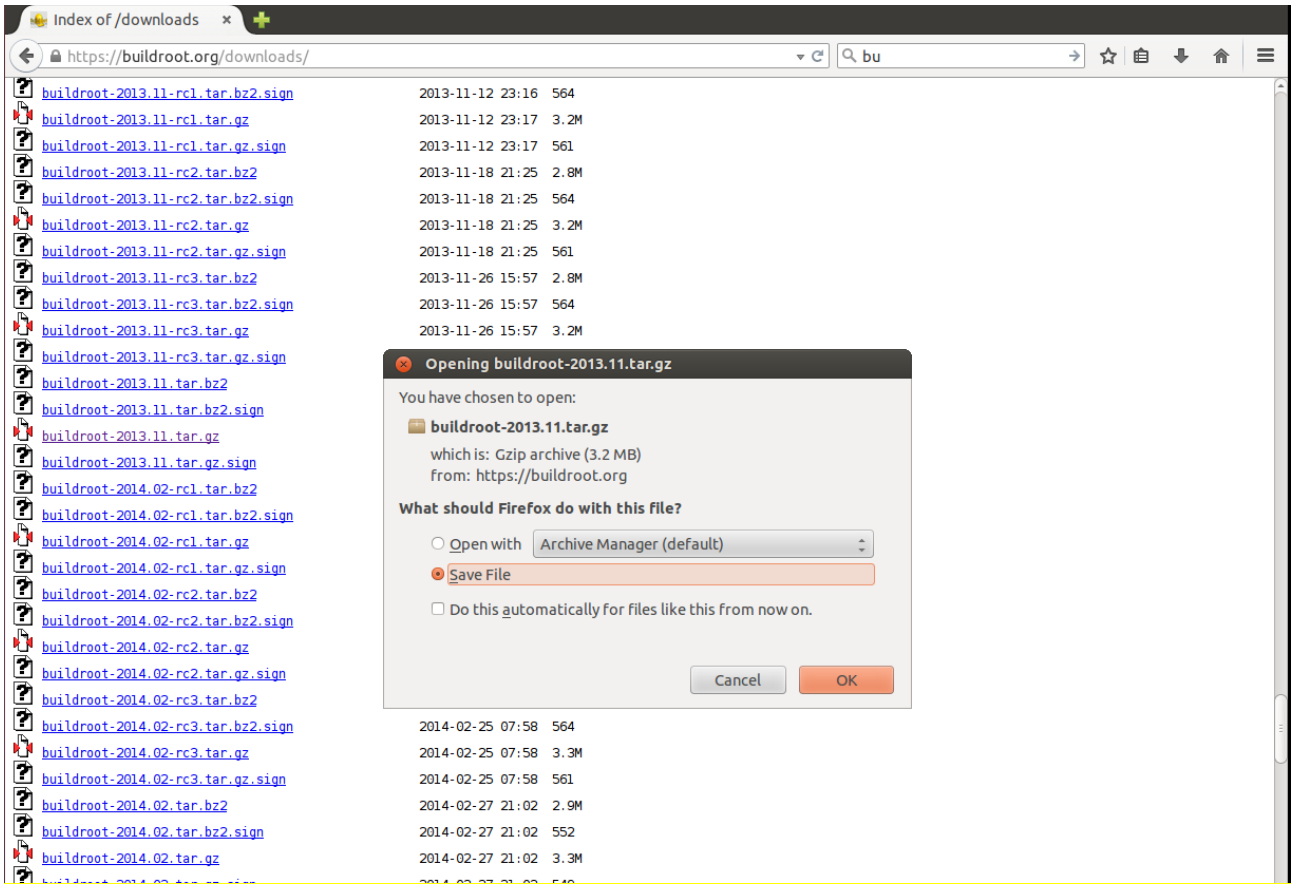


Fig. 4: Example of Downloading buildroot source code.

Copy the file to the “Documents” folder and decompress the file (Fig. 5).

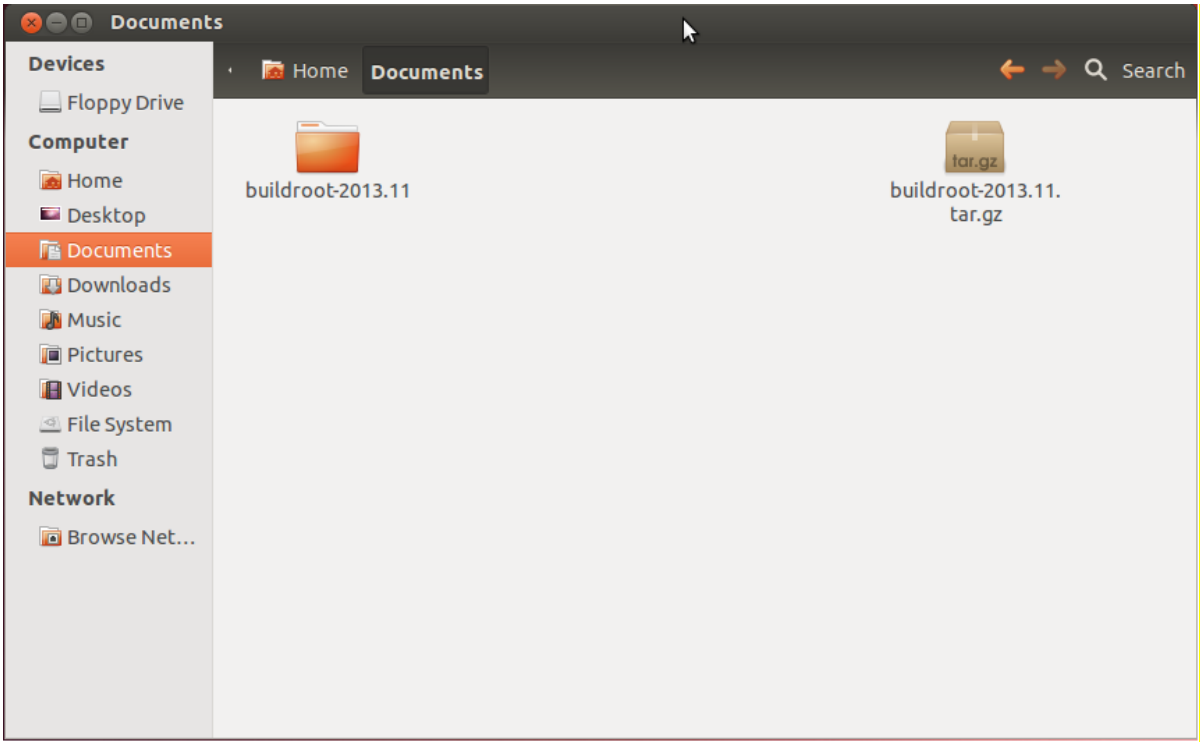


Fig. 5: Buildroot folder (the folder name depends on the version downloaded).

Right click in the window and execute “Open in Terminal” or execute from Dash home the Terminal application as is shown in the Fig. 6 (if “Open in Terminal” is not available, search how to install it in Ubuntu).

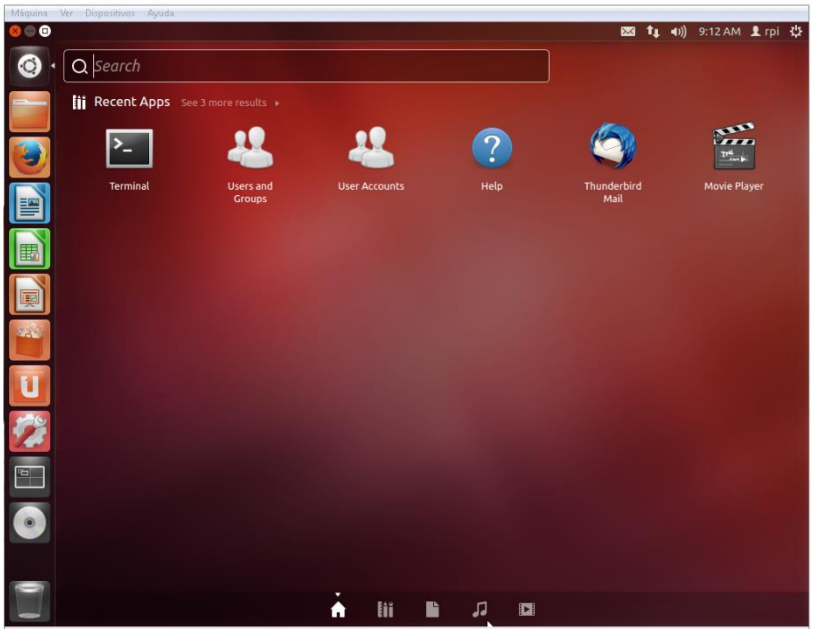


Fig. 6: Dash home, Terminal application

In some seconds a command window is displayed. Then, execute these commands:

```
rpi@ubuntu:~/Documents$ cd buildroot-xxxx.yy
rpi@ubuntu:~/Documents/buildroot-xxxx.yy$ make xconfig (or make menuconfig)
```

[Help]: In Linux "TAB" key helps you to autocomplete de commands, folders and files names. You can find a description of "make" application in this link https://www.gnu.org/software/make/manual/make.pdf

In some seconds you will see a new window similar to Fig. 7.

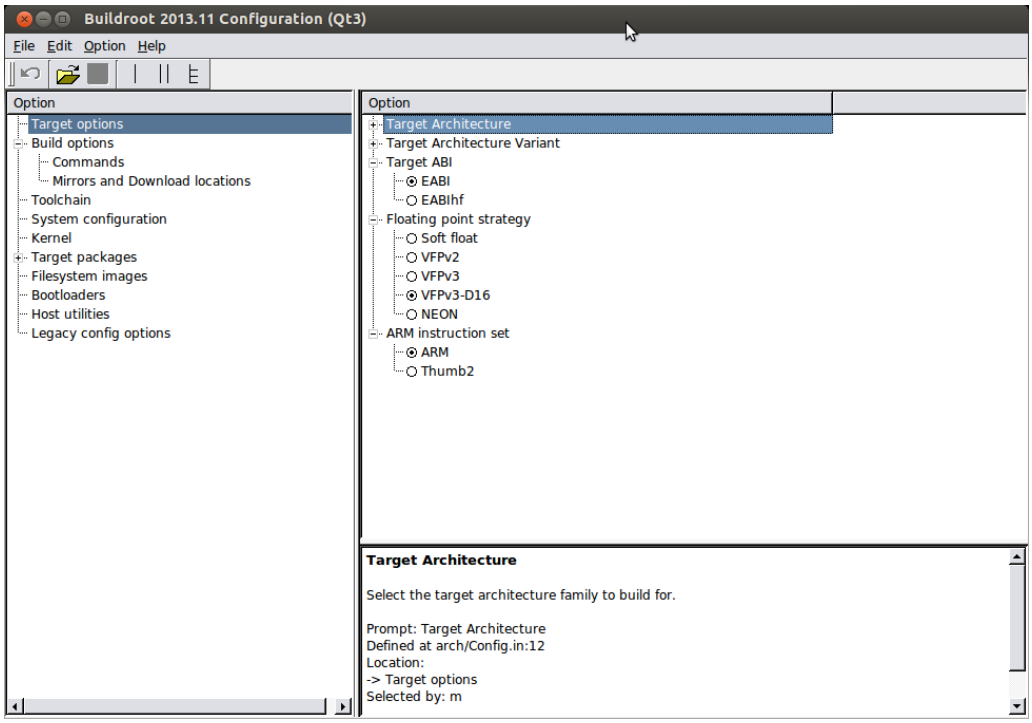


Fig. 7: Buildroot setup screen.

3.3 Configuring Buildroot for RPI3.

Once **Buildroot** configuration is started, it is necessary to configure the different items. You need to navigate through the different menus and select the elements to install. Table I contains the specific configuration of **Buildroot** for installing it in the Raspberry Pi. Depending on the version downloaded the organization and the items displayed can be different.

[Help]: The Buildroot configuration is an iterative process. In order to set up your embedded Linux system you will need to execute the configuration several times.

Table I: Parameters for Buildroot configuration

Main Item	Subitem	Value	Comments
Target options	Target Architecture	ARM (little endian)	
	Target Architecture Variant	Cortex-A53	
	Target ABI	EABIhf	An embedded-application binary interface (EABI) specifies standard conventions for file formats, data types, register usage, stack frame organization, and function parameter passing of an embedded software program.
	Floating point strategy	NEON/VFPv4	
	ARM instruction Set	ARM	
Build options		Default values	How Buildroot will built the code. Leave default values.

Main Item	Subitem	Value	Comments
Toolchain			Cross Compiler, linker, libraries to be built to compile out embedded application
	Toolchain Type	Buildroot toolchain	Embedded system will be compiled with tools integrated in Buildroot
	C library	uClibc-ng	Library (small size version) containing the typical C libraries used in Linux environments (stdlib, stdio, etc)[RD6]
	Kernel Header s	Same as kernel being built	Source header files of the Linux Kernel.
	Custom Kernel Headers Series	4.9.x	
	uClib configuration file to use?	**package/uclibc/uClibc-ng.config**	
	Enable WCHAR support	Yes	Support for extender set of chars.
	Thread library implementation	Native POSIX Threading (NTPL)	
	Thread library debugging	Yes	Embedded system will have debugeable threads,
	Compile and install uClibC utilities	Yes	
	Binutils Version	binutils 2.29.1	[RD7]. Binutils contains tools to manage the binary files obtained in the compilation of the different applications
	GCC compiler Version	gcc 6.x	GCC tools version to be installed
	Enable C++ support	Yes	Including support for C++ programming, compiling, and linking.
	Build cross gdb for the host	gdb 7.11.x	Includes the support for GDB. GCC debugger.
	Enable MMU support	Yes	Mandatory if building a Linux system
System Configuration			
	System Hostname	**buildroot**	Name of the embedded system
	System Banner	**Linux RPI 3**	Banner
	Passwords encoding	md5	

Main Item	Subitem	Value	Comments
	Init System	Busybox	
	/dev management	Dynamic using devtmpfs only	
	Path to permissions table	**system/device_table.txt**	Text files with permissions for /dev files
	Root filesystem skeleton	Default target skeleton	Linux folder organization for the embedded system
	Root password	**rpi**	
	Run a getty: Port to run a getty	**/dev/ttyAMA0** **console**	Linux device file with the port to run getty (login) process. Uses ttyAMA0 for serial port
	Baud rate to use	115200	
	remount root filesystem read-write during boot	Yes	
	Network interface to configure trough DHCP	eth0	
	Purge unwanted lcoales		
	Custom scripts to run before creating filesystem images	board/raspberrypi3/post-build.sh	
	Custom scripts to run after creating filesystem images	board/raspberrypi3/post-image.sh	
	Extra arguments passed to custom scripts	--add-pi3-miniuart-bt-overlay	
Linux Kernel			
	Kernel version	Custom Git Repository	
	URL of custom repository	**https://github.com/raspberrypi/linux.git**	
	Custom repository version	**33ee56d5927ceff630fbc87e3f5caa409b6ae114**	
	Kernel configuration	Using and in-tree defconfig file	
	Defconfig name	**bcm2709**	
	Kernel binary format	zImage	
	Kernel compression format	Gzip compression	
	Build a Device Tree Blob (DTB)	yes	
	In-tree Device Tree Source file names	bcm2710-rpi-3-b bcm2710-rpi-cm3	

Main Item	Subitem	Value	Comments
	Linux Kernel Extensions	Nothing	
Target Packages			
	Busybox	BusyBox 1.21.x	
	Busybox configuration file to use	**package/busybox/busybox.config**	
	Audio and video applications	Default values	
	Compresssors and decompressors	Default values	
	Debugging, profiling and benchmark	Gdb, gdbserver	
	Developments tools	Default values	
	Filesystem and flash utilities	Default values	
	Games	Default values	
	Graphic libraries and applications (graphic/text)	Default values	
	Hardware handling	Firmware->rpi-firmware Install DTB overlays	
	Interpreters language and scripting	Default values	
	Libraries		
	Miscellaneous	Default	
	Networking applications	**Open ssh->Yes**	
	Package managers Real Time Shell and utilities System Tools Text Editors and viewers	Default	
Filesystem Images			
	ext2/3/4 root filesystem	ext4 size **200M** Compression method **no compression**	
	tar the root filesystem	no compression	
Host utilities			

Main Item	Subitem	Value	Comments
	host u-boot tools	Yes	
	host mtools	Yes	
	host dosfstools	Yes	
	host genimage	Yes	
Legacy config options		Default values	

Once you have configured all the menus you need to exit saving the values (File->Quit).

[Help]: The **Buildroot** configuration is stored in a file named as ".config". You should have a backup of this file.

3.4 Compiling buildroot.

In the Terminal Window executes the following command:

```
rpi@ubuntu:~/Documents/buildroot-xxxx.yy$ make
```

If everything is correct you will see a final window similar to the represented in Fig. 8.

[Time for this step]: In this step buildroot is going to connect, using the internet, to different repositories. After downloading the code, Buildroot is going to compile the applications and generates a lot files and folders. Depending of your internet speed access and the configuration chosen this step could take up to **one hour and half**.

Warning. If you have errors in the configuration of buildroot you could obtain errors in this compilation phase. Check correctly your configuration. Use "make clean" to clean up your partial compilation.

Warning. dl subfolder in your buildroot folder contains all the packages download for the internet. If you want to move your buildroot configuration from one computer to another avoiding the copy of the virtual machine you can copy this folder.

```
dte@ubuntu: ~/Documents/rpi3/buildroot-2016.11.2
Version: Linux version 4.4.21-v7 (dte@ubuntu) (gcc version 5.4.0 (Buildroot 2016
.11.2) ) #1 SMP Fri Feb 3 14:14:25 PST 2017
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
dte@ubuntu:~/Documents/rpi3/buildroot-2016.11.2$
```

Fig. 8: Successful compilation and installation of Buildroot

Buildroot has generated some folders with different files and subfolders containing the tools for generating your Embedded Linux System. Next paragraph explains the main outputs obtained,

3.5 Buildroot Output.

The main output files of the execution of the previous steps can be located at the folder "./output/images". Fig. 9 summarizes the use of **Buildroot**. Buildroot generates a boot loader, a kernel image, and a file system.

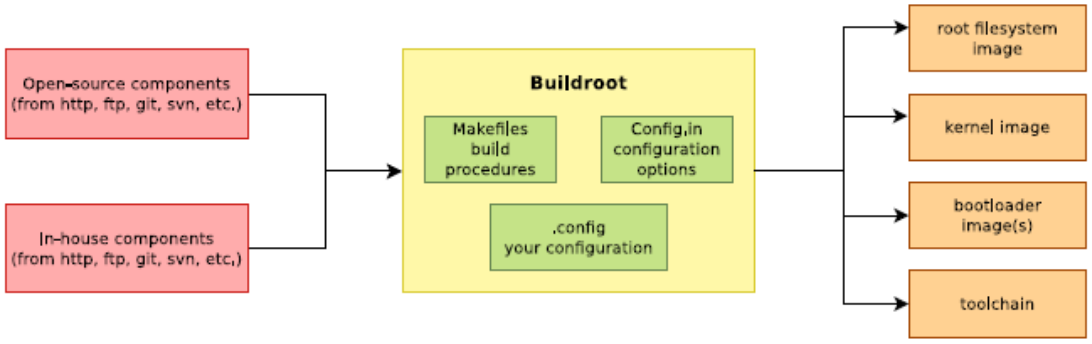


Fig. 9: Schematic representation of the Buildroot tool. Buildroot generates the root file system, the kernel image, the booloader and the toolchain. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/)

In our specific case the folder content is shown in Fig. 10

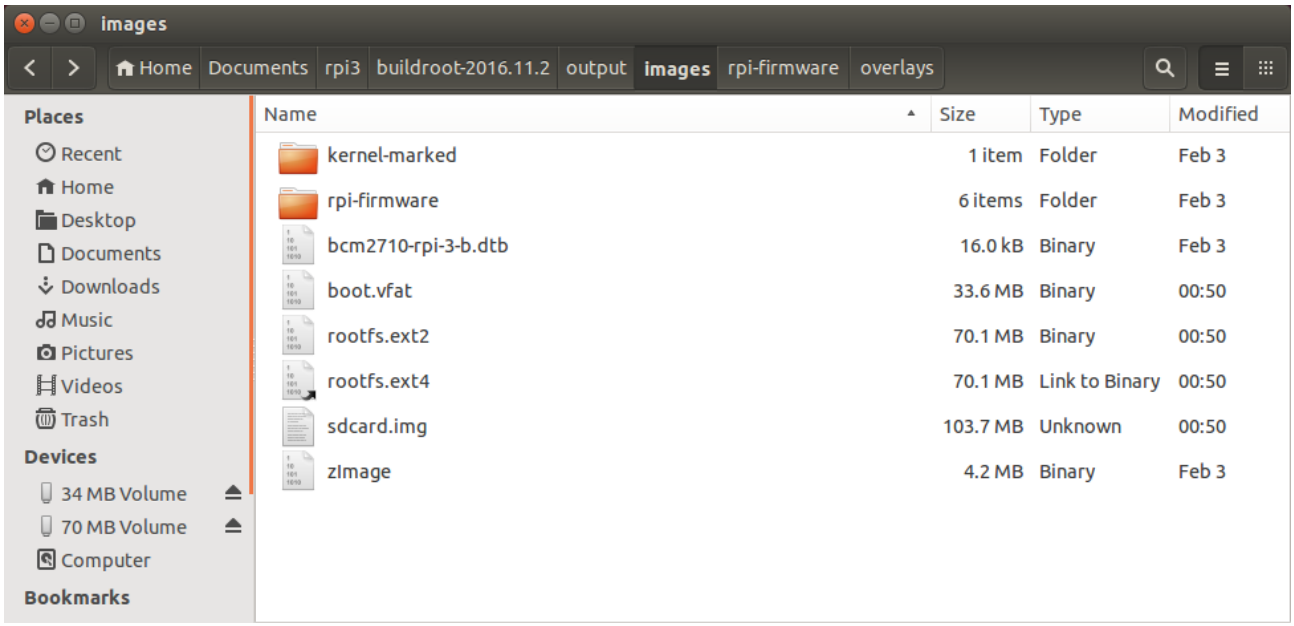


Fig. 10: images folder contains the binary files for our embedded system.

Copy the sdcard.img file to your SDcard using this linux command in the bBuildroot folfer:

```
$ sudo dd if=output/images/sdcard.img of=/dev/sdX
```

3.6 Booting the Raspberry Pi.

Fig. 11 displays a Raspberry Pi. The description of this card, their functionalities, interfaces and connectors are explained in the ref [RD5]. The basic connection requires:

a) To connect a USB to RS232 adapter (provided) to the raspberry-pi expansion header (see Fig. 12 and Fig. 13). This adapter will be provide the serial line interface to be used as console in the Linux operating system.
b) To connect the power supply with micro-USB connector provided (5 v).
c) To connect the Ethernet cable to the RJ45 port.

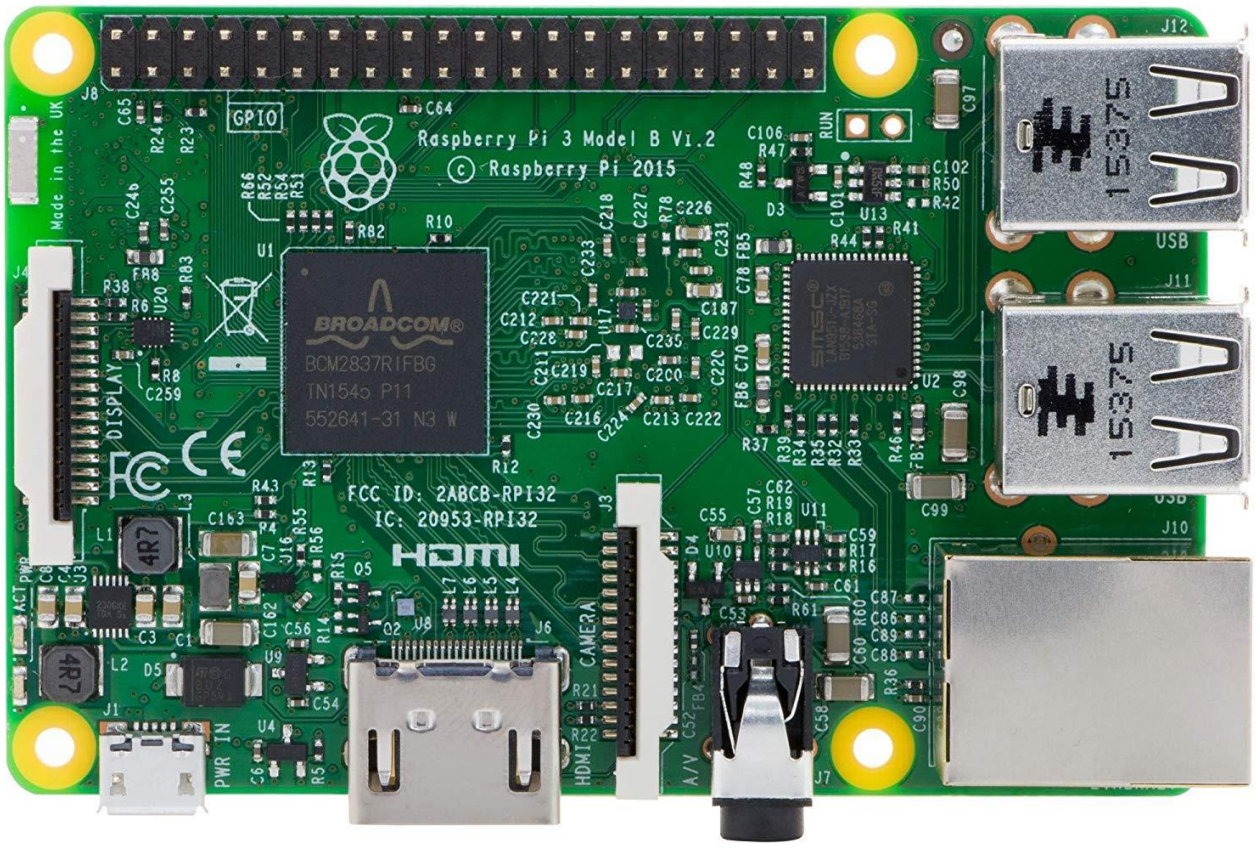


Fig. 11: RaspBerry-Pi 3 Model B hardware with main elements identified.

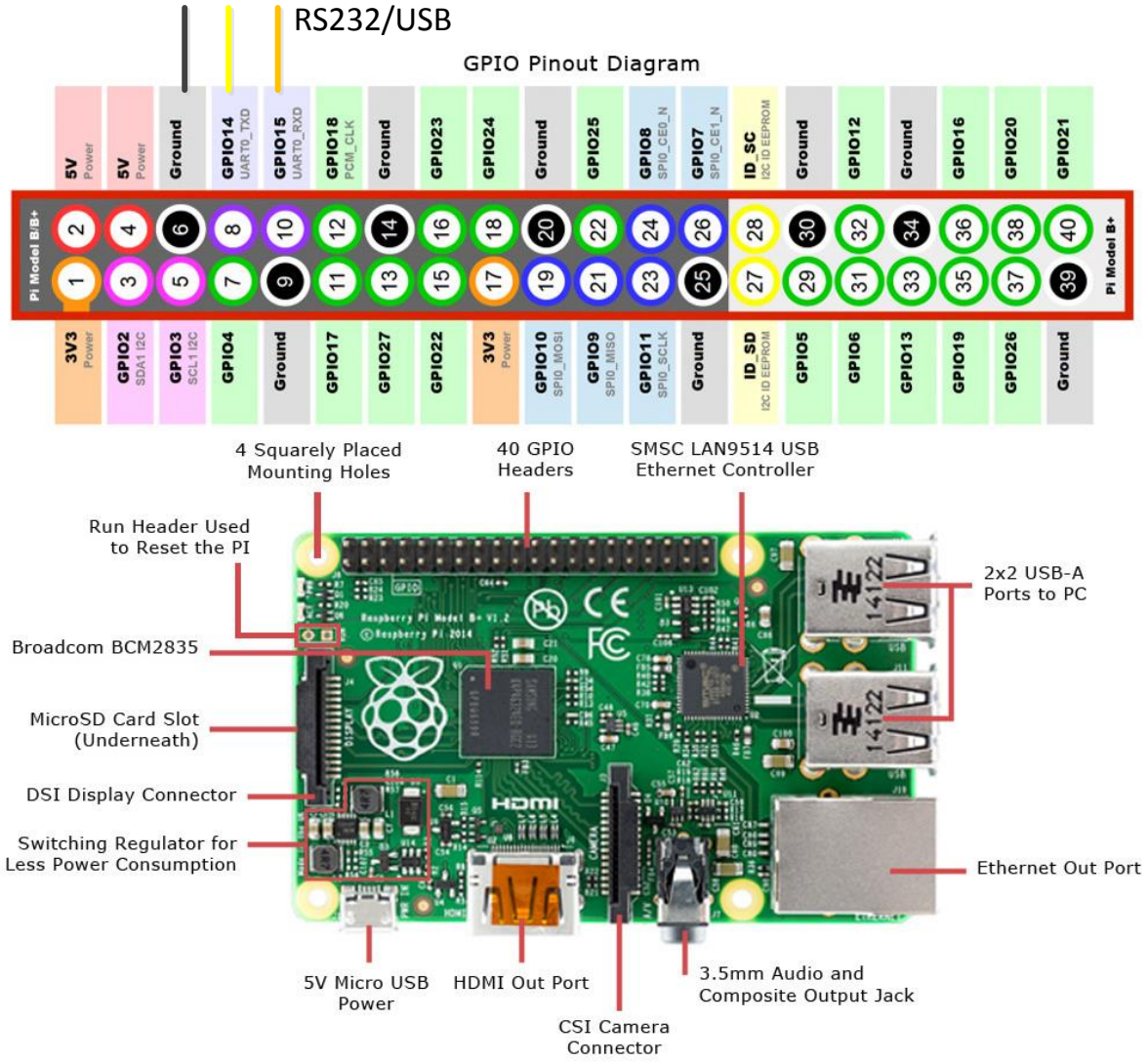


Fig. 12: Raspberry-PI header terminal identification.

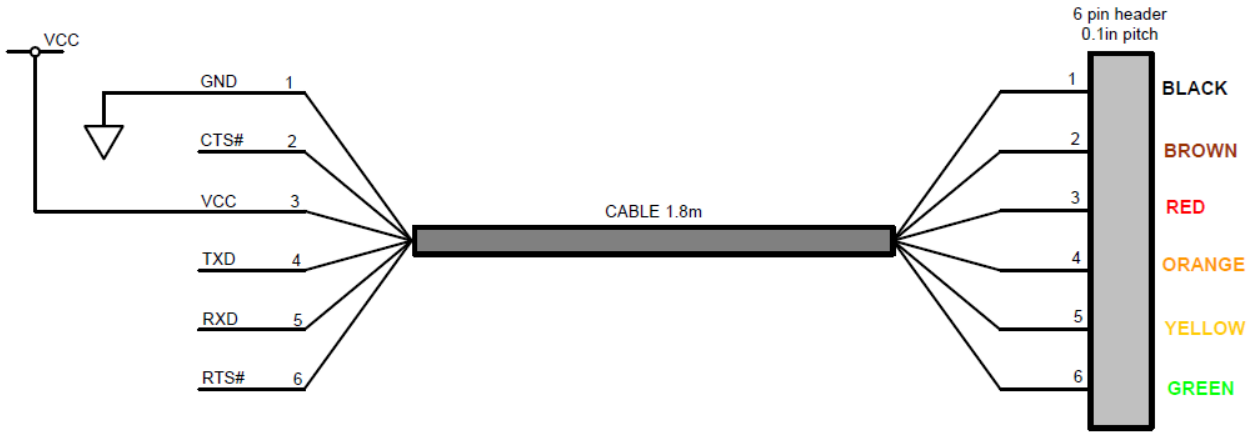


Fig. 13: Identification of the terminals in the USB-RS232 adapter

The booting process of the Raspberry Pi BCM2837 processor is depicted in Fig. 14. Take into account that this System On Chip (SoC), the BCM2837, contains two different processors: a GPU and an ARM processor. The programs bootcode.bin, start.elf are specifically written for GPU and the source code is not available. In fact Broadcom only provides details of this to customer that signs a commercial agreement. The last

executable (start.elf) boots the ARM processor and allows the execution of programs written for ARM such as Linux OS kernel or other binaries such as u-boot.

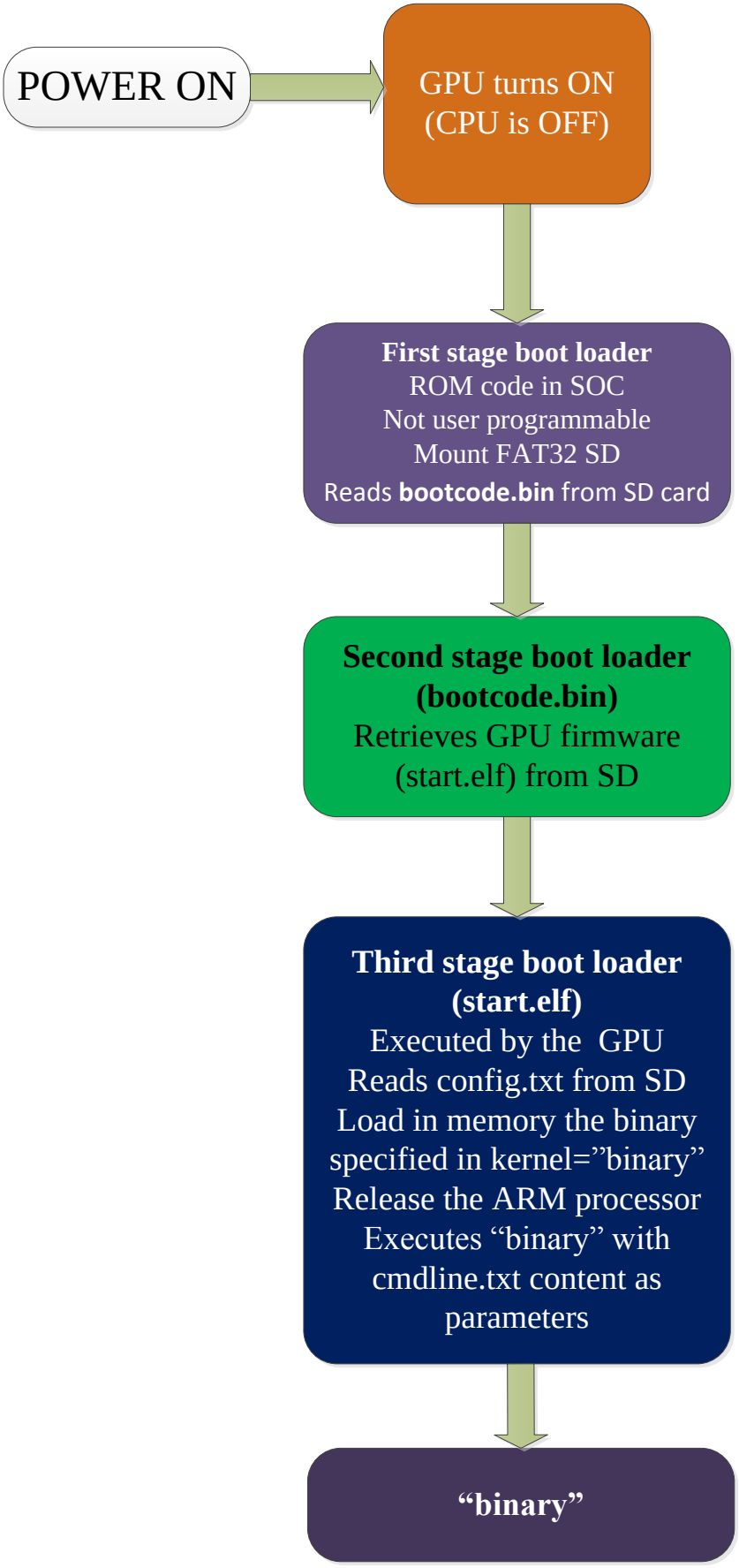


Fig. 14: Booting process for BCM2837 processor in the raspberry-pi.

By default the config.txt file contains these lines (have a look to http://elinux.org/RPiconfig and check the meaning of the different configuration parameters):

```
# Please note that this is only a sample, we recommend you to change it to fit
# your needs.
# You should override this file using a post-build script.
# See http://buildroot.org/manual.html#rootfs-custom
# and http://elinux.org/RPiconfig for a description of config.txt syntax

kernel=zImage

# To use an external initramfs file
#initramfs rootfs.cpio.gz

# Disable overscan assuming the display supports displaying the full resolution
# If the text shown on the screen disappears off the edge, comment this out
disable_overscan=1

# How much memory in MB to assign to the GPU on Pi models having
# 256, 512 or 1024 MB total memory
gpu_mem_256=100
gpu_mem_512=100
gpu_mem_1024=100

# fixes rpi3 ttyAMA0 serial console
dtoverlay=pi3-miniuart-bt
```

In this example, once the ARM is released from reset it executes the zImage application. This binary application is typically the Linux Kernel in zImage format. The parameters that will be passed to the application specified in the kernel=<….> are detailed in the cmdline.txt file. For instance, by default, Buildroot generates this one:

```
root=/dev/mmcblk0p2 rootwait console=tty1 console=ttyAMA0,115200
```

In the Linux machine, or in the windows one, open a Terminal and execute the program putty with sudo rights (sudo putty), in a seconds a window will be displayed. Configure the parameters using the information displayed in Fig. 15 (for the specific case of putty), and then press “Open”. **Apply the power to the raspberry PI** and you will see the booting messages.

[Serial interface identification in Linux]: In Linux the serial devices are identified typically with the names /dev/ttyS0, /dev/ttyS1, etc. In the figure the example has been checked with a serial port implemented with an USB-RS232 converter. This is the reason of why the name is /dev/ttyUSB0. In your computer you need to find the identification of your serial port. Use Linux **dmesg** command to do this.

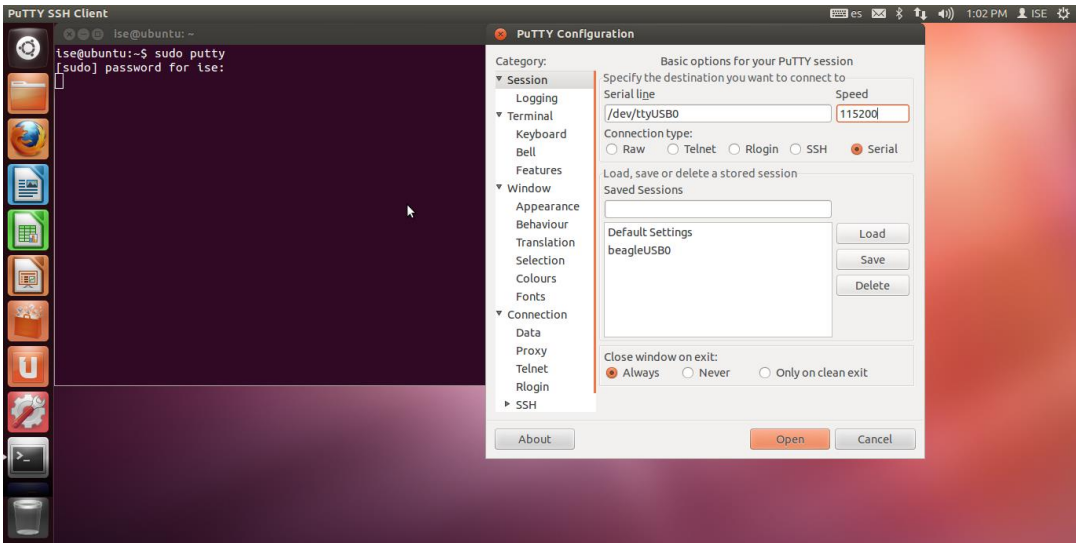


Fig. 15: Putty program main window.

After some seconds you will see a lot messages displaying in the terminal. Linux kernel is booting and the operating system is running its configuration and initial daemons. If the system boots correctly you will see an output like the represented in Fig. 16. Introduce the user name root and the Linux shell will be available for you.

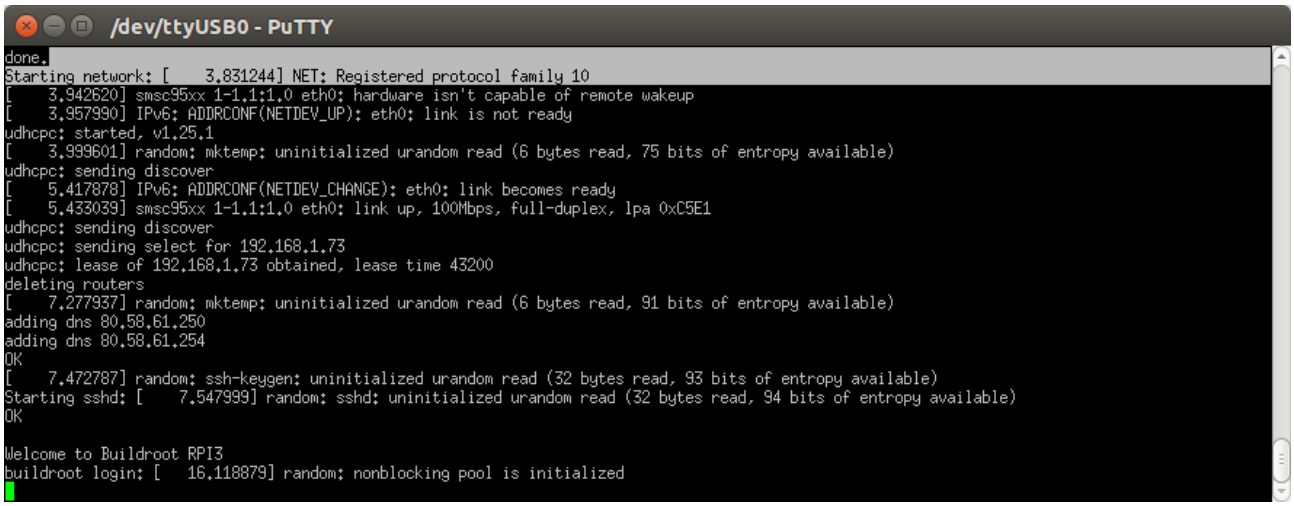


Fig. 16: Linux Running

[DHCP Server]: The DHCP server providing the IP address to the RPI should be active in your network. In the UPM ETSIST labs the IP is assigned using the RPI's MAC address. Check with your instructor the IP assigned to your RPI. If you are using the RPI at home, the DNS server is running in your router. The method used by this should be different from one manufactures to others. If you want to know the IP address assigned you have two options: use a serial cable connected to the RPI or check the router status web page and display the table of the DHCP clients connected. Looking for the MAC in the list you will obtain the IP.

3.7 Connecting the RPI to the network

3.7.1 Inspecting the configuration of the network interface automatically generated

Inspect the content of /etc/network/interfaces and /etc/init.d/S40network. You will see a content similar to this in the intrefaces file:

```
# interface file auto-generated by buildroot

auto lo
iface lo inet loopback

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

This configuration activates the use of eth0 with DHCP support. Test the connectivity trying to connect to another computer in the laboratory. Use the ping command.

[Help]: If you execute the ping command in the raspberry trying to connect with a computer in the laboratory probably you obtain a connection timeout. Consider that computer running windows could have the firewall activated. You can also try to run the ping in a windows computer or in Linux virtual machine. In this case the rpi doesn't have a firewall running and the connection should be ok.

[Question] What is the MAC address of your RPI? Is this MAC the same that your instructor has given you? Use dmesg command to see the kernel boot parameters and try to identify the method used to get the MAC address from the hardware.

3.7.2 Adding support to WIFI

3.7.2.1 Adding mdev support to Embedded Linux

Execute make menuconfig or make xconfig in the Buildroot root folder. Navigate to System Configuration and activates in "/dev management" the "Dynamic using devtmpfs+mdev" option. The folder ./package/busybox contains two files named S10mdev and mdev.conf. These files have to be added to the target filesystem. This step is done adding these commands to the ./board/raspberrypi3/post-build.sh script:

```
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
```

[mdev] mdev provides a method to add or remove homenutplug devices in Linux.

3.7.2.2 Adding the Broadcom firmware support for Wireless hardware.

The hardware element included in the RPI-3 for the Wireless communication is implemented with the BCM43438 chip. It is needed to include the packages with the chip firmware and the wireless utilities.

1. Execute “make menuconfig” or “make xconfig”. Navigate to “Target Packages->Hardware Handling->Firmware” and select the “rpi-wifi-firmware”.
2. Before compiling Buildroot we need to add more software supporting the configuration of the WIFI.
    a. Navigate to “Target Packages->Networking Applications” and select
        - “crda”
        - “ifupdown scripts”
        - “iw”
        - “wireless-regdb”
        - “wireless tools”
        - “wpa_supplicant”
            1. “Enable EAP”
            2. “Enable WPS”
            3. “Install wpa_cli binary”
            4. “Install wpa_client shared library”
            5. “Instal wpa passphrase binary”
    b. Add these lines to ./board/rapsberrypi3/post-build.sh.

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

    c. Modify the file ./board/raspberrypi3/interfaces adding this content:

```
auto lo
iface lo inet loopback

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
```

    d. Create the file ./board/raspberrypi3/wpa_supplicant.conf with this content (ask professors about the values to be provided as SSID and Key-passwd.

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

3. Perform a make and burn the new image in the SDcard. Boot the Raspberry and check that you can connect to the wireless network.

4 USING INTEGRATED DEVELOPMENT ENVIRONMENT: ECLIPSE/CDT

4.1 Adding cross-compiling tools to PATH variable.

Using a text editor, edit the **.profile** file (available at your home directory). Add a line at the end of the file containing: PATH="<your buildroot installation>/output/host/usr/bin:$PATH". This add to the PATH environment variable the location of the cross-compiling tools. You must logout and login again.

4.2 Cross-Compiling applications using Eclipse.

How a program will be compiled? Remember that we are developing cross applications. We are developing and compiling the code in a Linux x86 machine and we are executing it in an ARM architecture (see Fig. 17).

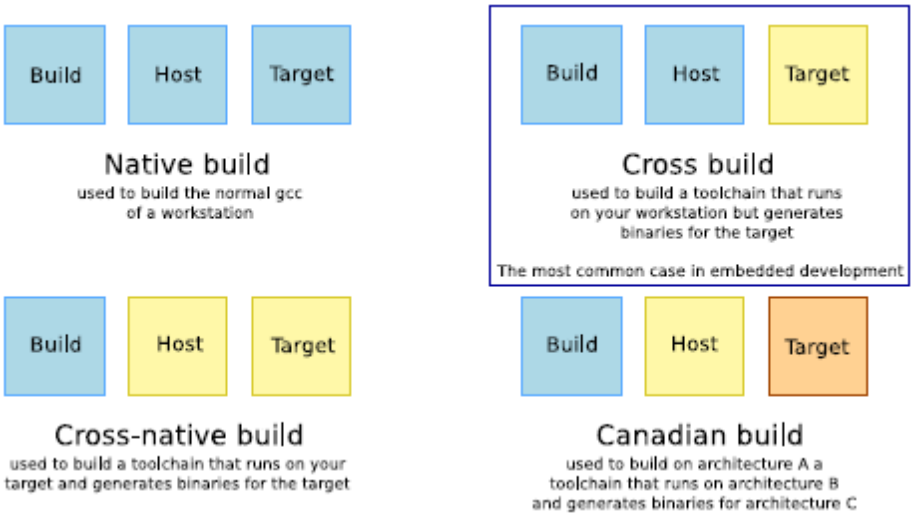


Fig. 17: Summary of the different configurations for developing applications for embedded systems. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/)

The first question is where the cross-compiler is located. The answer is this: in the folder "<buildroot>/output/host/usr/bin". If you inspect the content of this folder you can see the entire compiling, linking and debugging tool (see Fig. 18). These programs are executed in your x86 computer but they generate code for ARM processor.

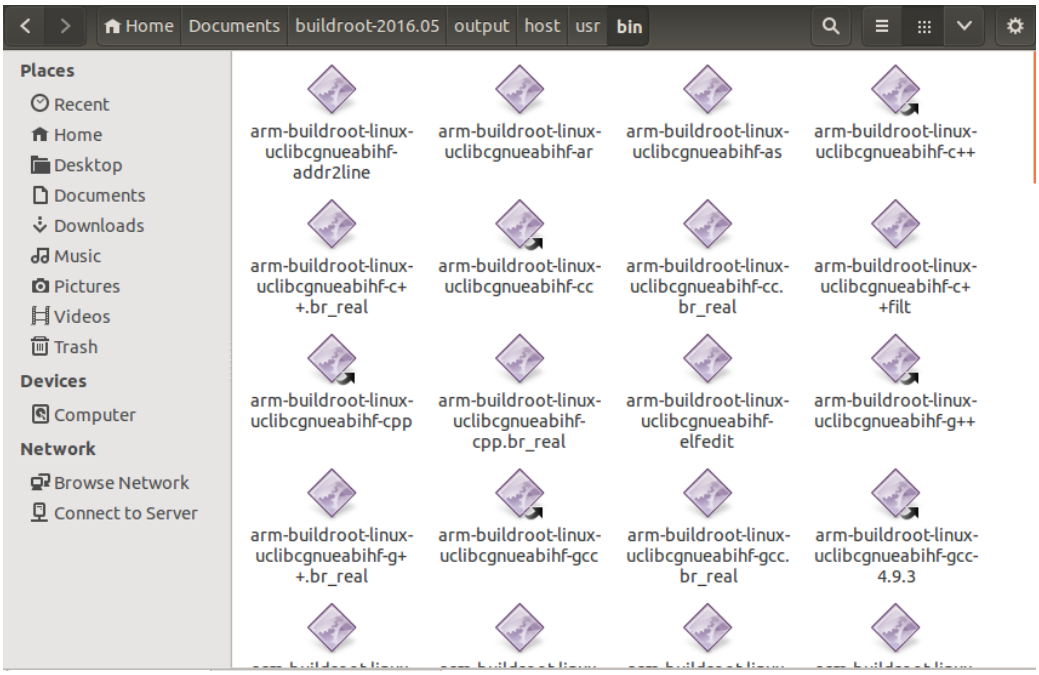


Fig. 18: Cross compiling tools installed in the host computer

In a Terminal window start Eclipse with the following command:

```
xxxx@ubuntu:~$ eclipse
```

The popup window invites you to enter the workspace (see Fig. 19). The workspace is the folder that will contain all the eclipse projects created by the user. You can have as many workspaces as you want. Please specify a folder in your account.

[Help]: The figures displayed in the following paragraphs can be different depending on the Eclipse version installed.

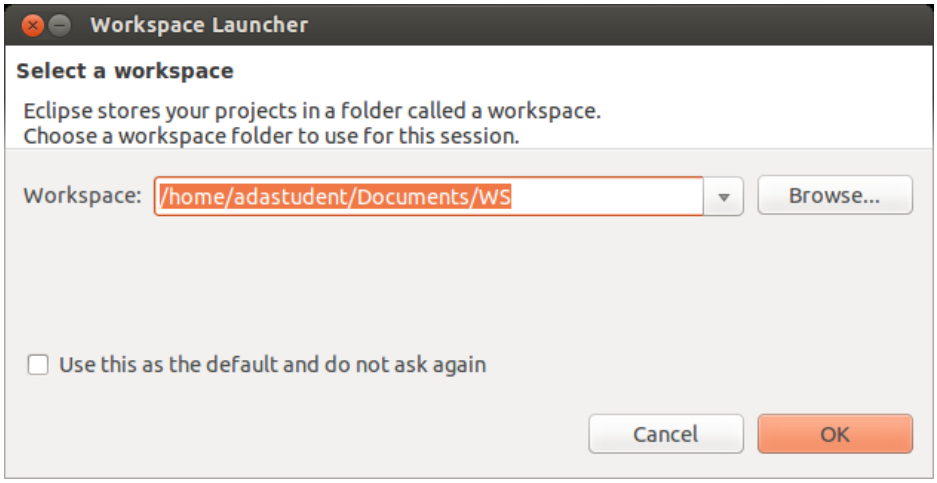


Fig. 19: Selection of the workspace for Eclipse. Use a folder in your account.

Select Ok and the welcome window of Eclipse will be shown (Fig. 20). Next, close the welcome window and the main eclipse window will be displayed (Fig. 21).

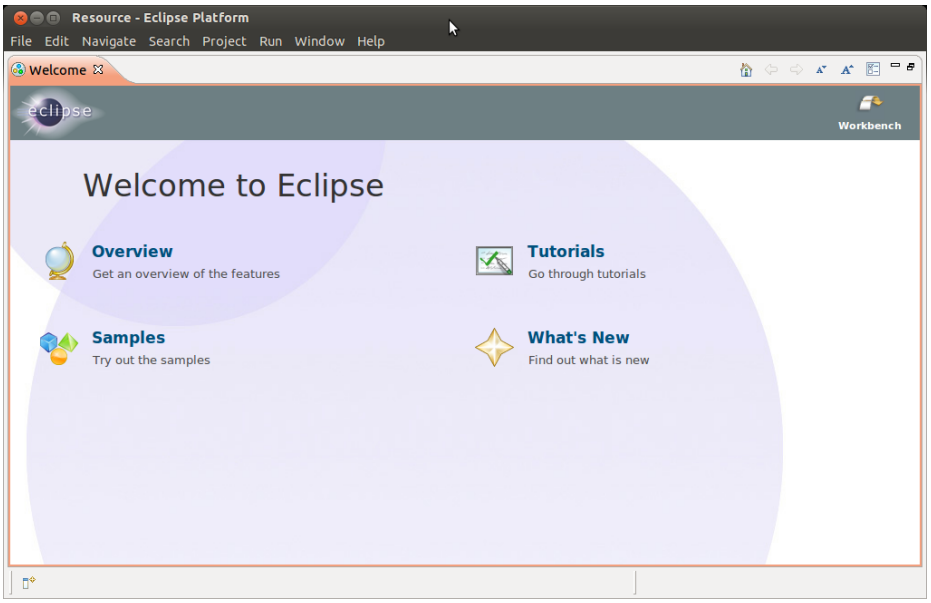


Fig. 20: Eclipse welcome window.

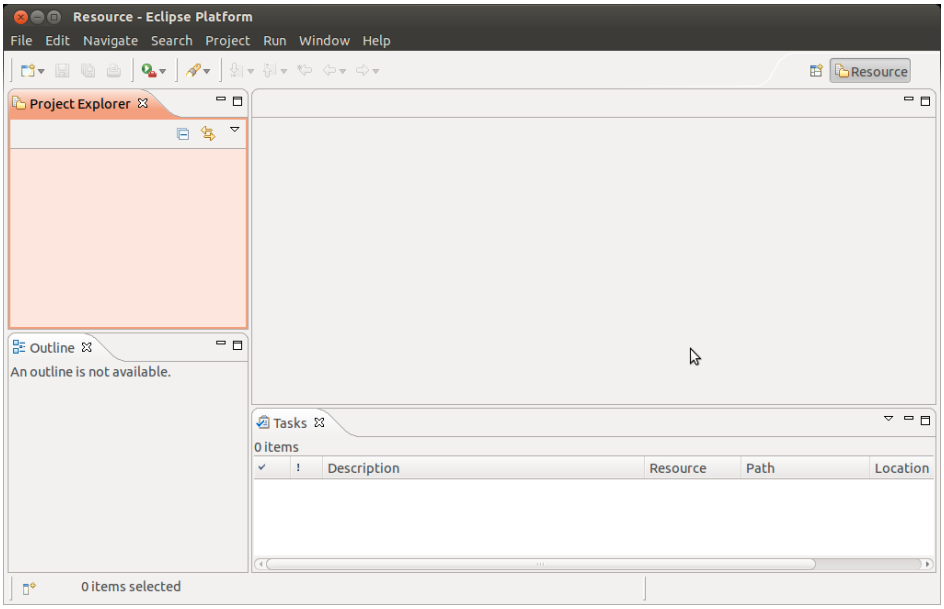


Fig. 21: Eclipse main window.

Create an Eclipse C/C++ project (File->New->Project->C/C++project->C project) selecting the hello world example (see Fig. 22). Specify the project name and the toolchain to be used. In this case a Cross GCC. Press Next.

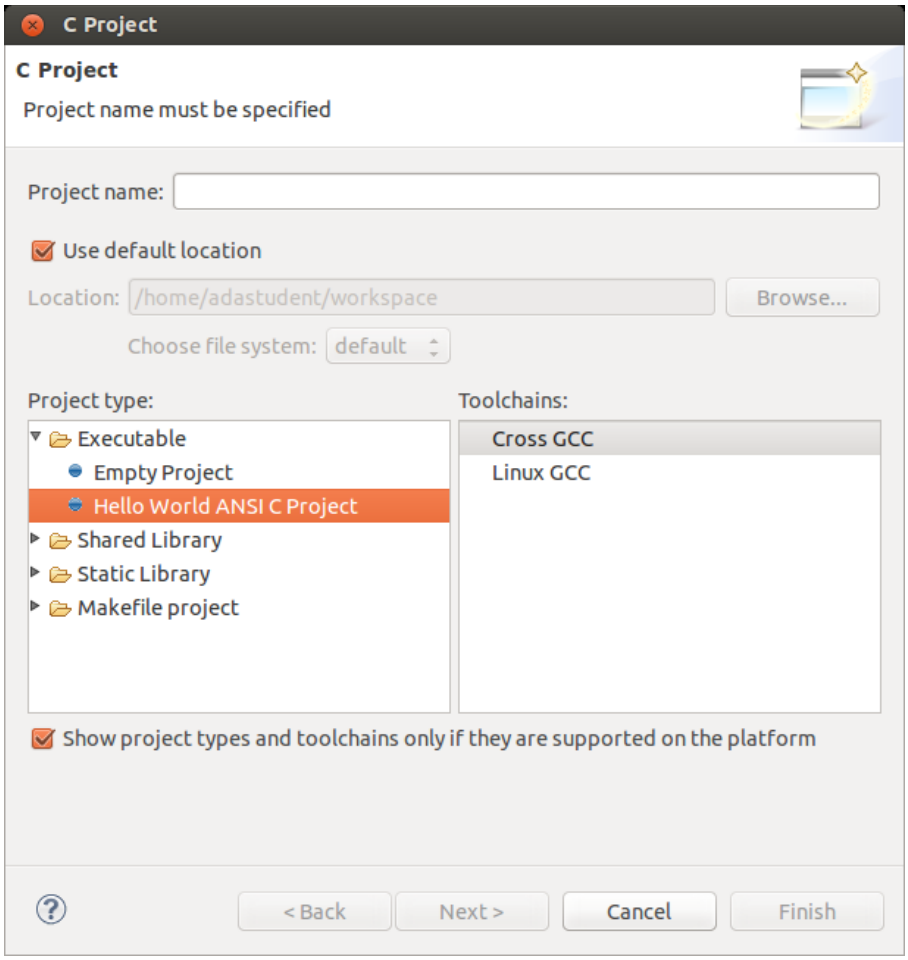


Fig. 22: Basic C project creation in Eclipse

There is a window (Fig. 23) requesting the Cross Compiler prefix and path, leave both inputs blank and click on the Finish button. You will obtain your first project created with eclipse.

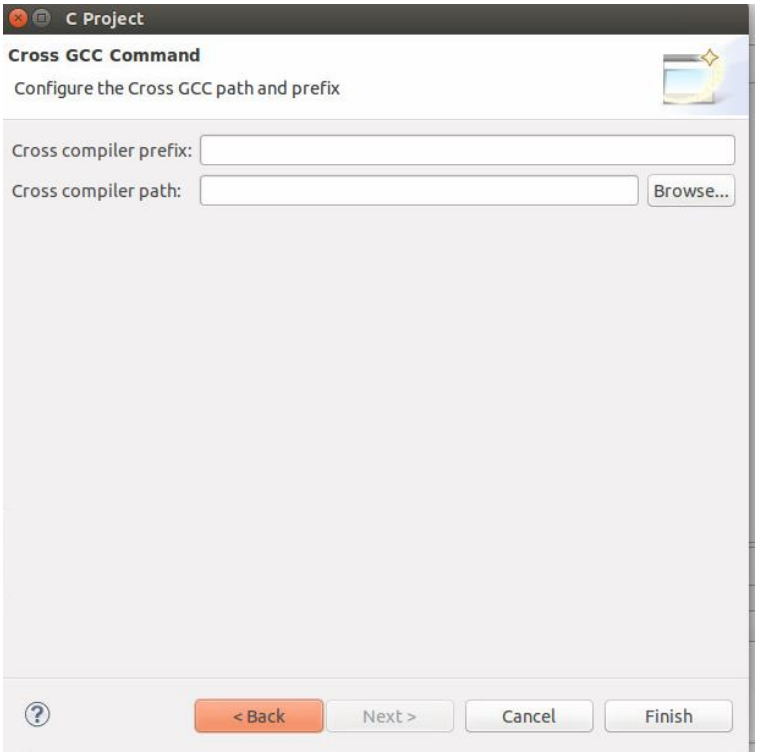


Fig. 23: Cross-compiler prefix and path window.

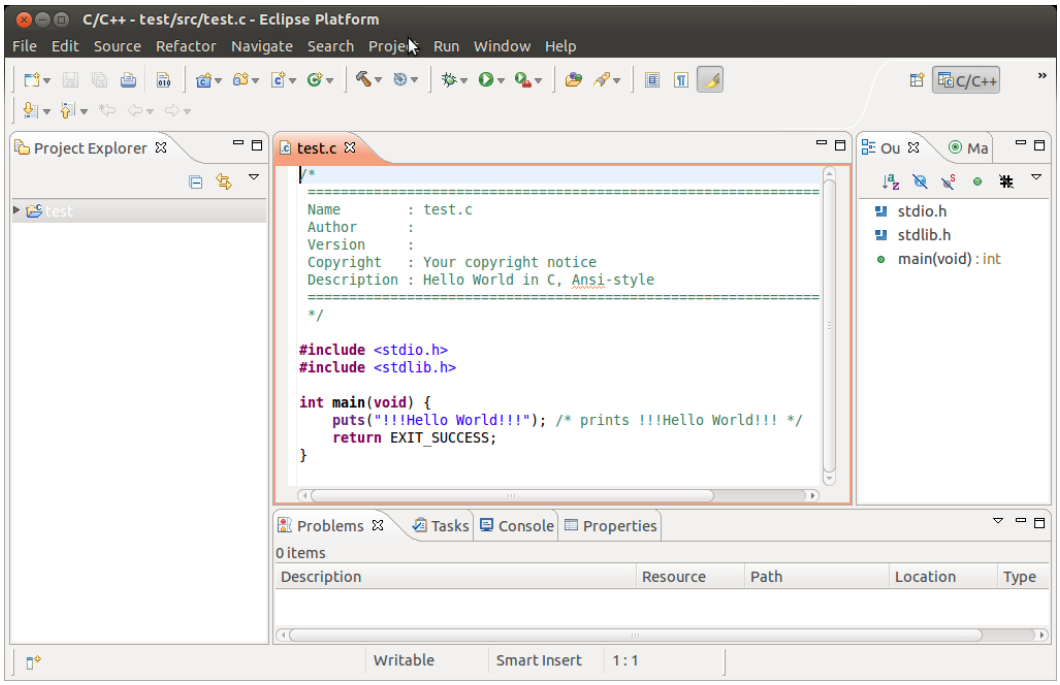


Fig. 24: Hello world example.

The next step (mandatory) is the Eclipse project configuration for managing the Cross-tools. In Project -> Properties configure the C/C++ Build Setting as the Fig. 25 and Fig. 26 shown. Pay attention that Prefix requires a string ending in a hyphen.

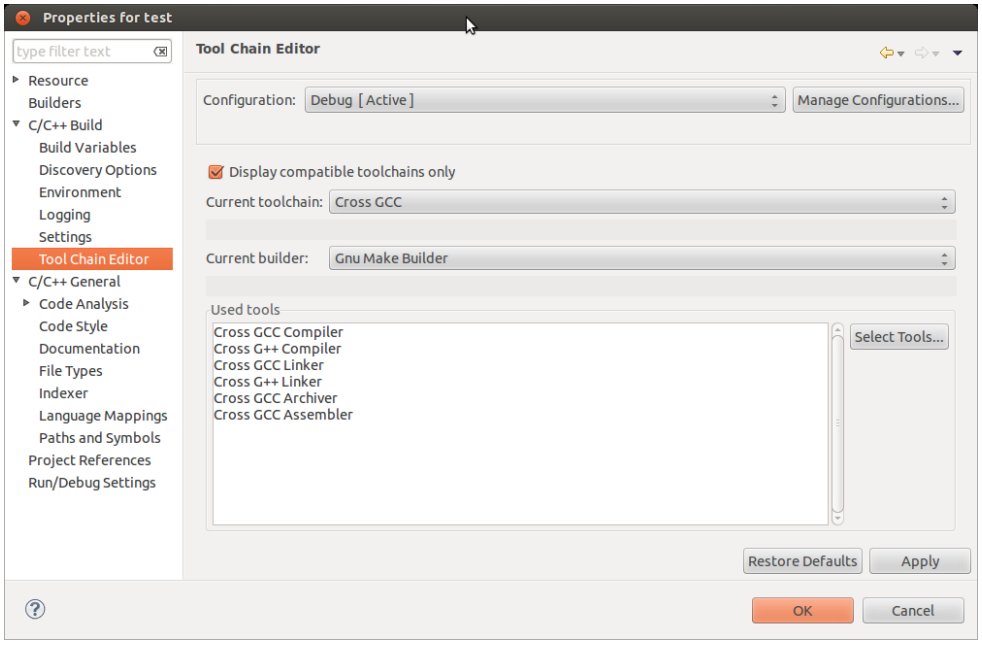


Fig. 25: Tool Chain Editor should be configured to use Cross GCC.

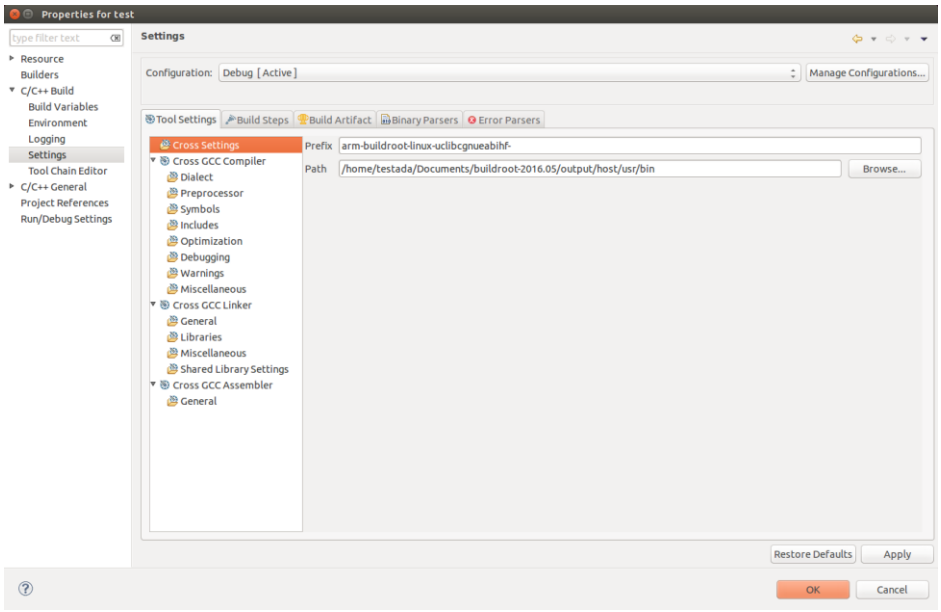


Fig. 26: Cross tools locate on (path).The path shown in this figure is an example.Use always the path of your toolchain.

The next step is to configure the search paths for the compiler and linker, and the different tools to use. Complete the different fields with the information included in Fig. 27 and Fig. 28. Please consider the paths. The figures are showing examples for a specific user account.

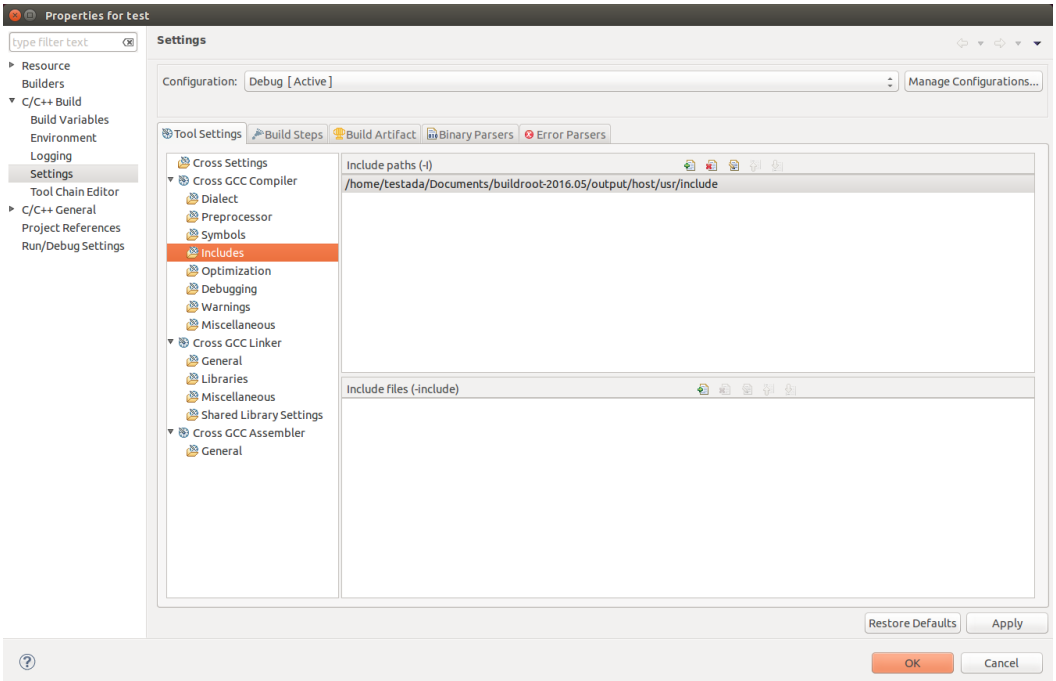


Fig. 27: Include search path.

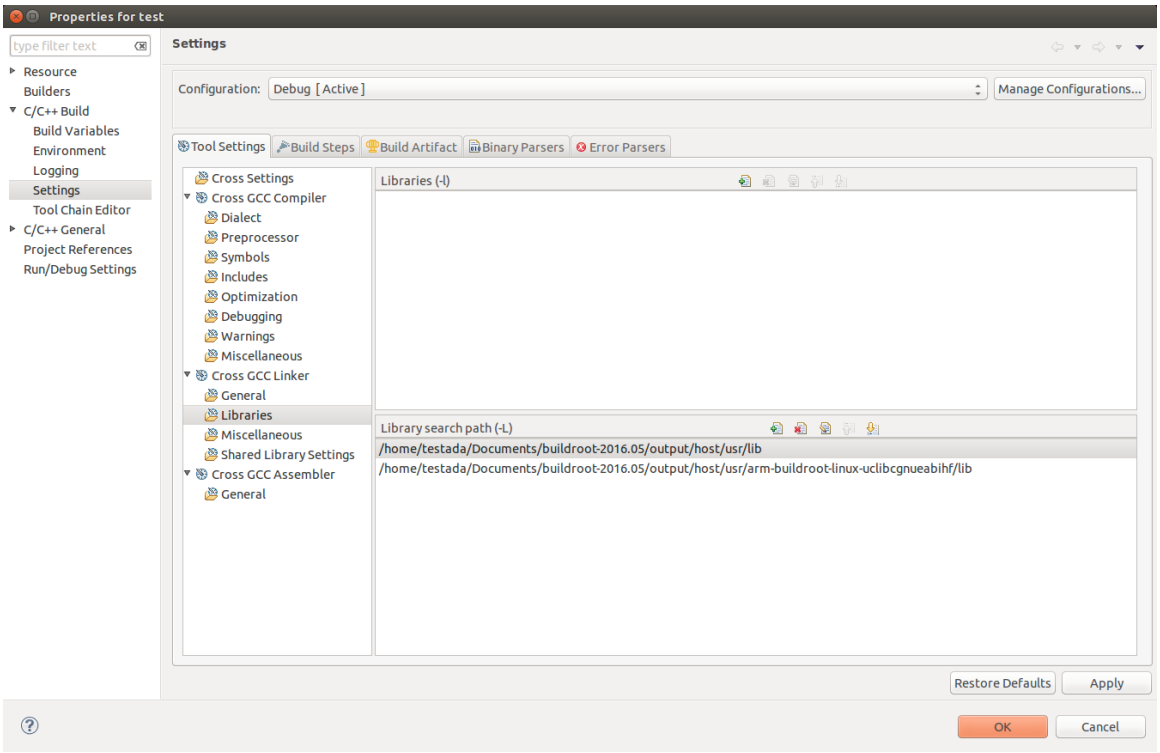


Fig. 28: Libraries search path.

Once you have configured the cross chain in Eclipse you can build your project using Project->Build Project. If everything is correct you will see the eclipse project as represented in Fig. 30.

[Console in Eclipse]: Have a look to the messages displayed in the Console. You will see how eclipse is calling the cross compiler with different parameters.

In order to copy the executalbe to the target you have different options. You can use the linux application called “scp” or other similar applications. In our case we are going to use “Connect to Server….” utility included in ubuntu (under Places menu). Specify in Server Address ssh://<ip address>

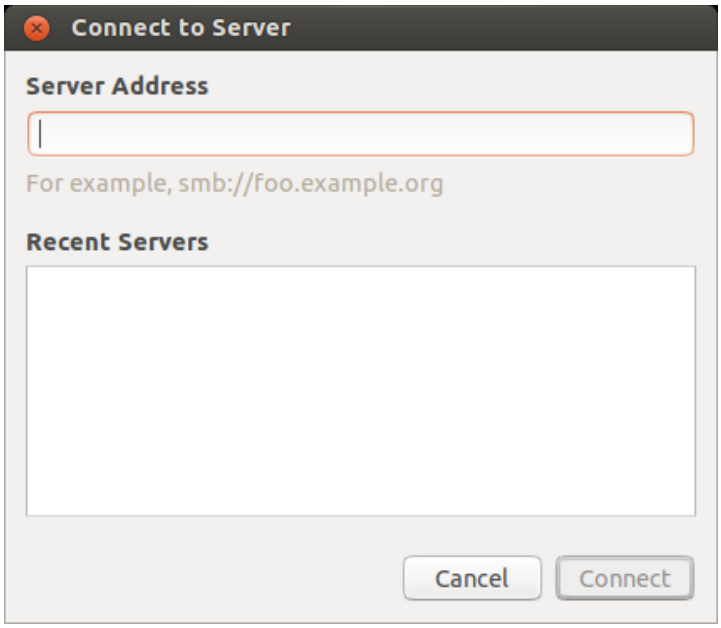


Fig. 29: Pop up window when executing "Connect to Server"

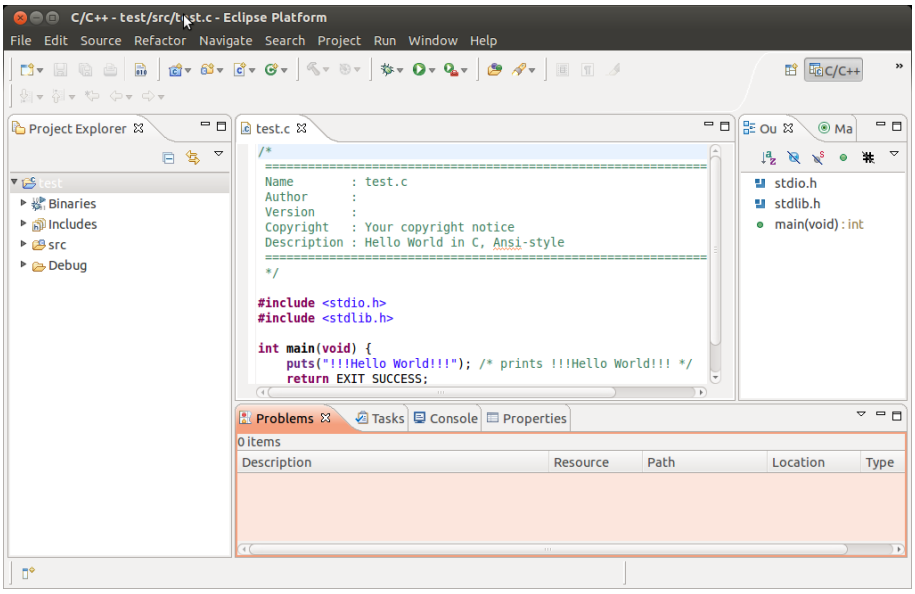


Fig. 30: Eclipse project compiled (Binaries has been generated).

You can run the program in the Raspberry PI using putty (remember that once you have a network connection available in the DE1-SoC you can also use putty to connect to it).

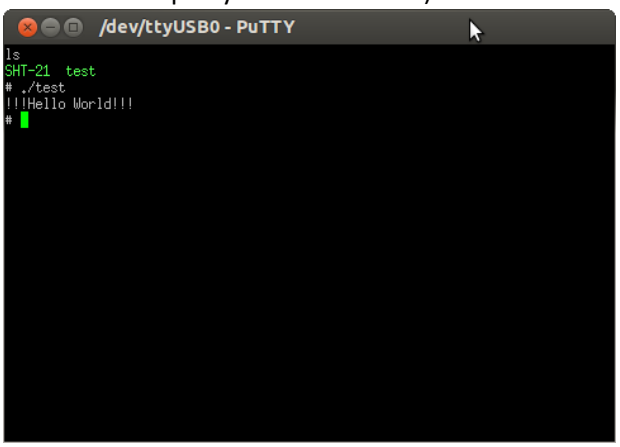


Fig. 31: Run test program in Raspberry PI

Warning. If you experiment problems using ssh, delete the .ssh folder in your home directory.

4.3 Automatic debugging using gdb and gdbserver.

You can directly debug the program running in the DE1-SoC using Eclipse. There are two methods to do it: manually and automatically. In the manual method, firstly, you need to copy the executable program to the DE1-SoC, change the file permissions to "executable" and execute the program to be debugged using gdbserver utility. Of course this is a time consuming process and very inefficient. The alternative solution is to use the automatic debugging. In order to debug your applications we need to define a debug session and configure it. Firstly, Select Run->Debug Configurations and generate a new configuration under C/C++ Remote Application. You need to complete the different tabs available in this window. The first one is the main tab (see Fig. 32). You need to configure here the path to the C/C++ application to be debugged, the project name, the connection with the target (you will need to create a new one using the IP address of your BBB), the remote path where your executable file will be downloaded, and the mode for the debugging (Automatic Remote Debugging Launcher). Secondly, in the argument tab you can specify the argument of your executable program. Very important here is that you can also specify the path of the working directory where the executable will be launched.

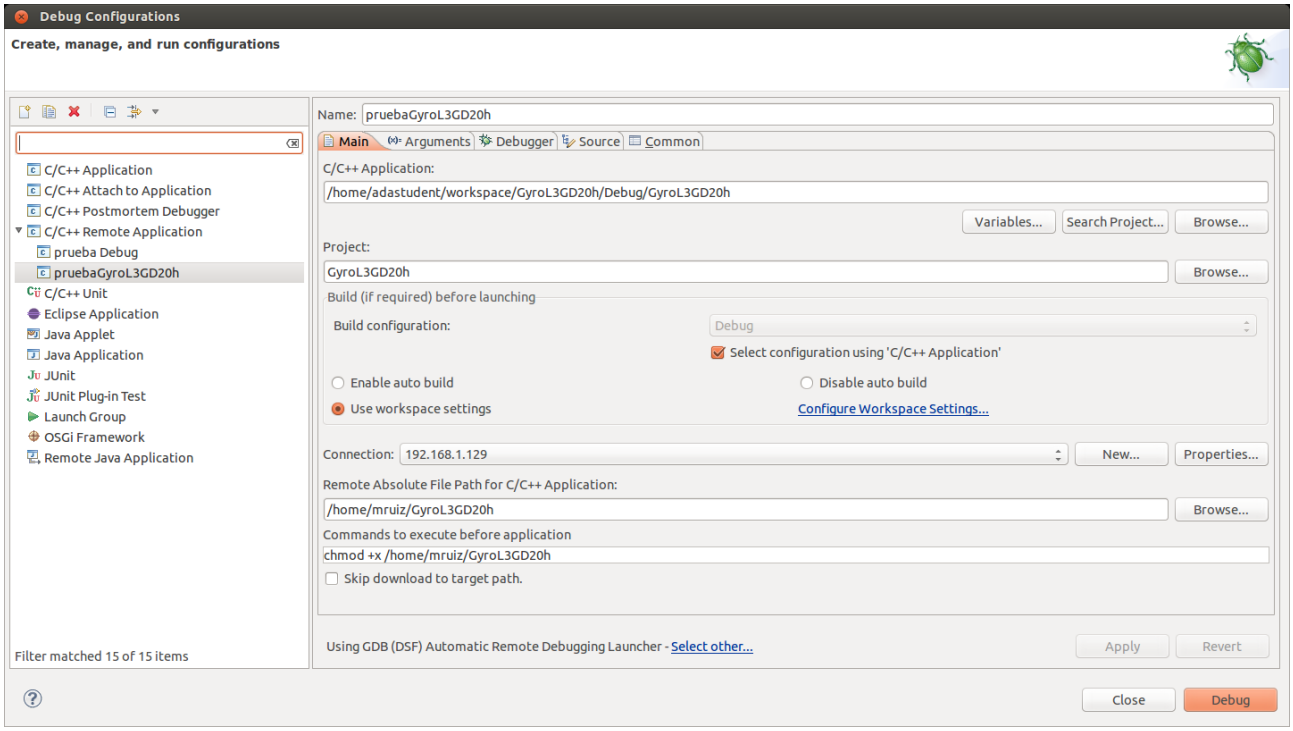


Fig. 32: Creating a Debug Configuration

In the debugger window (main tab) you need to configure the path of your gdb application. Remember that we are working with a cross-compiler, cross debugging, therefore, you need to provide here the correct path of your gdb. The GDB command file must be specified, providing a path with an empty file. In the Gdbserver settings tab you need to provide path to the gdbserver in the target and the port used (by default 2345).

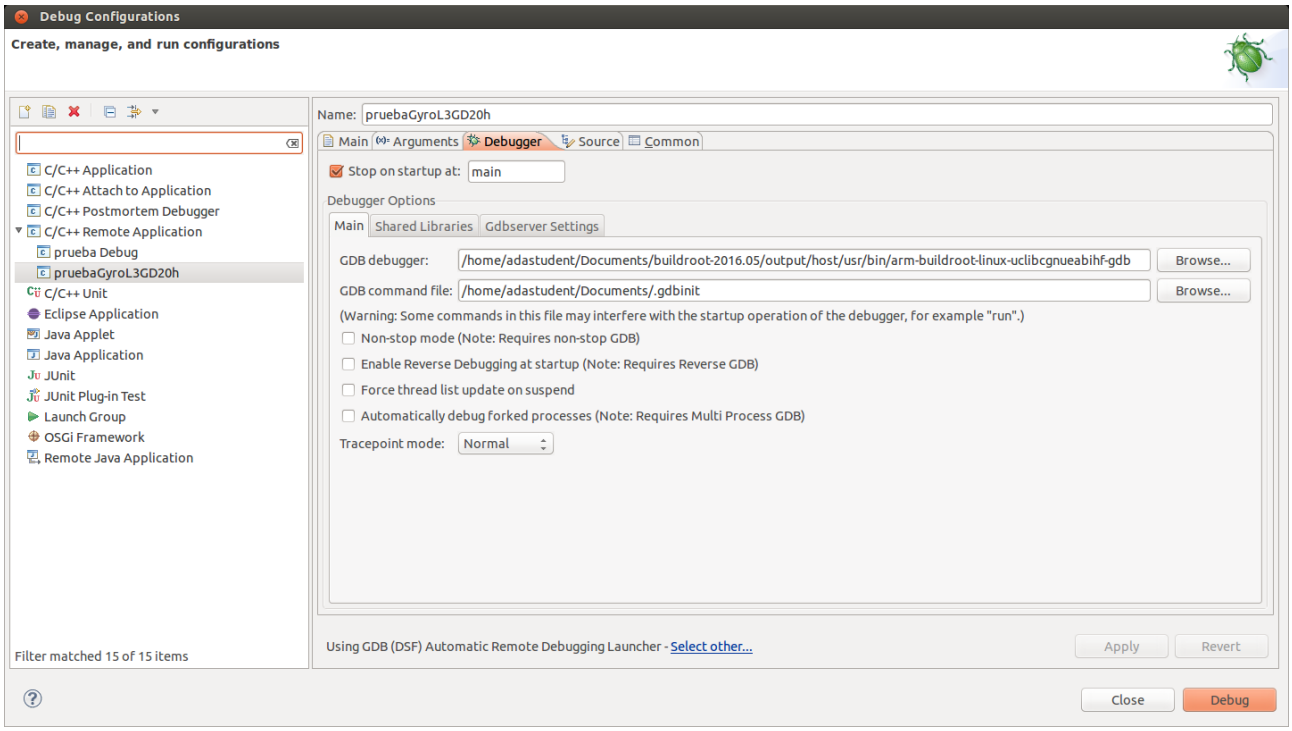


Fig. 33: Debug configuration including the path to locate the cross gdb tool.

Now, press Debug in Eclipse window and you can debug remotely your application.

5 PREPARING THE LINUX VIRTUAL MACHINE.

5.1 Download VMware Workstation Player.

The link https://www.vmware.com/support/pubs/player_pubs.html contains documentation describing the installation and basic use of VMware Workstation Player. Follow the instructions to setup the application in your computer.

5.2 Installing Ubuntu 14.04 LTS as virtual machine.

[Ubuntu version]: It is mandatory to install Ubuntu 14.04 version. 16.04 version will generate compatibility problems.

The first step is to download Ubuntu 14.04 (32 bit PC-i386) from Ubuntu web site using this link: http://releases.ubuntu.com/14.04/ . You will download an ISO image with this Linux operating System. Run WMware player and install Ubuntu using the VMWare player instructions. Consider the following when creating the virtual machine: you need at least 150Gbytes of hard disk space (in multiple files), 3GByte of RAM, and if possible 2 processors. The installation time will be half an hour more or less depending of your computer. Moving a virtual machine form one computer to another is a time consuming task, therefore, take this into account to minimize the development time.

5.3 Installing synaptic

If you need to install software packages you can do it using the command apt-get. Another alternative process is the use of synaptic utility. In order to use it you need to install it using this command:

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

Once installed you can search and execute the synaptic program. When you click two times over the package it will show all the dependent packages than would be installed.

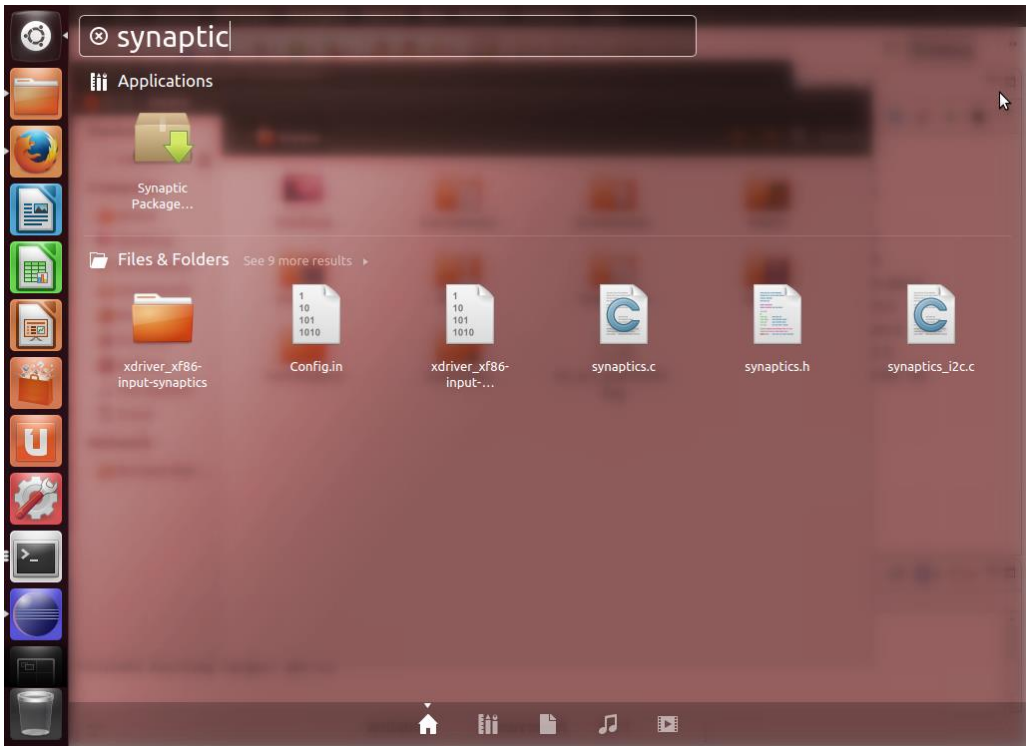


Fig. 34: Synaptic program from Dash

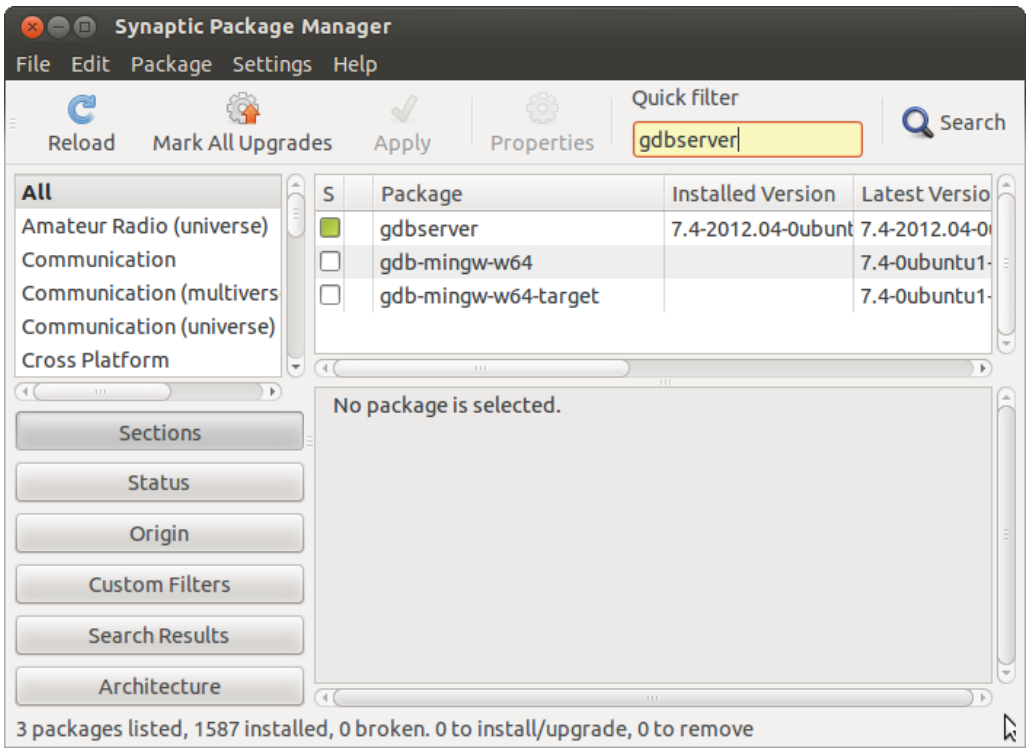


Fig. 35: Synaptic windows

5.4 Installing putty

You need to install:

- putty

5.5 Installing packages for supporting Buildroot.

Using buildroot requires some software packages that have to be installed in the VM. These are listed in this link http://buildroot.uclibc.org/downloads/manual/manual.html#requirement. You need to install:

- g++
- libqt-4-dev
- git

5.6 Installing packages supporting Eclipse

You need to install:

- eclipse-cdt (eclipse C/C++ programming)
- eclipse-rse (eclipse remote explorer)
- eclipse-cdt-launch-remote (eclipse for remote debugging)