

Dpto. de Telemática y Electrónica
Universidad Politécnica de Madrid

Embedded Linux Systems

Using Buildroot for building Embedded Linux
Systems (BeagleBone Black)
V1.2

Mariano Ruiz
Francisco Javier Jiménez

2016



Table of contents

1	SCOPE.....	4
1.1	Document Overview	4
1.2	Acronyms	4
2	REFERENCED DOCUMENTS	5
2.1	References.....	5
3	LAB1: BUILDING LINUX USING BUILDROOT	6
3.1	Elements needed for the execution of these LABS.....	6
3.2	Starting the VMware.....	6
3.3	Configuring Buildroot.....	10
3.4	Compiling buildroot.	14
3.5	Buildroot Output.....	15
3.6	Configuring the Linux kernel parameters	16
3.7	Booting the BeagleBone.	17
3.8	Basic test your embedded Linux System	22
3.9	Understanding the boot process with u-boot in the BeagleBone	23
3.10	Booting the BeagleBone using a script.	24
3.11	Additional details about the configuration of BBB in u-boot.	24
3.12	Configuring the network interface in the BBB.	25
4	LAB2: USING INTEGRATED DEVELOPMENT ENVIRONMENT: ECLIPSE/CDT	26
4.1	Adding cross-compiling tools to PATH variable.....	26
4.2	Cross-Compiling application using Eclipse.	26
4.3	Automatic debugging using gdb and gdbserver.	34
5	PREPARING THE LINUX VIRTUAL MACHINE.	36
5.1	Download VMware Workstation Player.....	36
5.2	Installing Ubuntu 14.04 LTS as virtual machine.....	36
5.3	Installing packages for supporting Buildroot.	36
6	ANNEX I: UBUNTU 14.04 LTS PACKAGES INSTALLED.	37
6.1	List of packages used by Buidlroot	37
6.2	Typical Ubuntu virtual machine problems.....	38
6.2.1	Manually installation of VMware client tools in a Linux Virtual Machine.	38
6.2.2	Ubuntu presents a black screen after graphical login	38
6.2.3	Ubuntu is using us keyboard and not Spanish one.....	38

Table of figures

Fig. 1: Main screen of VMware player with some VM available to be executed.	6
Fig. 2: Ubuntu Virtual Machine login screen.	7
Fig. 3 Buildroot home page.	7
Fig. 4: Downloading buildroot source code.	8
Fig. 5: Buildroot folder (the folder name depends on the version downloaded).	8
Fig. 6: Dash home, Terminal application	9
Fig. 7: Buildroot setup screen (make xconfig). The content of this windows depends on the parameter selected.	10
Fig. 8: Successful compilation and installation of Buildroot	15
Fig. 9: Schematic representation of the Buildroot tool. Buildroot generates the root file system, the kernel image, the bootloader and the toolchain. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/)	15
Fig. 10: images folder contains the binary files for our embedded system.	16
Fig. 11: Main window for configuring the Linux kernel.	17
Fig. 12: BBB.	18
Fig. 13: BeagleBone Black block diagram	18
Fig. 14: Powering BBB with USB	19
Fig. 15: BBB serial line header terminal identification.	19
Fig. 16: Identification of the terminals in the USB-RS232 adapter	19
Fig. 17: Booting process for BBB.	20
Fig. 18: BeagleBone. S2 push button	21
Fig. 19: Putty program main window.	21
Fig. 20: Running Linux.	22
Fig. 21: AM335x Processor's memory map.	23
Fig. 22: Summary of the different configurations for developing applications for embedded systems. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/)	26
Fig. 23: Cross compiling tools installed in the host computer	27
Fig. 24: Selection of the workspace for Eclipse. Use a folder in your account.	27
Fig. 25: Eclipse welcome window.	28
Fig. 26: Eclipse main window.	28
Fig. 27: Basic C project creation in Eclipse	29
Fig. 28: Cross-compiler prefix and path window.	29
Fig. 29: Hello world example.	30
Fig. 30: Tool Chain Editor should be configured to use Cross GCC.	30
Fig. 31: Cross tools locate on (path).The path shown in this figure is an example.Use always the path of your toolchain.	31
Fig. 32: Include search path.	31
Fig. 33: Libraries search path.	32
Fig. 34: Eclipse project compiled (Binaries has been generated).	33
Fig. 35: Run test program in Raspberry PI	34
Fig. 36: Creating a Debug Configuration	34
Fig. 37: Debug configuration including the path to locate the cross gdb tool.	35
Fig. 38: Synaptic program from Dash	37
Fig. 39: Synaptic windows	38

1 SCOPE

1.1 Document Overview

- This document describes the basic steps to develop and embedded Linux-based system using the BeagleBone Black (BBB). The document has been specifically written to use a BBB development system based on the AM335x Texas Instruments Sitara processor. All the software elements used to build the Linux distribution 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

BBB	BeagleBone Black
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] http://elinux.org/Beagleboard:BeagleBone#Rev_A6A User Guide. Reference Manual.

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

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

[RD8] [http://elinux.org/BeagleBoardBeginners#Formatting_the_SD_card_via_fdisk_.22Expert_m
ode.22](http://elinux.org/BeagleBoardBeginners#Formatting_the_SD_card_via_fdisk_.22Expert_mode.22)

3 LAB1: 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 WorkStation player version 12.0 or above. Available at www.vmware.com (free download and use). This software is already installed in the laboratory desktop computer.
2. A VMWare virtual machine with Ubuntu 14.04 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 BBB, 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 Ubuntu Virtual Machine. Wait until the welcome screen is displayed (see Fig. 1 and Fig. 2). Login as “*ubuntuBuildroot*” user using the password “*dte.2016*”. An Ubuntu tutorial is available at Moodle site.

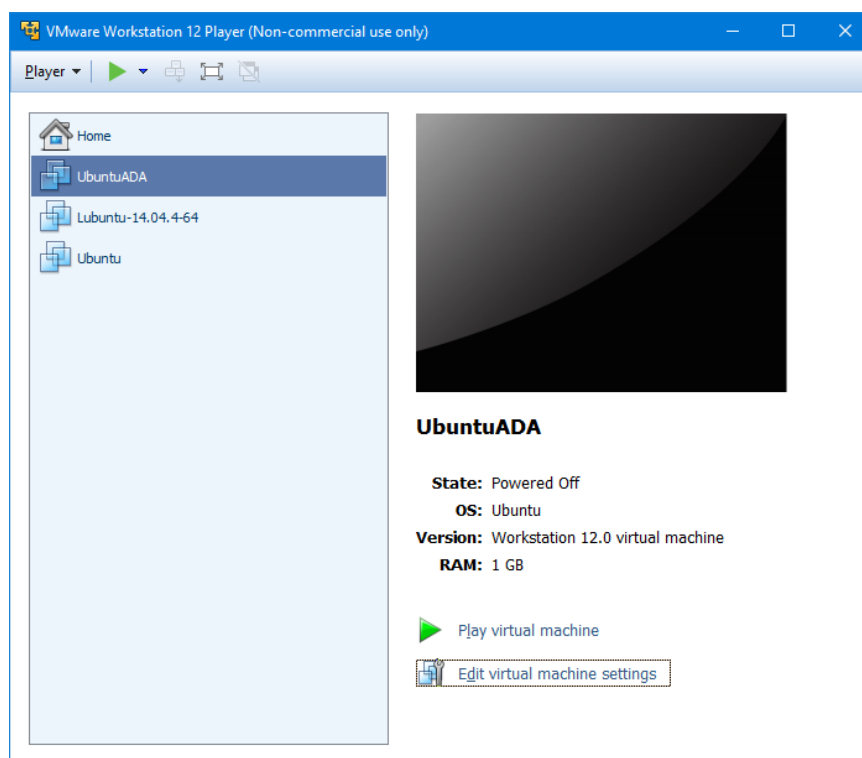


Fig. 1: Main screen of VMware player with some VM available to be executed.

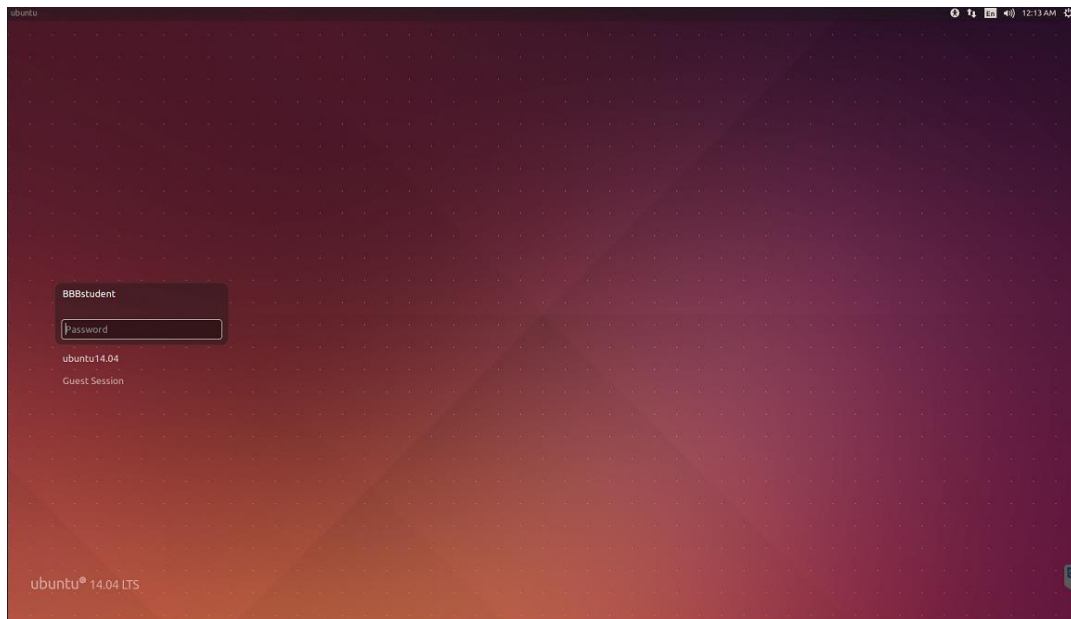


Fig. 2: Ubuntu Virtual Machine login screen.

Open the **Firefox** web browser and download from <https://buildroot.org/> the version identified as buildroot2016-05 (use the download link, see Fig. 3, and navigate searching for a specific release, <https://buildroot.org/downloads/>). Save the file to the **Documents** folder in your account (Fig. 4).

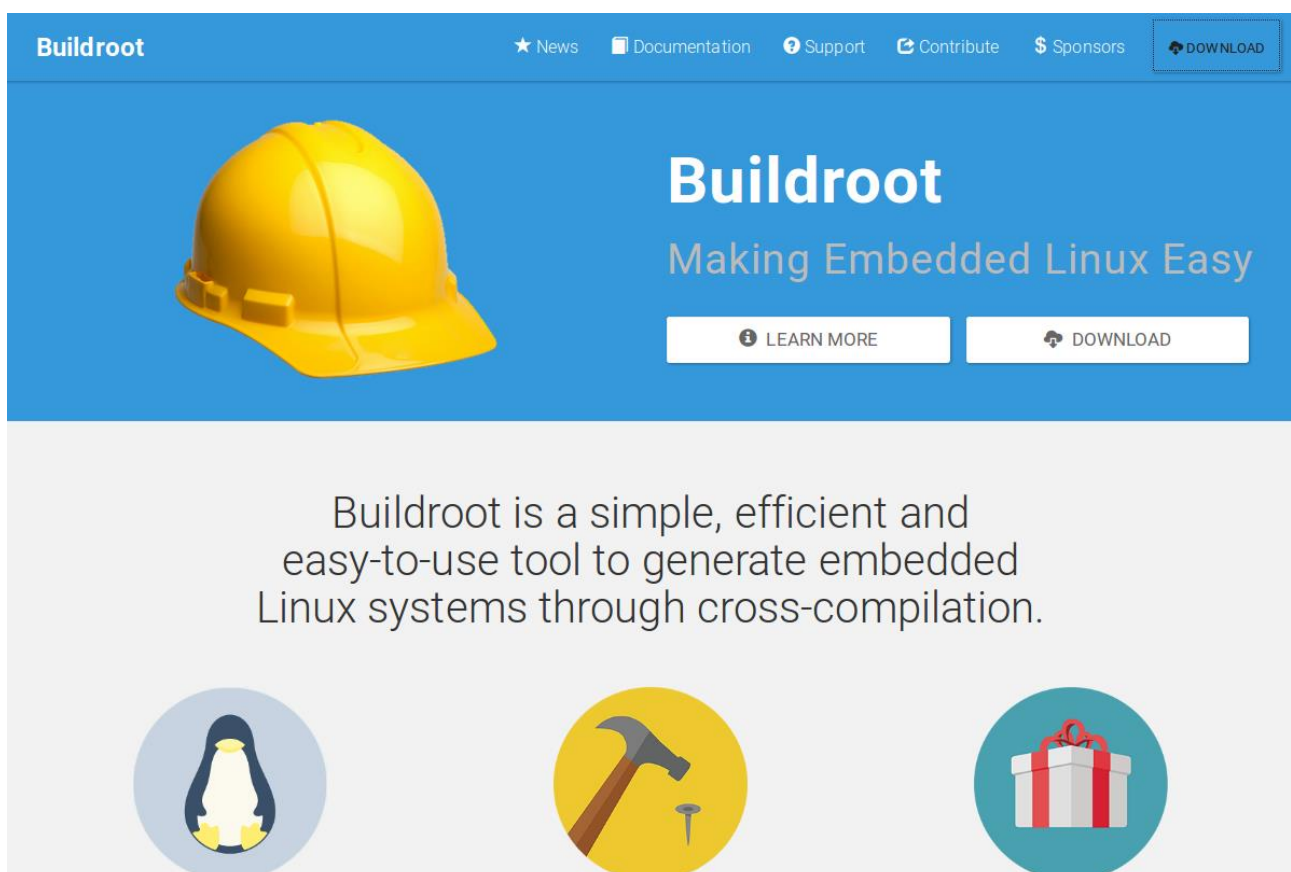


Fig. 3 Buildroot home page.

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

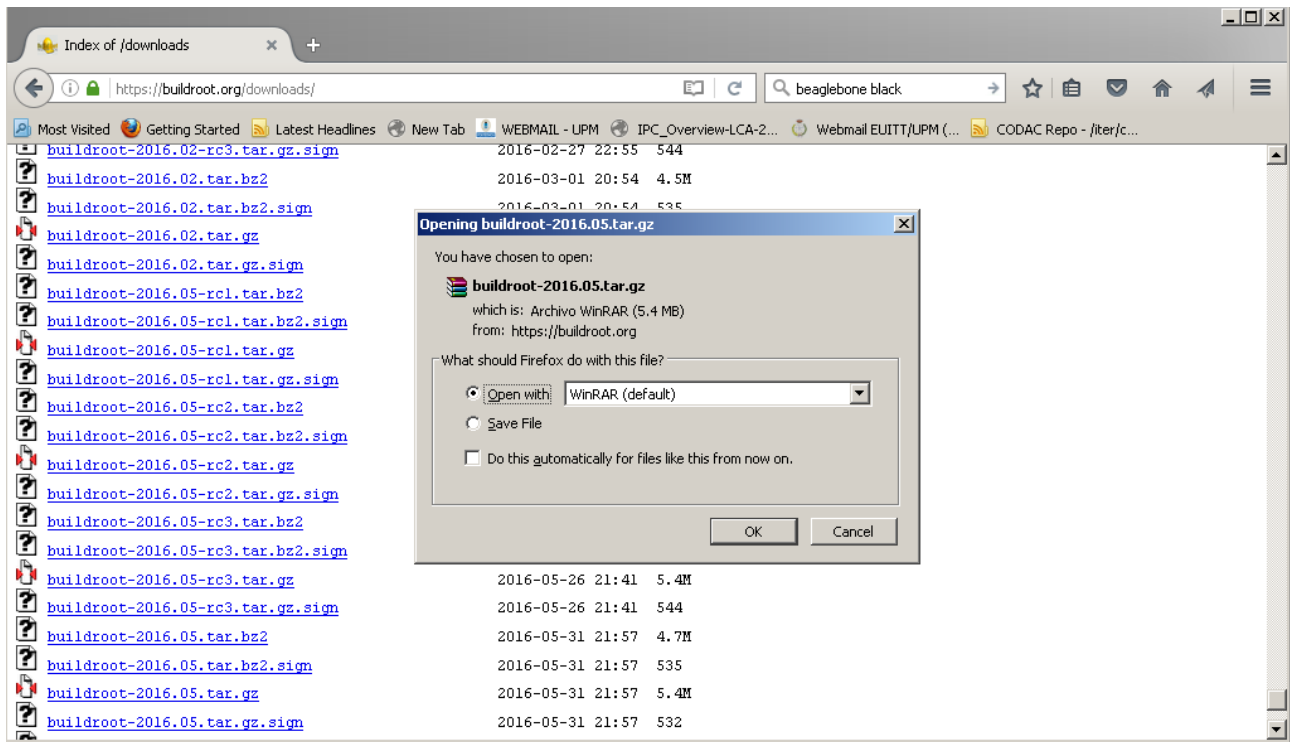


Fig. 4: Downloading buildroot source code.

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

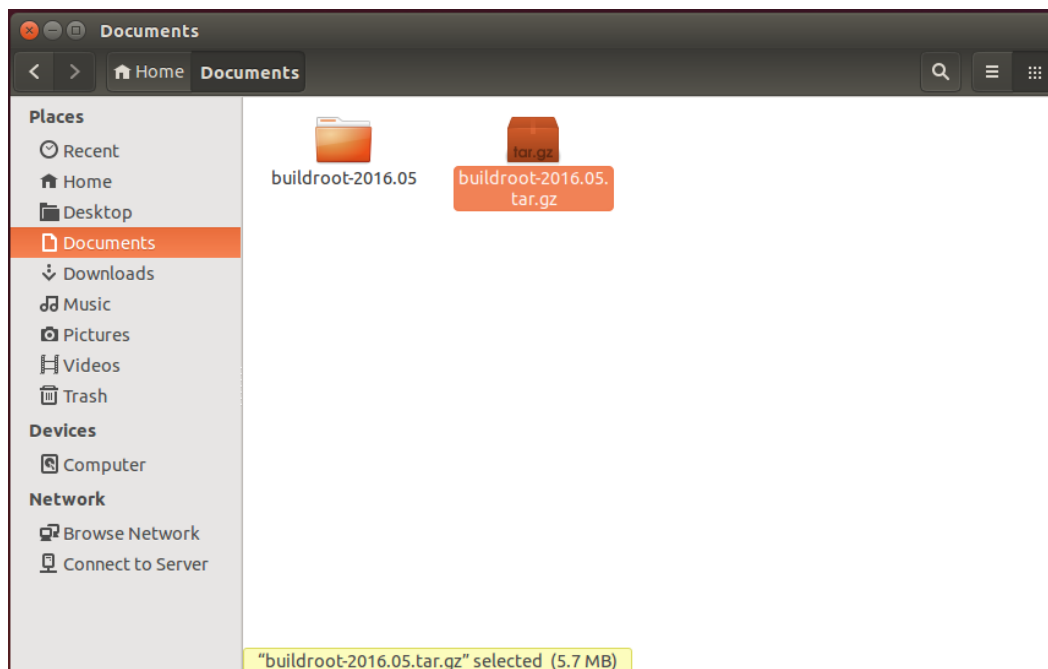


Fig. 5: Buildroot folder (the folder name depends on the version downloaded).

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:

```
xxx@ubuntu:~/Documents$ cd buildroot-2016.05
xxx@ubuntu:~/Documents/buildroot-2016.05$ make xconfig {or make menuconfig}
```



[Help]: You can use either “make xconfig” or “make menuconfig”. If “make xconfig” returns an error try “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 with a content similar to Fig. 7.

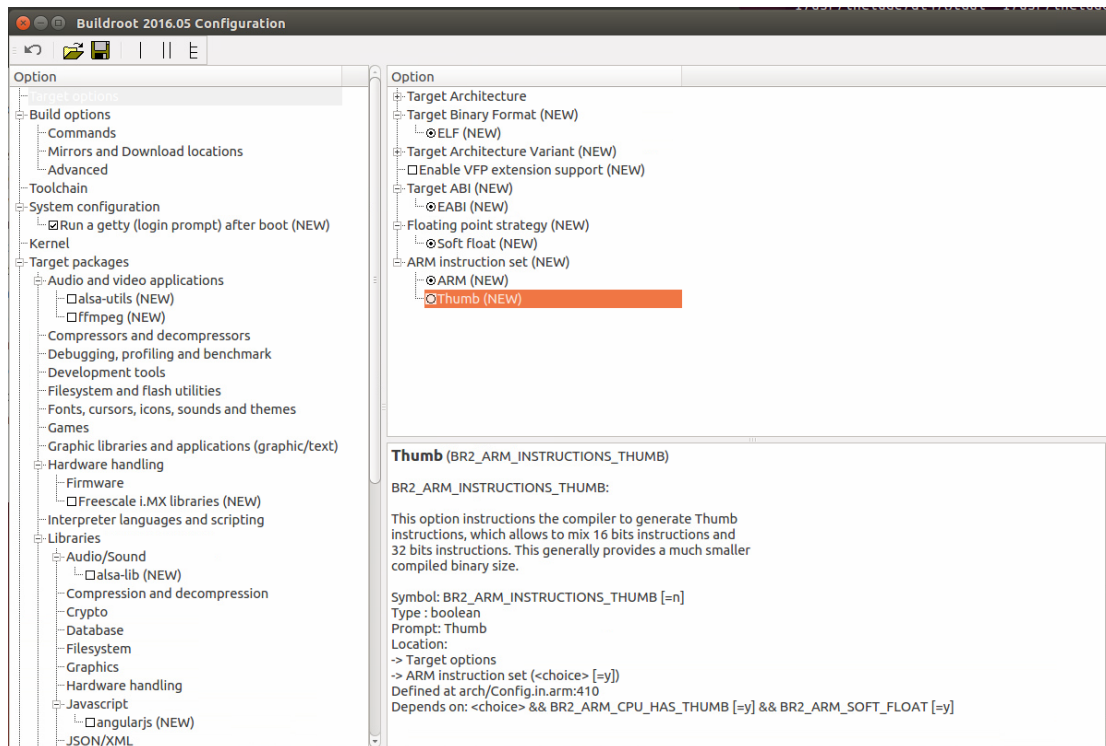


Fig. 7: Buildroot setup screen (make xconfig). The content of this windows depends on the parameter selected.

3.3 Configuring Buildroot.

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 BeagleBone Black. 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 probably 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-A8	
	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	VFPv3-D16	
	ARM instruction Set	ARM	

Main Item	Subitem	Value	Comments
Build options		Default values	How Buildroot will built the code. Leave default values.
Toolchain			Cross Compiler, linker, libraries to be built to compile our embedded application
	Toolchain Type	Buildroot toolchain	Embedded system will be compiled with tools integrated in Buildroot
	Kernel Headers	Same as kernel	Source header files of the Linux Kernel. Navigate to kernel option and select it. The option is now displayed.
	Custom kernel headers series	3.12.x	
	C library	uClibc	
	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 debugable threads,
	Compile and install uClibc utilities	Yes	
	Binutils Version	binutils 2.25.1	Binutils contains tools to manage the binary files obtained in the compilation of the different applications
	GCC compiler Version	gcc 4.9.x	GCC tools version to be installed
	Enable C++ support	Yes	Including support for C++ programming, compiling, and linking.
	Enable compiler tls support	Yes	Enable code generation for Thread Local Storage
	Build cross gdb for the host	GDB Debugger Version, gdb 7.9.x	Includes the support for GDB. GCC debugger.
	Enable MMU support	Yes	Mandatory if building a Linux system
System Configuration			
	System Hostname	beaglebone	Name of the embedded system
	System Banner	Linux BBB Master MISSSI	Banner

Main Item	Subitem	Value	Comments
	Passwords encoding	md5	
	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 FS skeleton	Default target skeleton	Linux folder organization for the embedded system
	Enable root login with password	Root passwd: ada	
	/bin/sh	busybox	
	Remount root filesystem...	yes	
	Network Interface to configure through DHCP	eth0	
	Path to user tables	/home/<user>/ut/test	This file contains the list of user to be created in the embedded Linux system. Use this link to understand how to complete the file https://buildroot.org/downloads/manual/manual.html#makeuser-syntax
	Custom Scripts to run ...	board/beaglebone/post-image.sh	This script is executed after buildroot has finished all the job. Inspect this Shell script to see an example.
	Run a getty: Port to run a getty	/dev/ttyO0	Linux device file with the port to run getty (login) process. Uses ttyO0 for serial port
	Baud rate to use	Keep kernel default	
	TERM environment variable	Vt100	
Linux Kernel			
	Kernel version	Custom Git Repository	
	URL of custom repository	git://git.ti.com/ti-linux-kernel/ti-linux.git	
	Custom repository version	7f280334068b7c875ade51f8f3921ab311f0c824	
	Custom kernel patches	board/beaglebone/patches/linux	
	Kernel configuration	Using a custom (def)config file	
	Configuration file path	board/beaglebone/linux-3.12.config	
	Kernel binary format	zImage	

Main Item	Subitem	Value	Comments
	Kernel compression format	gzip	
	Build Device Tree Blob	Use a device tree present in the kernel	
	Device Tree Source Filenames	am335x-bone am335x-boneblack	Device tree source filenames
	Linux Kernel Extensions	Nothing	
	Linux kernel tools	Nothing	
Target Packages			
	BusyBox configuration file to use	package/busybox/busybox.config	
	Audio and video applications	Default values	
	Compressors 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->am33x-cm3	
	Interpreters language and scripting	Default values	
	Libraries	Default	
	Mail	Default	
	Miscellaneous	Default	
	Networking applications	openssh	
	Package managers Real Time Shell and utilities System Tools Text Editors and viewers	Default	
	Real-Time	Default	
	Security	Default	
	Shell and utilities	Default	
	System tools	Default	

Main Item	Subitem	Value	Comments
	Text editors and viewers	Default	
Filesystem Images			
	ext2/3/4 filesystem root	ext2 (rev0) Compression method no compression	
Bootloaders			
	U-Boot	Build system: Legacy U-Boot board name: am335x_evm U-Boot Version: 2016.03 U-Boot binary format: u-boot.img Install U-Boot SPL binary image name: "MLO"	You need to provide in the "U-boot SPL binary image name" the name of the file to be generated. In this case is "MLO"
Host utilities			
	host u-boot tools ?? host omap-ti-boot-utils??	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.

Empezamos a las 11:40

3.4 Compiling buildroot.

In the Terminal Window executes the following command:

```
xxx@ubuntu:~/Documents/buildroot-2016.05$ 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 **two hour and half**. The download of kernel source code from TI is very slowly.



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.

```
adastudent@ubuntu: ~/Documents/buildroot-2016.05
adastudent@ubuntu: ~/Documents/buildroot-... x adastudent@ubuntu: ~/Documents/buildroot-... x
ot-2016.05/output/build/_fakeroot.fs
chmod a+x /home/adastudent/Documents/buildroot-2016.05/output/build/_fakeroot.fs
PATH="/home/adastudent/Documents/buildroot-2016.05/output/host/bin:/home/adastudent/Documents/buildroot-2016.05/output/host/sbin:/home/adastudent/Documents/buildroot-2016.05/output/host/usr/bin:/home/adastudent/Documents/buildroot-2016.05/output/host/usr/sbin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games" /home/adastudent/Documents/buildroot-2016.05/output/host/usr/bin/fakeroot -- /home/adastudent/Documents/buildroot-2016.05/output/build/_fakeroot.fs
rootdir=/home/adastudent/Documents/buildroot-2016.05/output/target
table='/home/adastudent/Documents/buildroot-2016.05/output/build/_device_table.txt'
tune2fs 1.42.13 (17-May-2015)

mke2img: e2fsck was successfully run on '/home/adastudent/Documents/buildroot-2016.05/output/images/rootfs.ext2' (ext2)

tune2fs 1.42.13 (17-May-2015)
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/adastudent/Documents/buildroot-2016.05/output/target/THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
>>> Executing post-image script board/beaglebone/post-image.sh
adastudent@ubuntu: ~/Documents/buildroot-2016.05$
```

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.

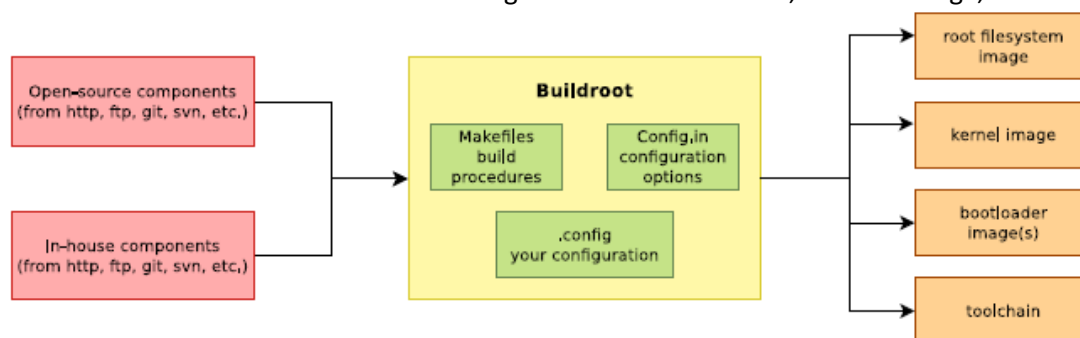


Fig. 9: Schematic representation of the Buildroot tool. Buildroot generates the root file system, the kernel image, the boot loader and the toolchain. Figure copied from “Free Electrons” training materials (<http://free-electrons.com/training/>)

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

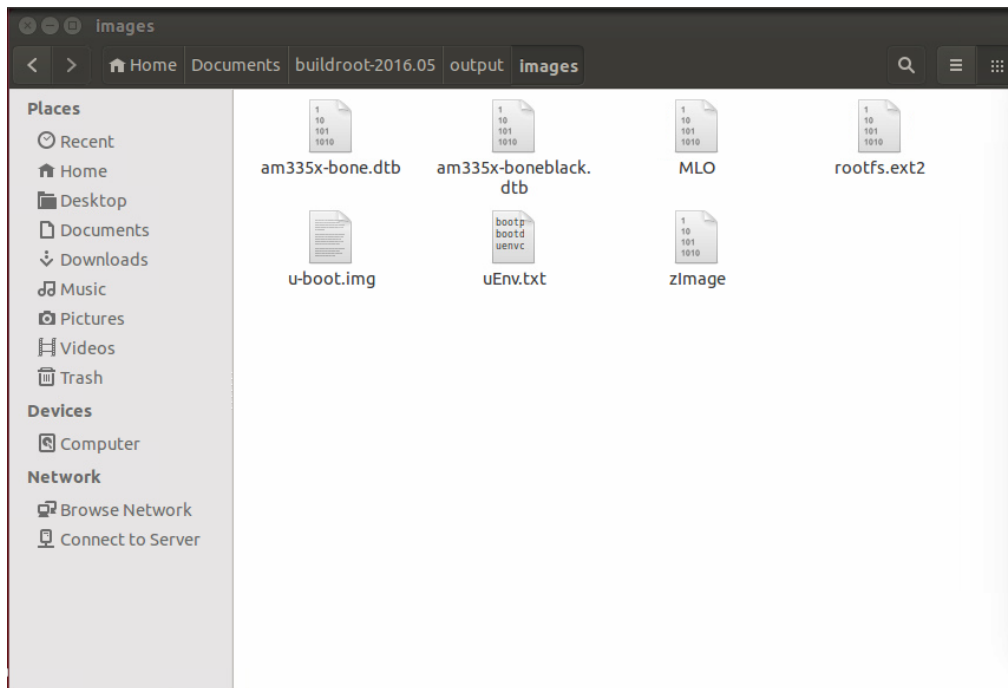



Fig. 10: images folder contains the binary files for our embedded system.

The file MLO is the X-Loader. The u-boot.img file contains the u-boot binary for booting your Linux. Linux image is zImage. The files with dtb extension are the device tree in binary form used by Linux in the booting phase to discover the hardware available in the BeagleBone. The file rootfs.ext2 contains all the files of the file system of your embedded Linux. In order to boot the BBB from a SD card firstly you need to format it adequately, follow the instructions included in [RD8]. All files (Fig. 10) with the exception of the rootfs.ext2 must be copied to the SD card FAT32 partition for booting BBB. It is mandatory to copy first the MLO file. Copy the content of rootfs.ext2 to Linux partition using the dd command as follow:

```
xxx@ubuntu:~/Documents$ cd buildroot-2016.05/output/images
xxx@ubuntu:~/Documents/output/images $ sudo dd if=rootfs.ext2 of=/dev/sd<n>2
```

Please use “dmesg” Linux command to find out the device assigned to your SD card. Use “eject” of Ubuntu () to unmount the SD Card, this is similar to eject an USB disk in Windows. Insert again the SD card in the computer and verify that the files are correctly copied.

3.6 Configuring the Linux kernel parameters

In all the Linux embedded applications it is necessary to set up the kernel in order to support the different physical devices and user space applications. All kernel sources provide a basic configuration for specific hardware platforms. If you are using a commercial hardware platform probably you will find a kernel configuration suitable for you, if not you will need to define it. This is a hard task but you will find hundreds of examples in the Internet. Linux kernel has a directory with predefined hardware configurations. The relative path is this: **“./<your buildroot installation path>/output/build/linux-<xxxxxxxx>/arch/arm/configs”**. Have a look to this folder and you will find plenty of text files with a defconfig suffix containing the kernel parameter configuration of a specific hardware.

If you want to modify the kernel configuration from a Buidlroot folder execute this command (Fig. 11):

```
rpi@ubuntu:~/Documents/buildroot-2016.05$ make linux-menuconfig
```

You can navigate in this application using the arrows. Inspect the configuration of the kernel. If you add or remove features, save them and compile again buildroot using make command. **This will generate new images that must be copied to SD card.**

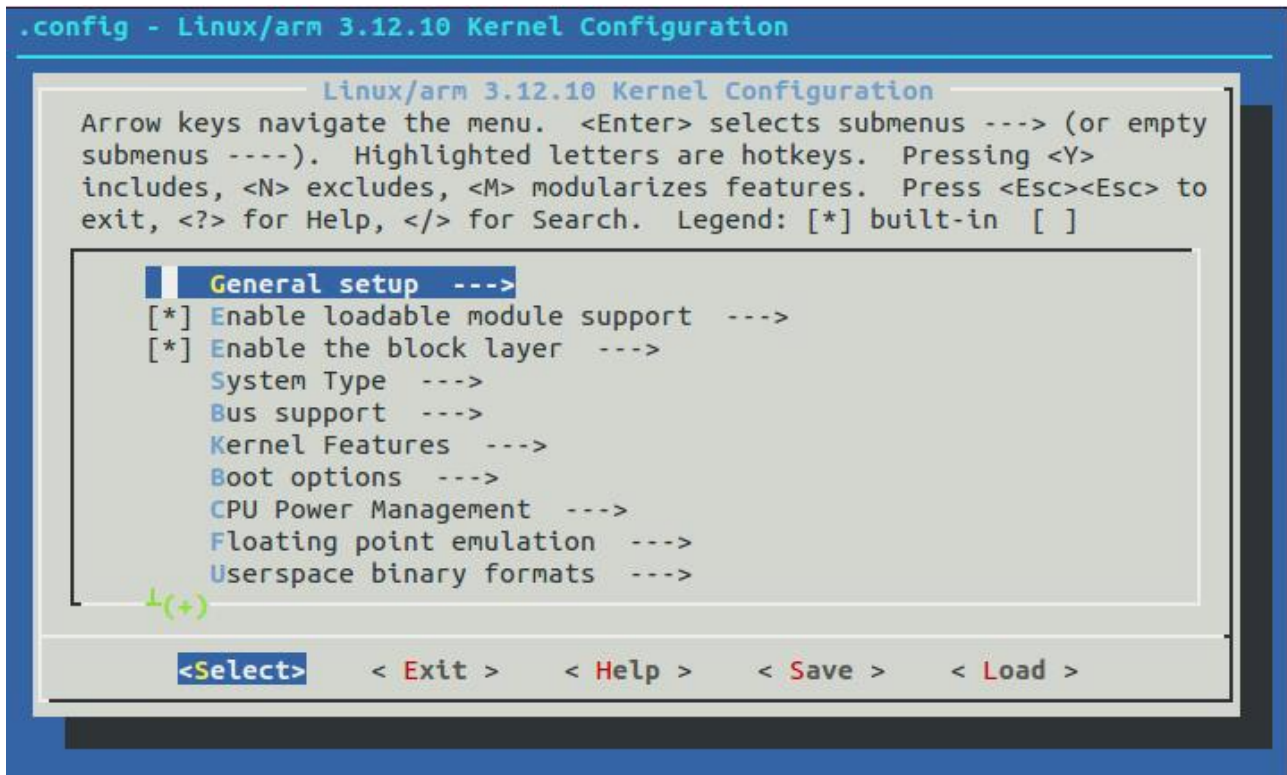


Fig. 11: Main window for configuring the Linux kernel.



[Info]: We do not need to apply changes in the default kernel configuration for BBB. This step has been included here in order to understand how to proceed whether you need to tune your kernel

3.7 Booting the BeagleBone.

Fig. 12 displays a BeagleBone Black. The description of this card, their functionalities, interfaces and connectors are explained in the ref [RD5]. The basic connection requires:

- To connect the USB to RS232 adapter (provided) to the BBB serial line header (see Fig. 15 and Fig. 16). This adapter will be provide the serial line interface to be used as console in the Linux operating system.



[Connecting the USB serial interface in the VM]: In order to connect the serial interface in your virtual machine you need to connect it selecting un VMWare Player the option Player->Removable Devices->Future Devices TTL 232R 3V3->Connect

- To connect the power supply provided to the jack identified as 5v or using a USB cable.

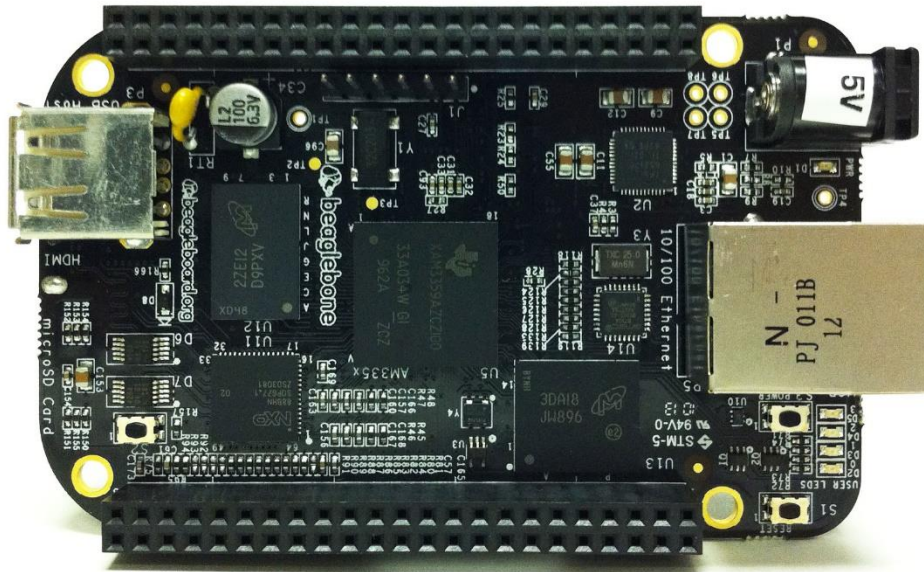


Fig. 12: BBB.

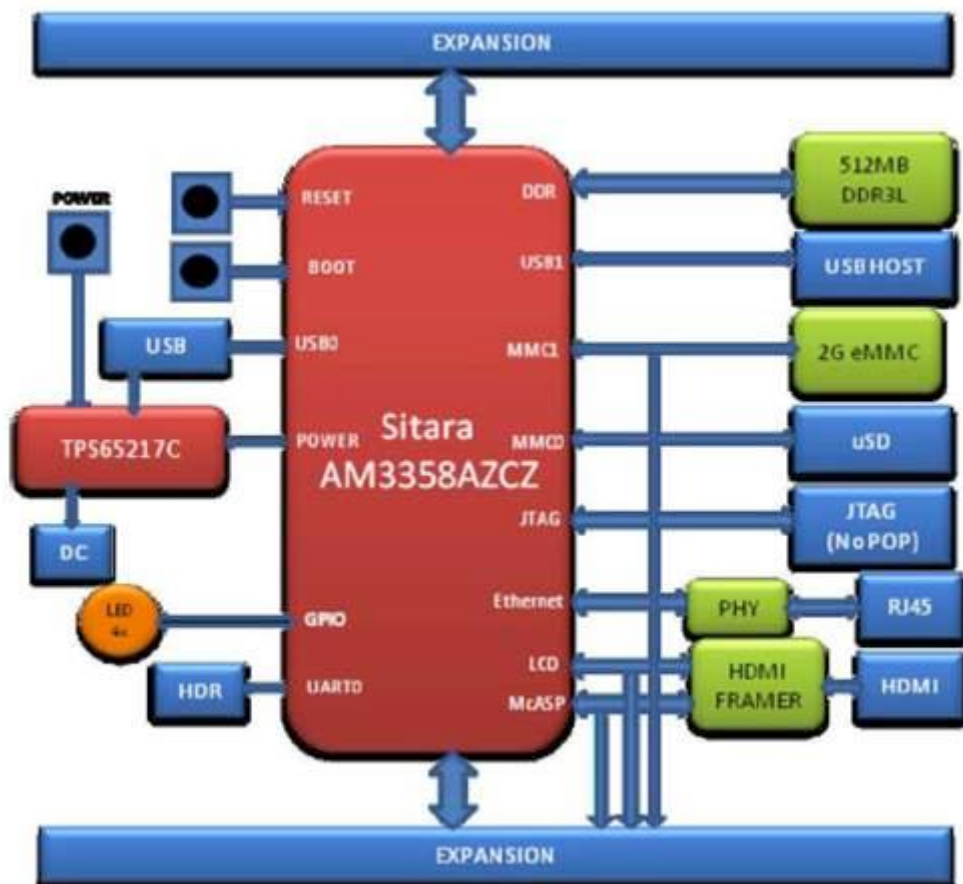


Fig. 13: BeagleBone Black block diagram

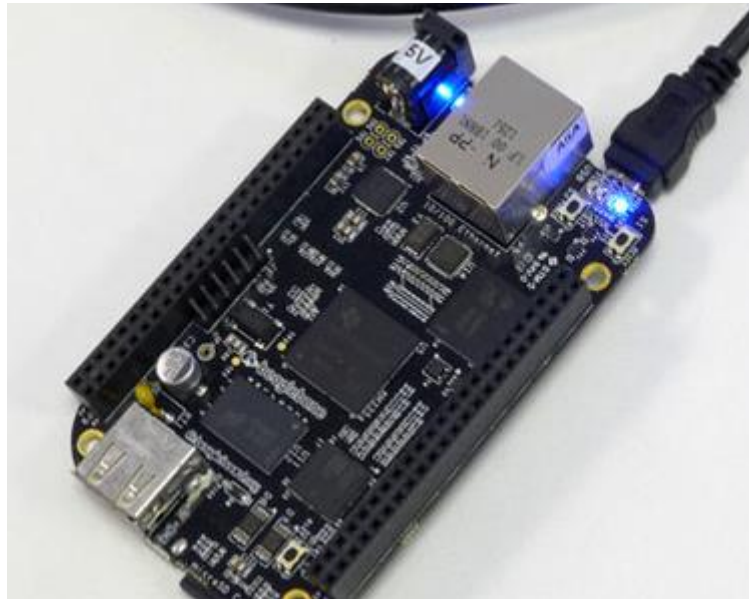


Fig. 14: Powering BBB with USB

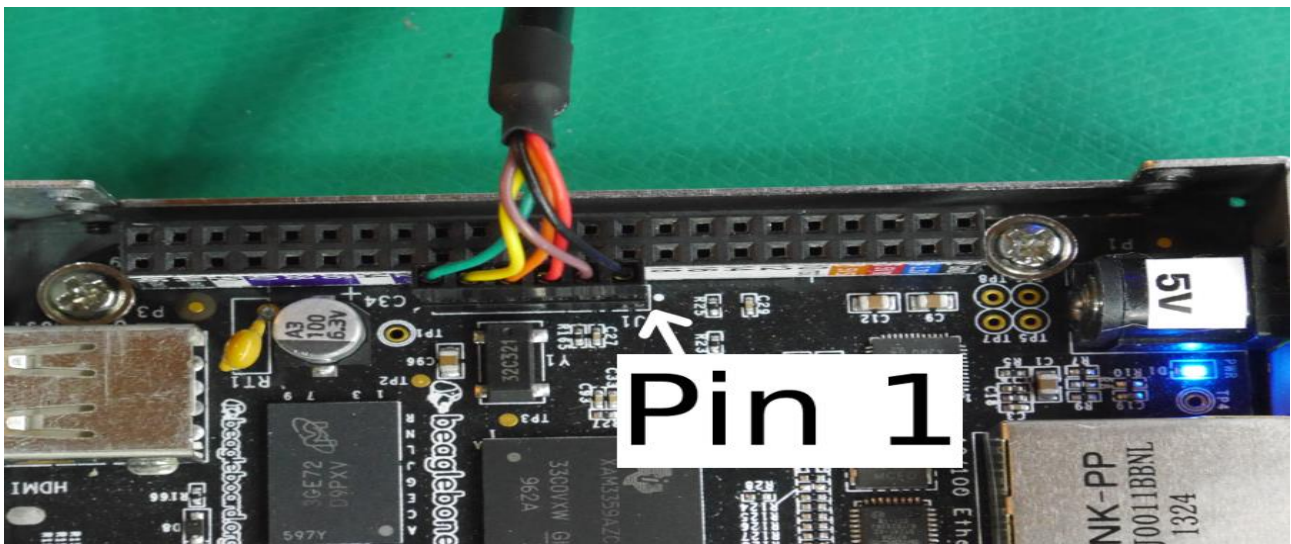


Fig. 15: BBB serial line header terminal identification.

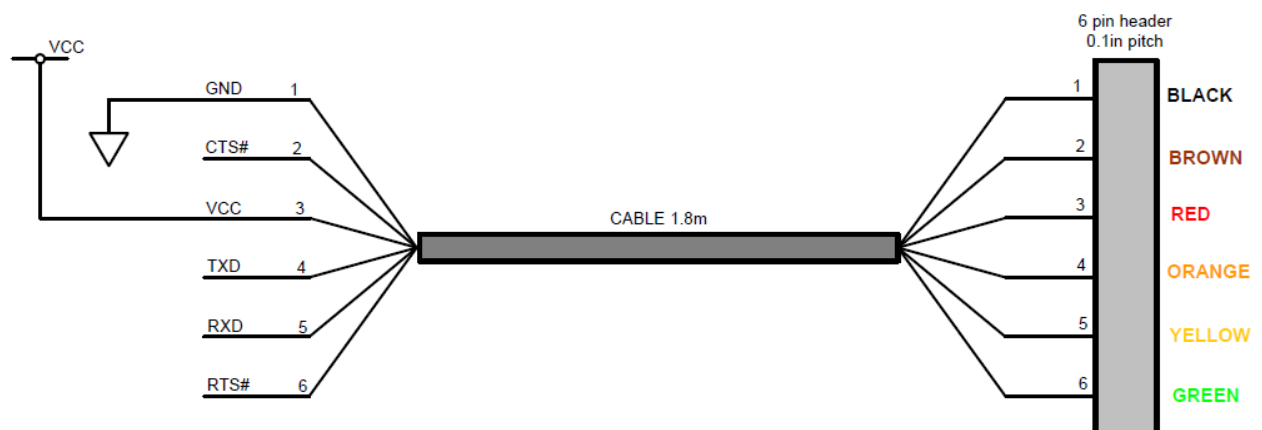


Fig. 16: Identification of the terminals in the USB-RS232 adapter

The booting process of the BBB with its AM335x processor is depicted in Fig. 17. By default, the ROM in the Sitara AM335x will boot from the MMC1 interface first (the onboard eMMC), followed by MMC0 (MicroSD), UART0 and USB0. If the boot switch (S2) is held down during power-up, the processor starts running the ROM code and boots from the SPI0 Interface first, followed by MMC0, USB0 and UART0. This allows the BeagleBone Black to bypass the on-board eMMC and boot from the removable uSD (provided no valid boot device is found on SPI0).

Open the putty application in the Linux host (execute “sudo putty” in a Linux terminal) and open a session to the serial line (use /dev/ttyUSB0 with 115200 baud rate). Apply the power supply pressing S2 button (Fig. 18) and releasing it after (for instance) one second, the BeagleBone will boot from SD card.

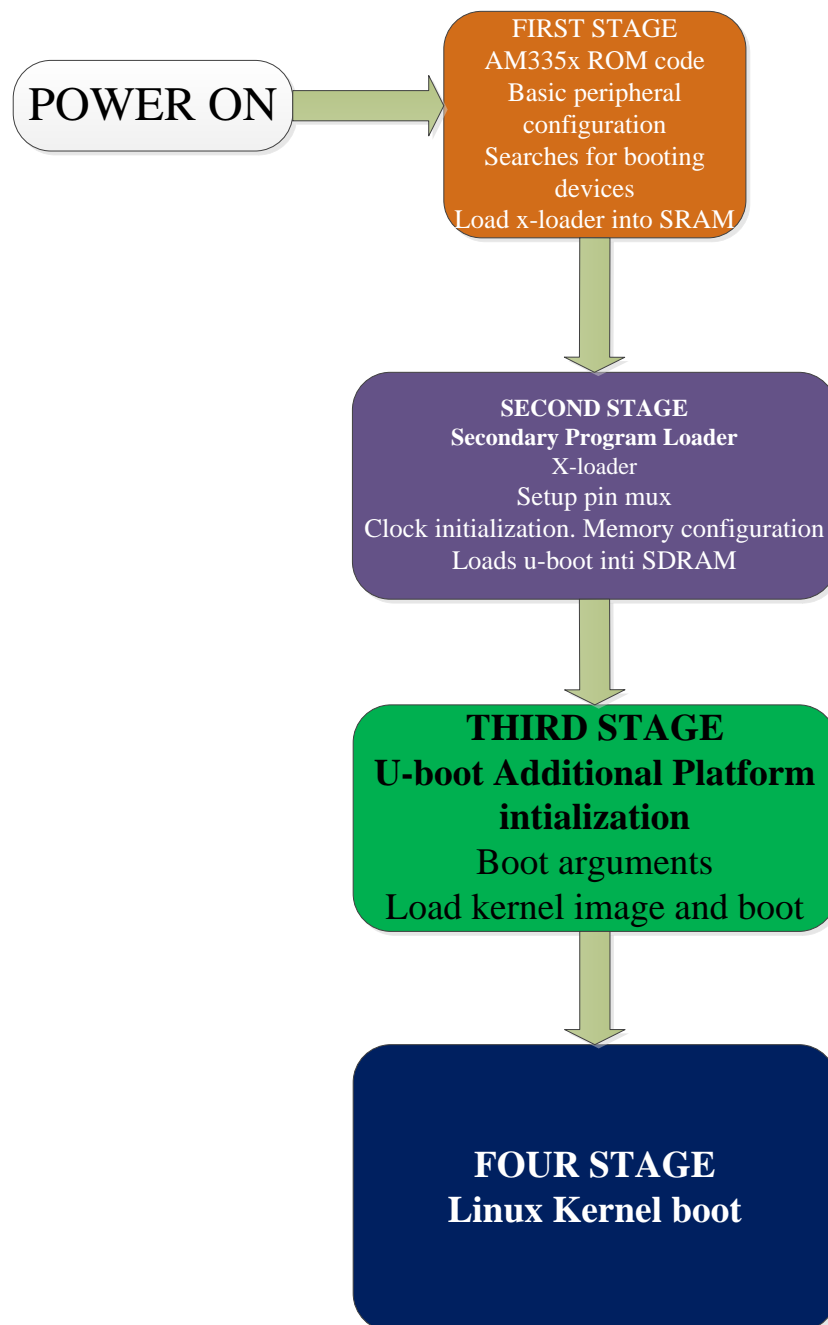


Fig. 17: Booting process for BBB.

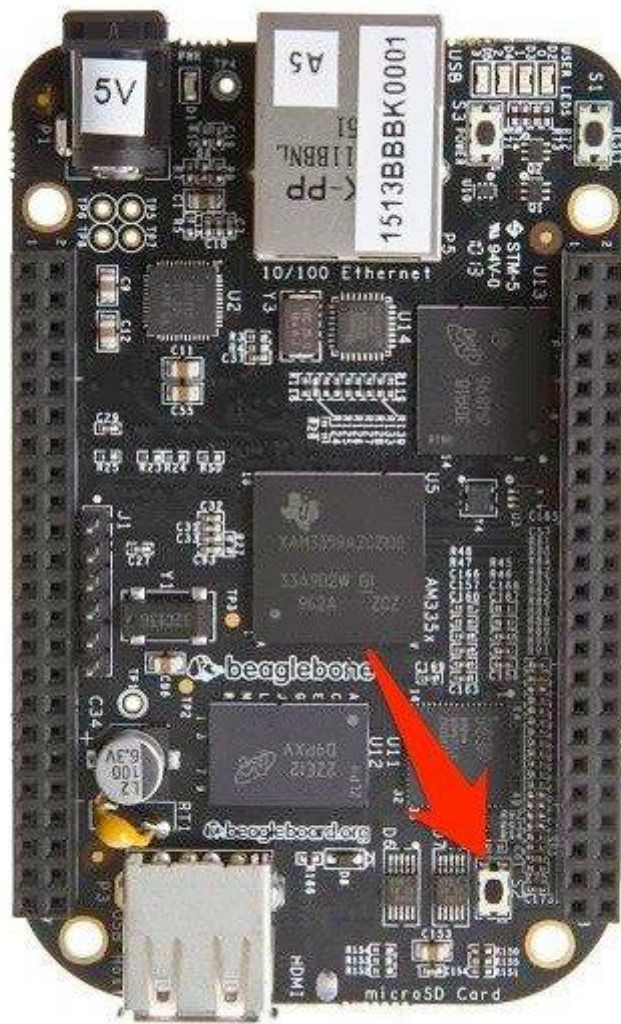


Fig. 18: BeagleBone. S2 push button

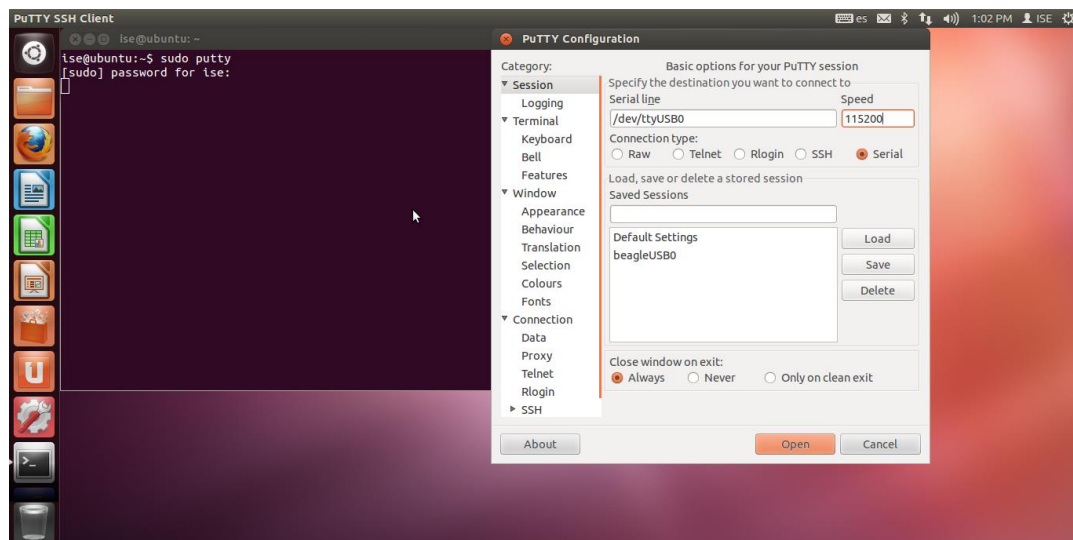
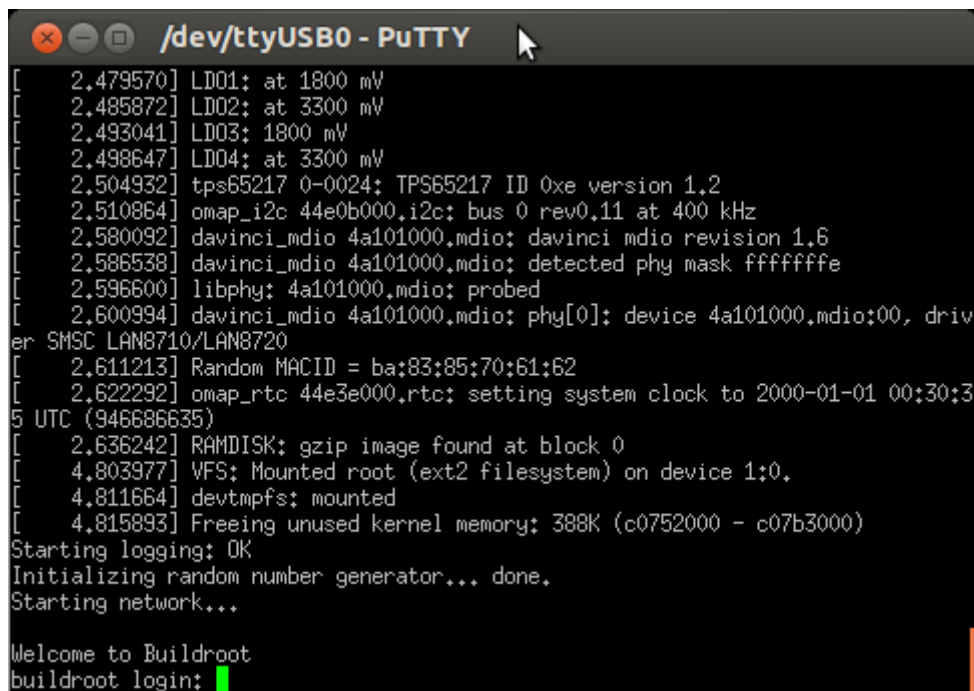


Fig. 19: Putty program main window.



[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.

After some seconds you will see a lot messages displaying in the terminal. Linux kernel is booting and the operating system is running their configuration and initial daemons. If the system boots correctly you will see an output like the represented in Fig. 20. Introduce the user name root and the Linux shell will be available for you.



```
/dev/ttyUSB0 - PuTTY
[ 2.479570] LD01: at 1800 mV
[ 2.485872] LD02: at 3300 mV
[ 2.493041] LD03: 1800 mV
[ 2.498647] LD04: at 3300 mV
[ 2.504932] tps65217 0-0024: TPS65217 ID 0xe version 1,2
[ 2.510864] omap_i2c 44e0b000.i2c: bus 0 rev0.11 at 400 kHz
[ 2.580092] davinci_mdio 4a101000.mdio: davinci mdio revision 1,6
[ 2.588538] davinci_mdio 4a101000.mdio: detected phy mask ffffffff
[ 2.596600] libphy: 4a101000.mdio: probed
[ 2.600994] davinci_mdio 4a101000.mdio: phy[0]: device 4a101000.mdio:00, driv
er SHSC LAN8710/LAN8720
[ 2.611213] Random MACID = ba:83:85:70:61:62
[ 2.622292] omap_rtc 44e3e000.rtc: setting system clock to 2000-01-01 00:30:3
5 UTC (946686635)
[ 2.636242] RAMDISK: gzip image found at block 0
[ 4.803977] VFS: Mounted root (ext2 filesystem) on device 1:0.
[ 4.811664] devtmpfs: mounted
[ 4.815893] Freeing unused kernel memory: 388K (c0752000 - c07b3000)
Starting logging: OK
Initializing random number generator... done.
Starting network...

Welcome to Buildroot
buildroot login: 
```

Fig. 20: Running Linux.

3.8 Basic test your embedded Linux System

Your embedded Linux system is running in the Beaglebone, execute the following commands and analyse the output information.

```
$dmesg
...
$uname -r
...
$ifconfig
...
$ps -ax
```

Answer the following questions:

- Which are the kernel parameters used to boot the system? Explain the meaning of them.
- What IP address has been assigned to BBB? Who is providing this IP?
- How many processes are running in the Linux OS?

- d) Which devices have been detected in the system boot?
- e) Try to access to BBB using putty with a ssh connection. First try to access with root account and later with another account. What happens? Find a possible solution to this.

3.9 Understanding the boot process with u-boot in the BeagleBone

The Texas Instruments am335x processor (based on an ARM cortex-A8) has a memory area for SDRAM available in the range 0x80000000-0xC0000000. In this memory area we are going to load using u-boot two basic elements: the kernel and the device-tree. The starting address for the kernel is 0x80200000 and for the device tree 0x80F80000.

Table 2-1. L3 Memory Map (continued)

Block Name	Start_address (hex)	End_address (hex)	Size	Description
SGX530	0x5600_0000	0x56FF_FFFF	16MB	SGX530 Slave Port
Reserved	0x5700_0000	0x57FF_FFFF	16MB	Reserved
Reserved	0x5800_0000	0x58FF_FFFF	16MB	Reserved
Reserved	0x5900_0000	0x59FF_FFFF	16MB	Reserved
Reserved	0x5A00_0000	0x5AFF_FFFF	16MB	Reserved
Reserved	0x5B00_0000	0x5BFF_FFFF	16MB	Reserved
Reserved	0x5C00_0000	0x5DFF_FFFF	32MB	Reserved
Reserved	0x5E00_0000	0x5FFF_FFFF	32MB	Reserved
Reserved	0x6000_0000	0x7FFF_FFFF	512MB	Reserved
EMIF0 SDRAM	0x8000_0000	0xBFFF_FFFF	1GB	8-/16-bit External Memory (Ex/R/W) ⁽³⁾
Reserved	0xC000_0000	0xFFFF_FFFF	1GB	Reserved

Fig. 21: AM335x Processor's memory map.

When we perform the reset in the BBB pressing the S2 button the first stage bootloader, boot ROM inside the AM335x) will search for the MLO (secondary bootloader) and u-boot (tertiary) loaders in the SD card. The u-boot binary performs the following actions:

- a) Once U-boot performs the basic initialization it is waiting for a user keystroke a specific time in seconds. This time is controlled with a u-boot environment variable named bootdelay (typically 3 seconds).
- b) If this timeout is reached U-boot firstly searches for a binary script named "boot.scr.uimg". If this scripts ends without executing the Linux kernel, u-boot will search a "uEnv.txt" file, with user defined variables, in the SD card. If these file doesn't run the Linux kernel, u-boot will execute the content of "bootcmd" environment variable. Typically embedded Linux systems uses the "boot.scr.uimg" script to boot using a sequence defined by the developer. This file has a binary content and is obtained using a text file (for instance called boot.scr) with the mkimage tool. This tool is part of u-boot software.



[Comment] uEnv.txt allows pre-setting of the U-Boot environment variable values, prior to running bootcmd. boot.scr allows running of a U-Boot script file, prior to running bootcmd.



[Info] u-boot source code can be found in different internet repositories. In this tutorial we are using the official repository of DENX Software Engineering ([git://git.denx.de/u-boot.git](https://git.denx.de/u-boot.git)). You can also find u-boot in other internet sites. In particular Texas Instruments, the manufacture of the AM335x processor provides

a site with the u-boot code for the TI processors. This link is “git://git.ti.com/ti-u-boot/ti-u-boot.git”. For sure that if you compare the code you will see small differences between them. Be aware of this.

The embedded Linux system developed in this tutorial uses the content of “uEnv.txt” file in the boot process. This file has been copied to FAT32 partition in the SD card. Remove it and power on the BBB pressing the S2 button and releasing it after some seconds. During u-boot boot process you will see a message inviting you to press a key to interrupt the u-boot sequence. Press a key to get the u-boot prompt and execute the following commands (Explain the meaning of the different u-boot commands). The bootz command will launch the execution of the zImage loaded in the BBB DRAM memory.

```
mmc rescan
setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext2 rootwait
fatload mmc 0 0x80F80000 am335x-boneblack.dtb
fatload mmc 0 0x80007fc0 zImage
bootz 0x80007fc0 - 0x80F80000
```

Of course booting in this way, manually, is not very interesting for functional embedded system. In the next point we are going to develop a script automating this boot process.

3.10 Booting the BeagleBone using a script.

The previous aforementioned step for starting up the Linux can be simplified a little bit using a u-boot script. If you create a text file with the u-boot commands listed previously you obtain a binary script using the following command in an Ubuntu Linux command terminal:

```
mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "BBB" -d boot_mmc.txt boot.scr.uimg
```

Copy the “boot.scr.uimg” file into the SD card and reboot the BBB. Now the BBB is booting automatically without the need of user intervention.

3.11 Additional details about the configuration of BBB in u-boot.

The configuration of the BBB in u-boot is available in the following files listed in Table 2.

Table 2: Main files in u-boot with the configuration for BBB hardware based on Am335x processor.

File	Objective
<u-boot>/include/configs/am335x_evm.h	Definition of the different constants and environment variables of u-boot for AM335x-based hardware. In this file the default addresses to store kernel, device tree and filesystem are defined.
<u-boot>/board/ti/am335x	Folder with source files containing the code to initialize and manage the AM335x processor and the board resources.
<uboot>/configs/am335x_evm_defconfig	Default u-boot configuration for BBB

3.12 Configuring the network interface in the BBB.

The configuration generated by buildroot for the BBB's network interface (eth0) is DHCP. This means that in Linux initialization the "init" process is launching an application working as DHCP client. Execute the commands `ifconfig` and `ps` to see the network configuration and the processes running in the BBB. You can test if the network is working using the ping command. Use the ping command with a known address. For instance, the address of one computer in the lab.



[Help]: If you execute the ping command in the BBB trying to connect with a computer in the laboratory probably you obtain a connection timeout. Consider that computers 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 BBB doesn't have a firewall running and the connection should be ok.



[Question] What is the MAC address of your BBB?



[DHCP Server]: The DHCP server providing the IP address to the BBB should be active in your network. In the UPM ETSIST labs the IP is assigned using the BBB'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.

4 LAB2: 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 application 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. 22).

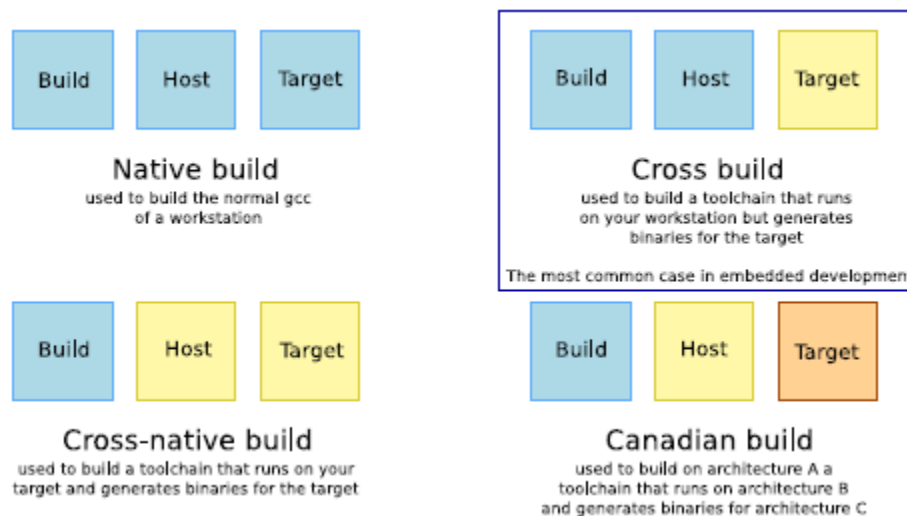


Fig. 22: Summary of the different configurations for developing applications for embedded systems. Figure copied from “Free Electrons” training materials (<http://free-electrons.com/training/>)

The first question is where the cross-compiler is located. The answer is this: in the folder “buildroot-2016.05/output/host/usr/bin”. If you inspect the content of this folder you can see the entire compiling, linking and debugging tool (see Fig. 23). These programs are executed in your x86 computer but they generate code for ARM processor.

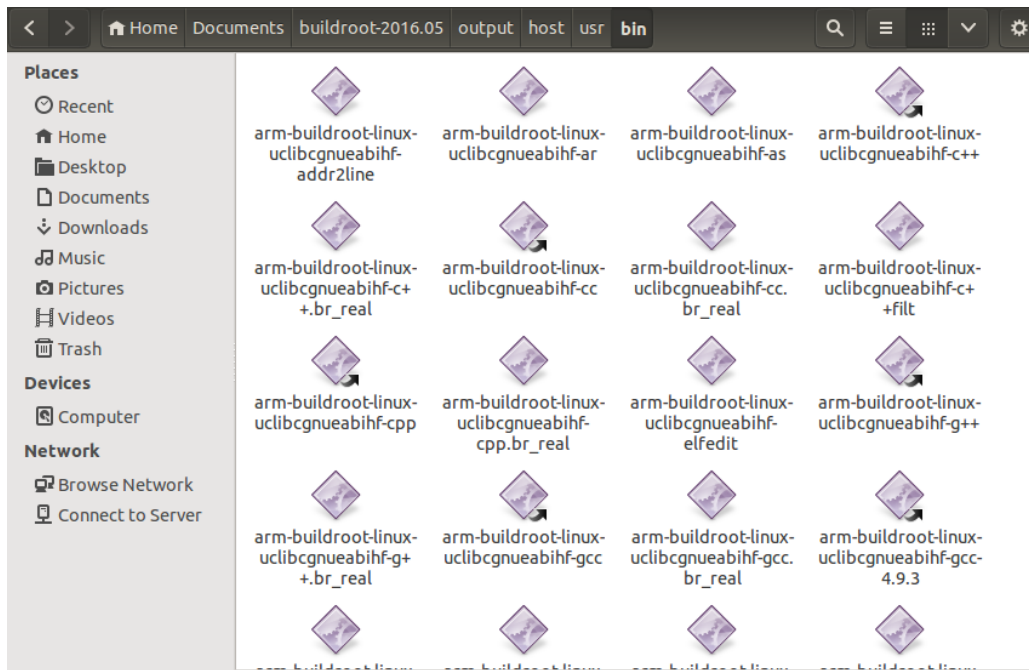


Fig. 23: Cross compiling tools installed in the host computer

In a Terminal window start Eclipse with the following command:

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

The popup window invites you to enter the workspace (see Fig. 24). 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.

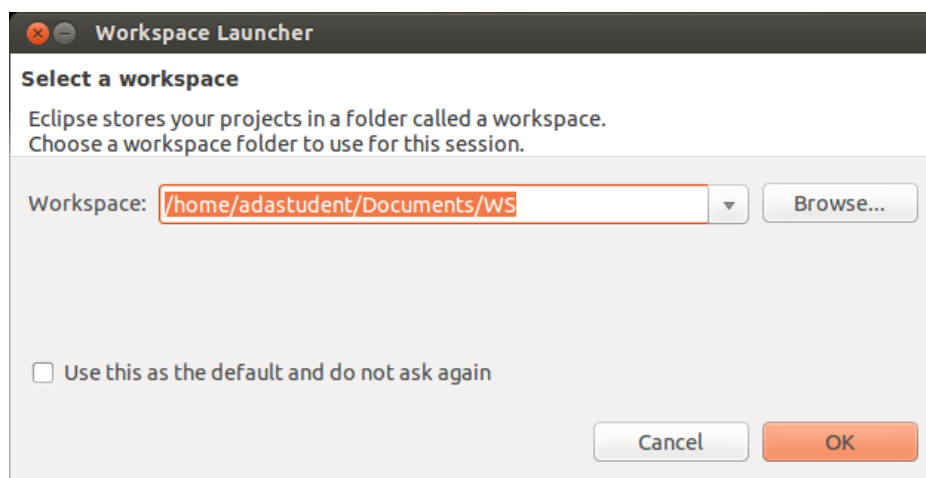


Fig. 24: Selection of the workspace for Eclipse. Use a folder in your account.

Select Ok and the welcome window of Eclipse will be shown (Fig. 25). Next, close the welcome window and the main eclipse window will be displayed (Fig. 26).

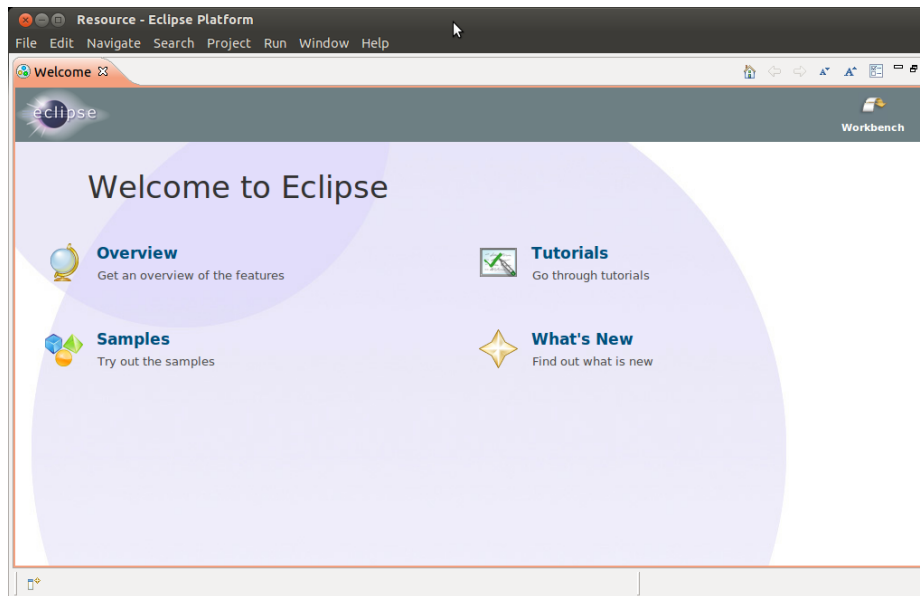


Fig. 25: Eclipse welcome window.

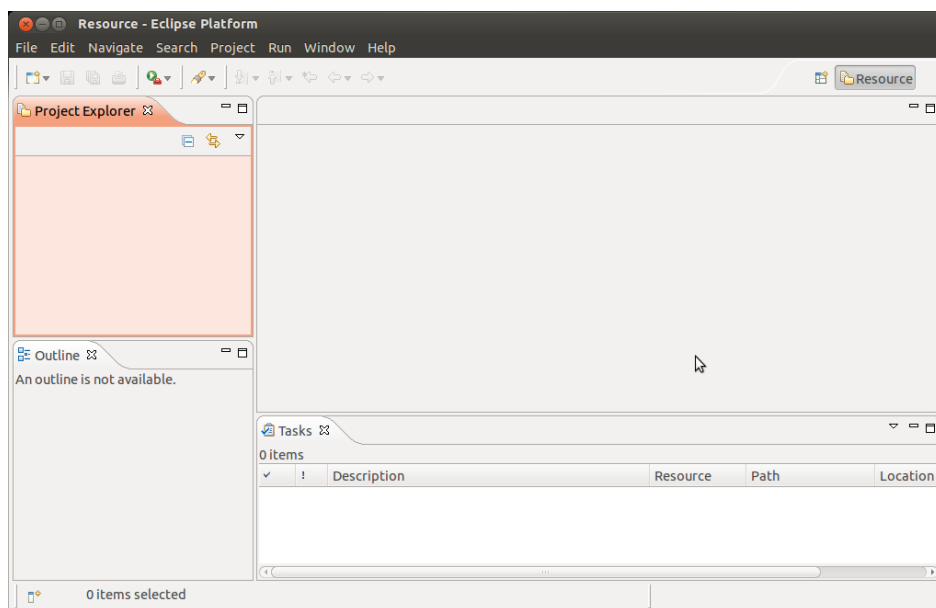


Fig. 26: Eclipse main window.

Create an Eclipse C/C++ project (File->New->Project->C/C++project->C project) selecting the hello world example (see Fig. 27). Specify the project name and the toolchain to be used. In this case a Cross GCC. Press Next.

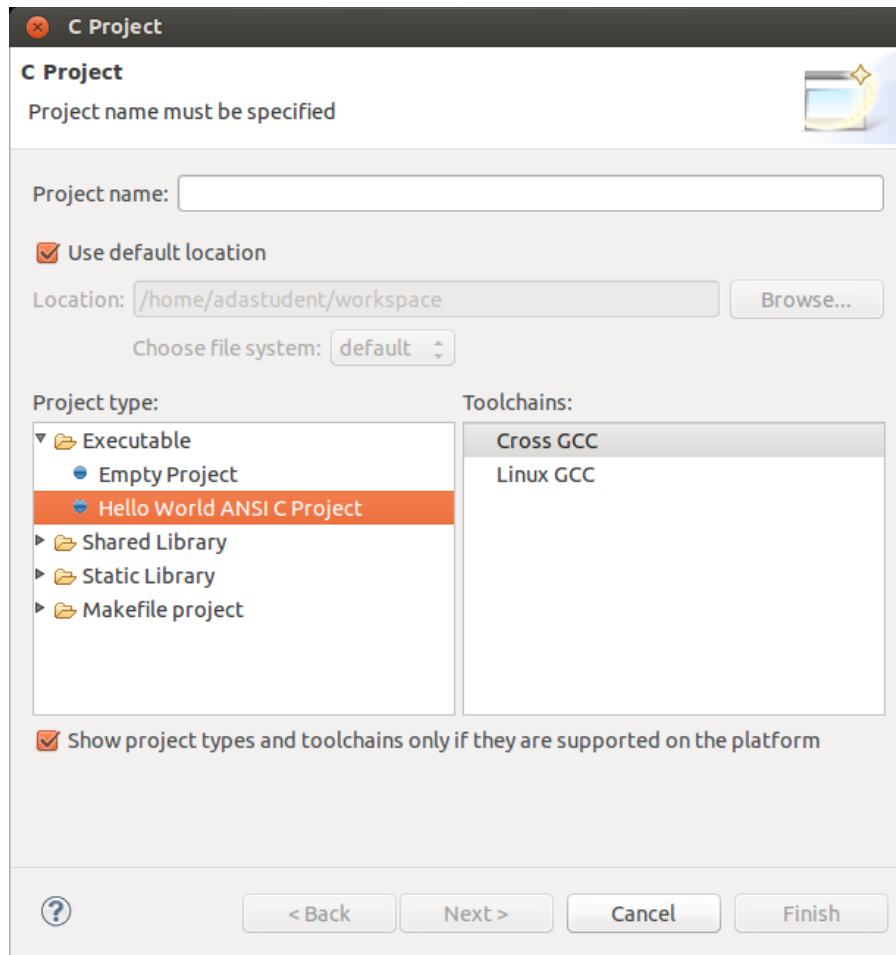


Fig. 27: Basic C project creation in Eclipse

There is a window (Fig. 28) 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.

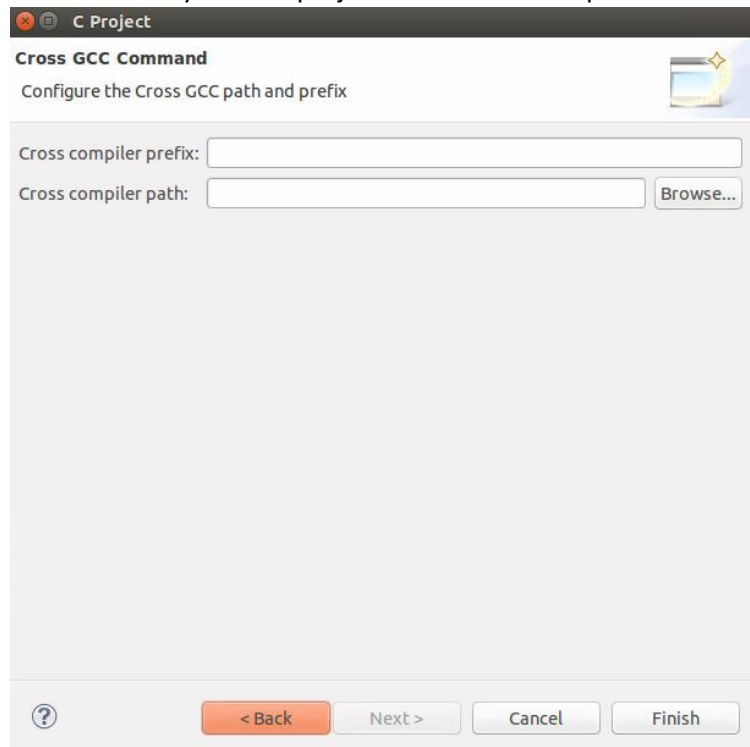


Fig. 28: Cross-compiler prefix and path window.

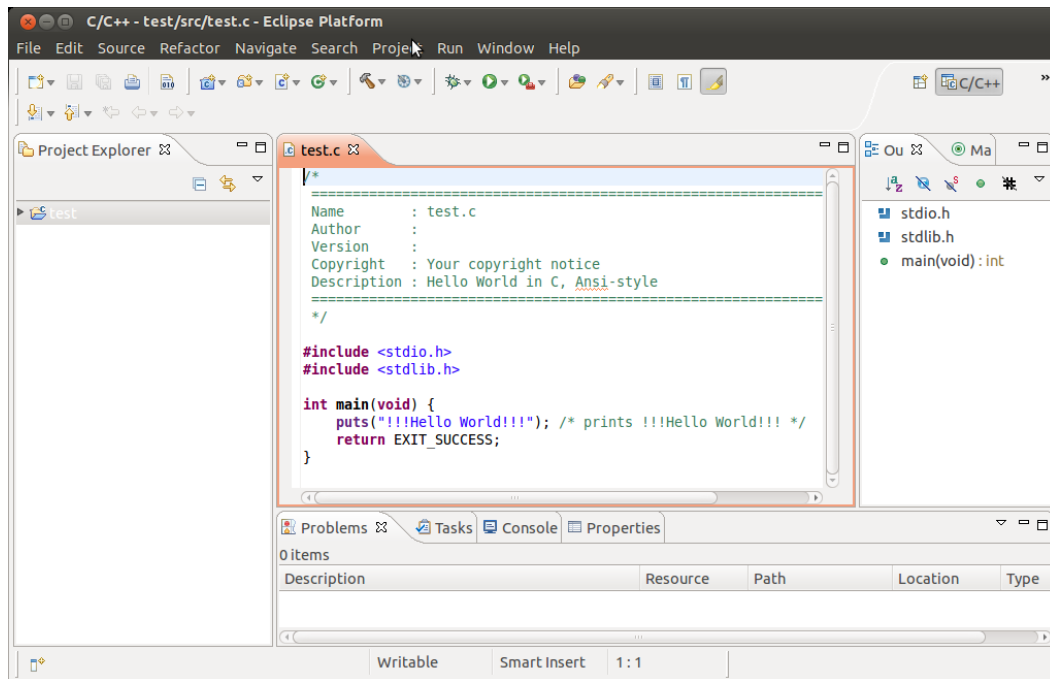


Fig. 29: Hello world example.

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. 30 and Fig. 31 shown. Pay attention that Prefix requires a string ending in a hyphen.

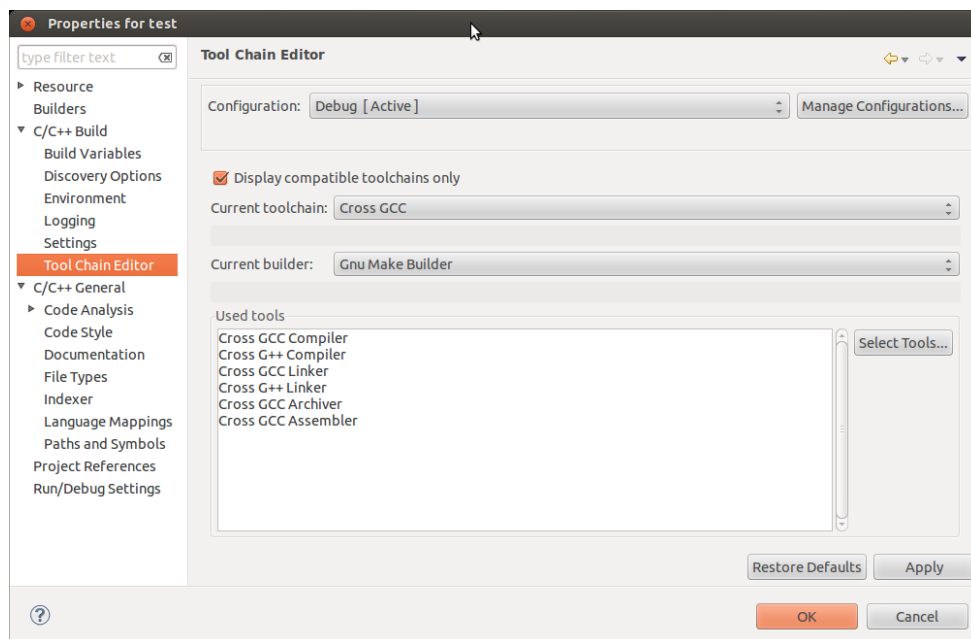


Fig. 30: Tool Chain Editor should be configured to use Cross GCC.

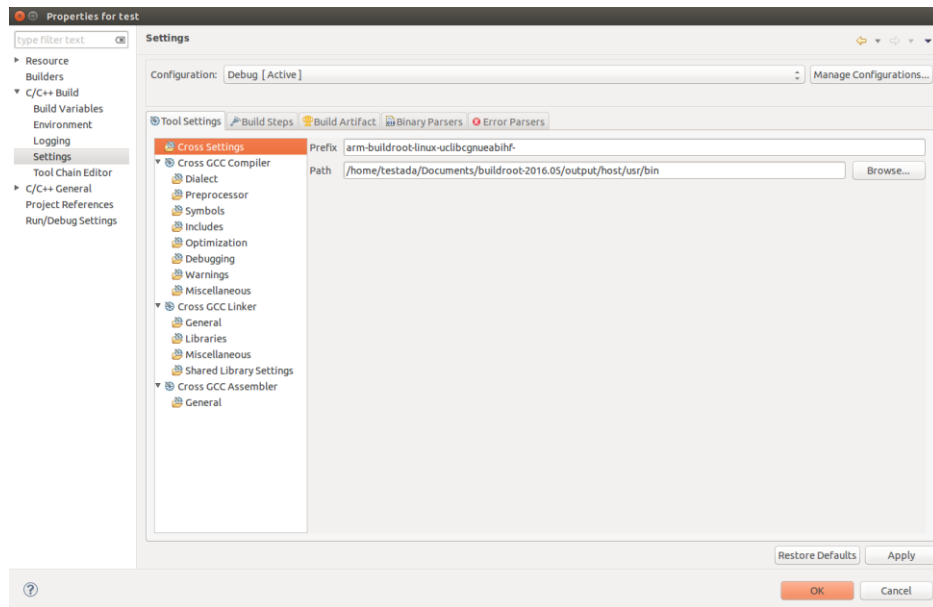


Fig. 31: Cross tools locate on (path).The path shown in this figure is an example.Use always the path of your toolchain.

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. 32 and Fig. 33. Please consider the paths. The figures are showing examples for an specific user account.

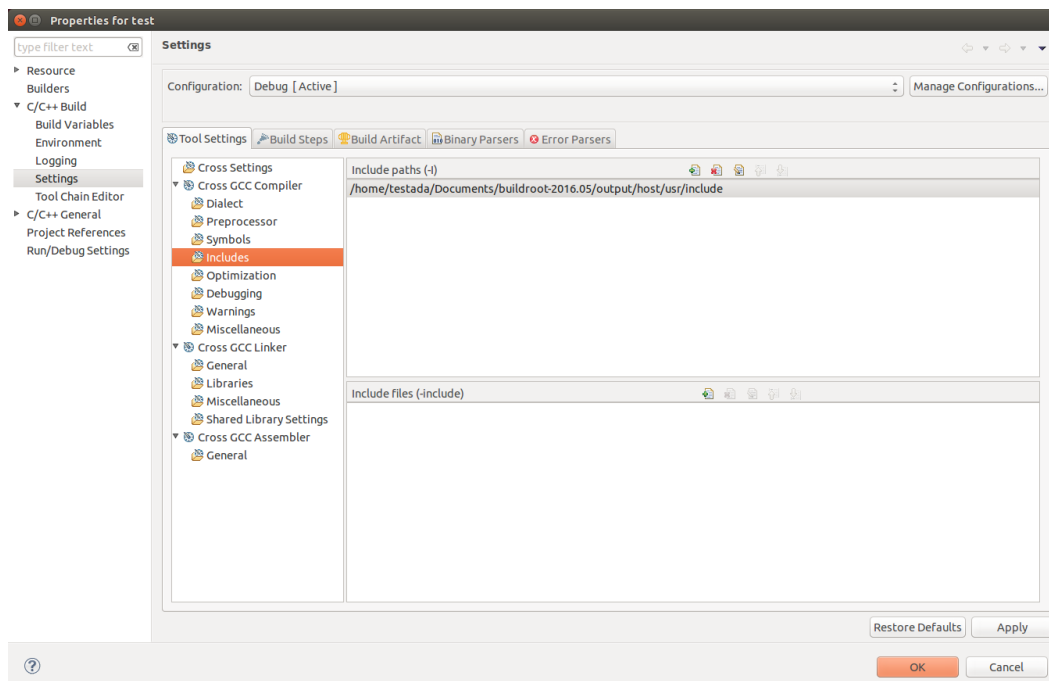


Fig. 32: Include search path.

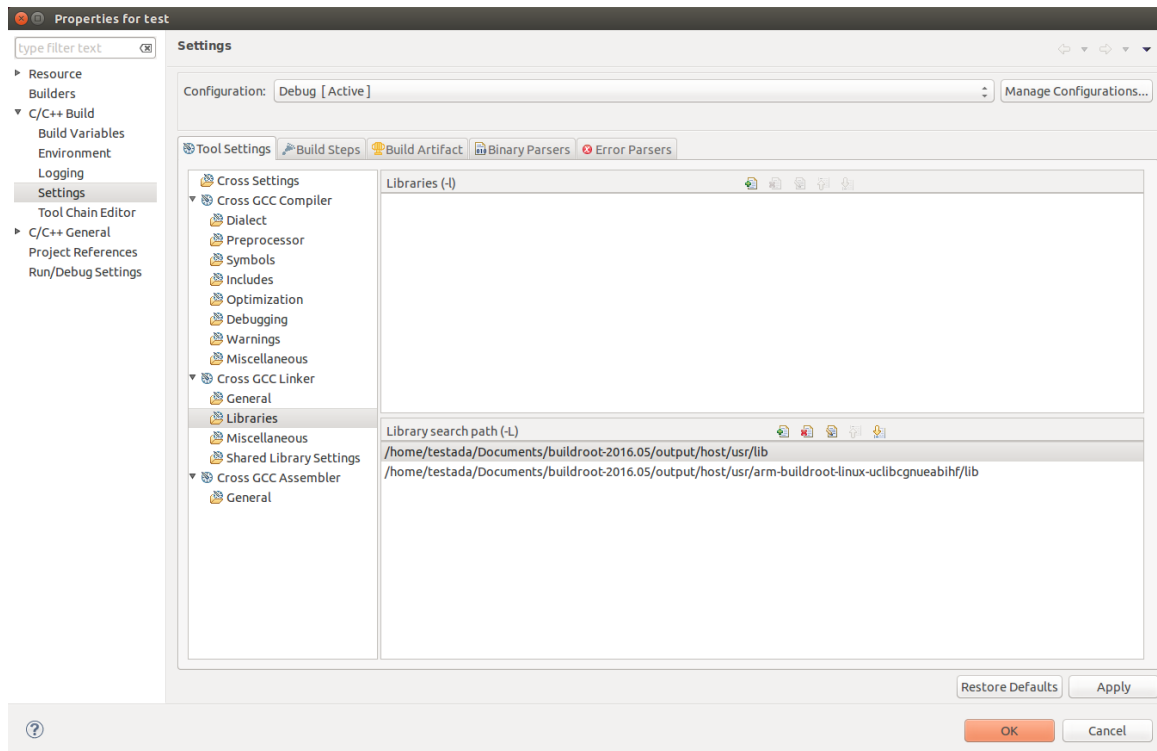


Fig. 33: Libraries search path.

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. 35.



[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 executable 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>`

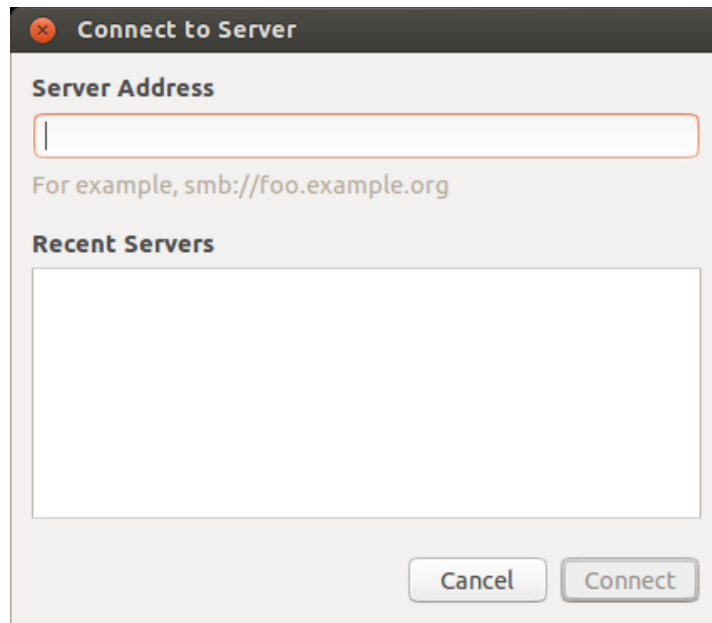


Fig. 34: Pop up window when executing “Connect to Server”

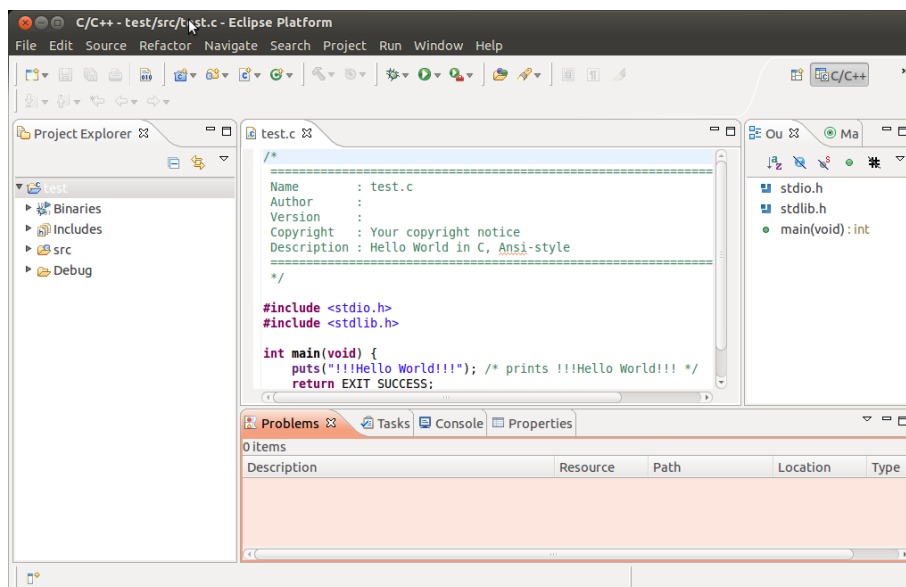


Fig. 35: Eclipse project compiled (Binaries has been generated).

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

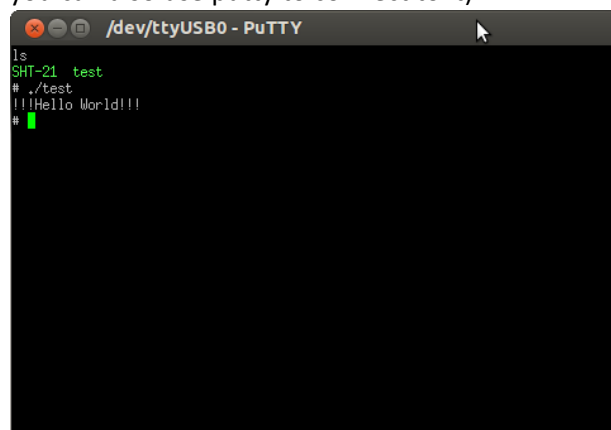


Fig. 36: Run test program in Raspberry PI



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 Raspberry-pi 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 BBB, 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. 37). 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.

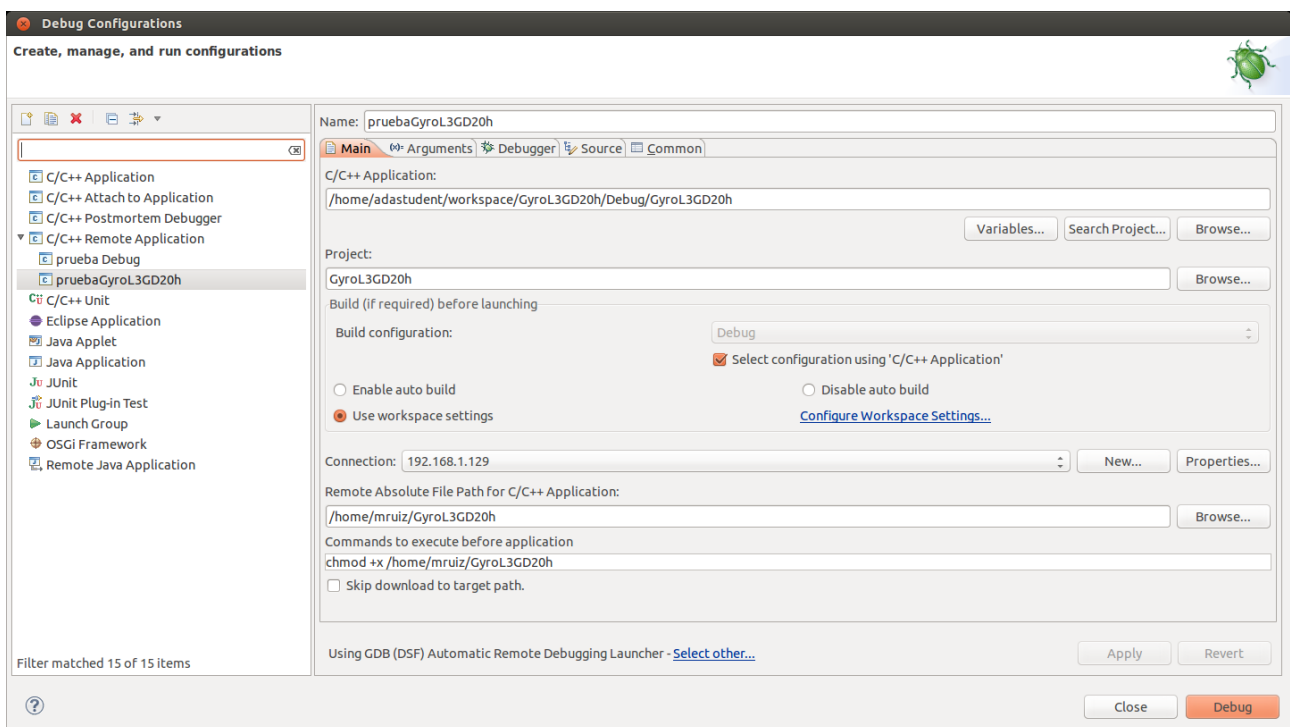


Fig. 37: Creating a Debug Configuration

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).

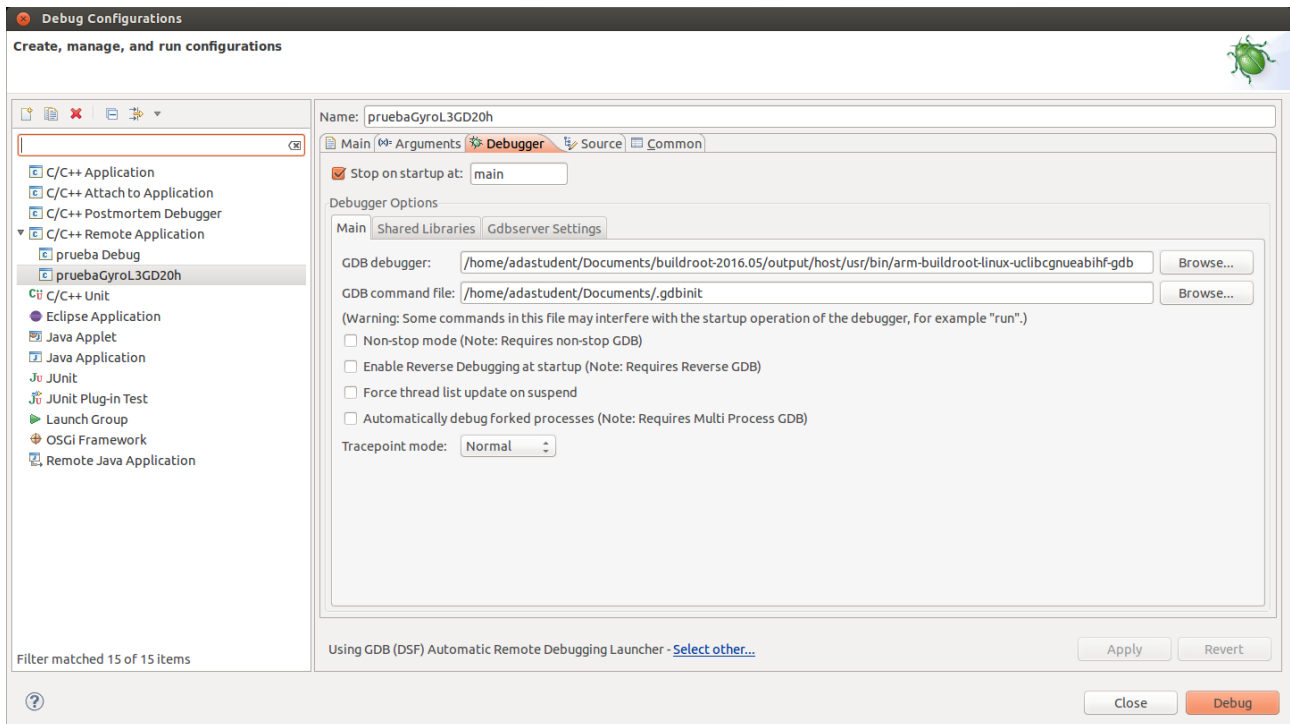


Fig. 38: Debug configuration including the path to locate the cross gdb tool.

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 VMware player and install Ubuntu using the VMware player instructions. Consider the following when creating the virtual machine: you need at least 80Gbytes of hard disk space (in multiple files) and 1GByte of RAM. The installation time will be half an hour more or less depending of your computer. Moving a virtual machine from one computer to another is a time consuming task, therefore, take this into account to minimize the development time.

5.3 Installing packages for supporting Buildroot.

The annex I contains the instructions for downloading the list of packages installed in the Ubuntu 14.04 LTS in order to run correctly Buildroot tools.

6 ANNEX I: UBUNTU 14.04 LTS PACKAGES INSTALLED.

6.1 List of packages used by Buidlroot .

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>. 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.

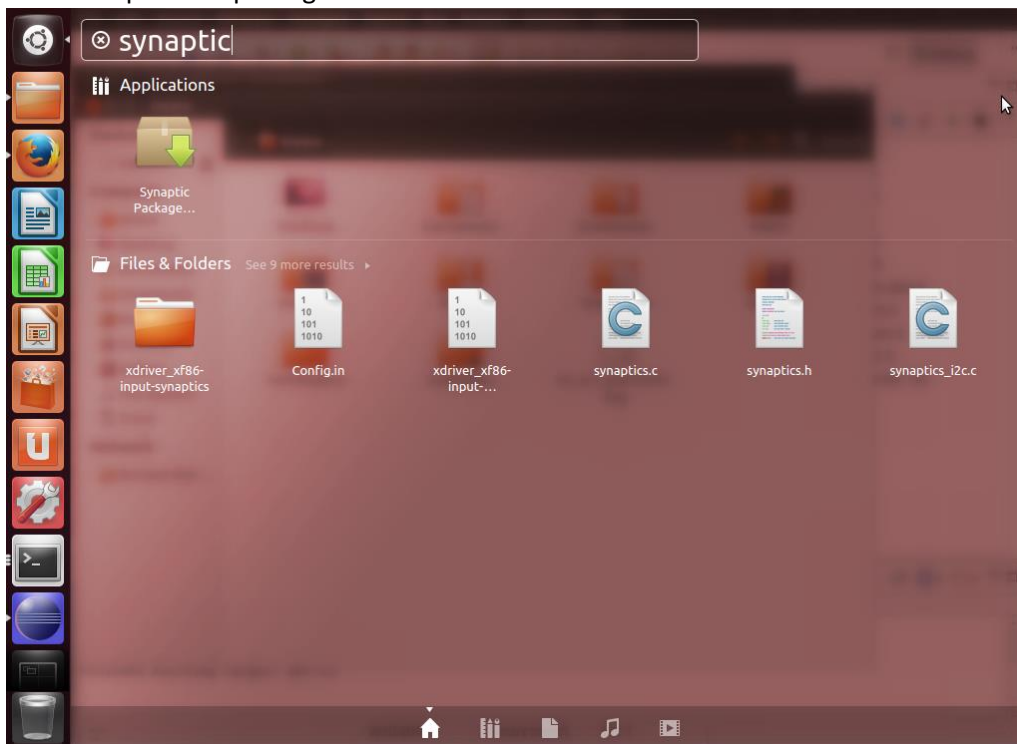


Fig. 39: Synaptic program from Dash

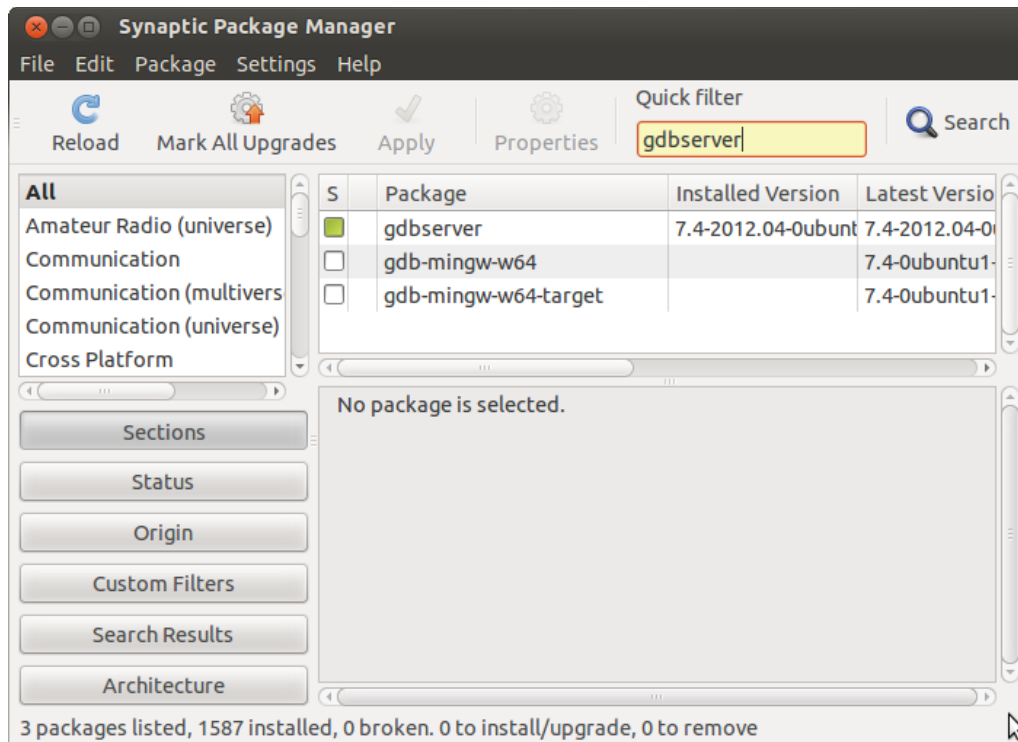


Fig. 40: Synaptic windows

6.2 Typical Ubuntu virtual machine problems

6.2.1 Manually installation of VMware client tools in a Linux Virtual Machine.

Check the tag "*Manually Install or Upgrade VMware Tools in a Linux Virtual Machine*" in VMWare player help file to see how to install client tools.

6.2.2 Ubuntu presents a black screen after graphical login

<http://www.ubuntugeek.com/ubuntu-tip-how-to-removeinstall-and-reconfigure-xorg-without-reinstalling-ubuntu.html>

6.2.3 Ubuntu is using us keyboard and not Spanish one

In a terminal window change the keyboard with this command: `loadkeys es`