what is a toolchain??

A toolchain is the set of **compiler + linker + librarian + any other tools you need to produce the executable** (+ shared libraries, etc) for the target. A debugger and/or IDE may also count as part of a toolchain.

the tools include the cross-compiler and other utilities.

What is the difference between them?

Image: the generic Linux kernel binary image file.

zImage: a compressed version of the Linux kernel image that is self-extracting.

uImage: an image file that has a U-Boot wrapper (installed by the **mkimage** utility) that includes the OS type and loader information.

A very common practice (e.g. the typical Linux kernel Makefile) is to use a zImage file. Since a zImage file is self-extracting (i.e. needs no external decompressors), the wrapper would indicate that this kernel is "not compressed" even though it actually is.

eg:**GNU toolchain**

Toolchains have a loose name convention like arch[-vendor][-os]-abi.

- arch is for architecture: arm, mips, x86, i686…
- svendor is tool chain supplier: apple,
- os is for operating system: linux, none (bare metal)
- abi is for application binary interface convention: eabi, gnueabi, gnueabihf

For your question, arm-none-linux-gnueabi and arm-linux-gnueabi is same thing. arm-linux-gcc is actually binary for gcc which produces objects for ARM architecture to be run on Linux with default configuration (abi) provided by toolchain.

Cross-compilation:

Cross-compilation is the act of **compiling** code for one computer system (often known as the target) on a **different system**, called the host.

It's a very useful technique, for instance when the **target system is too small** to host the compiler and all relevant files.
Common examples include many **embedded systems**, but also **typical game consoles.**
In a strict sense, it is the compilation of code on one host that is intended to run on another.

Therefore:
To compile the linux kernel for the BeagleBone Black, you must first have an ARM cross compiler installed.

step 1) Installing the Cross-Compiler:
To install the compiler (default):
sudo apt-get install gcc-arm-linux-gnueabi
this is by default available if you want different cross compiler like **linaro** etc it can also be used
benifit:

- The path to this compiler is **/usr/bin,** which should already be part of the PATH environment variable, so no need to change it.
- If you type ls /usr/bin/arm* then you will see that the executables are named arm-linux-gnu-gcc for example.

OR

To install the compiler linaro:

- dowload it from
- wget http://releases.linaro.org/14.04/components/toolchain/binaries/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux.tar.xz
- The compiler will be extracted to **/opt/ directory**.
- opt is nothing but the **optional directory.**
- Next, step is to **add the compiler to the PATH variable**, in order to direct the shell to find our compile

- Go to /opt/ directory and change the directory name for adaptivity.
- Then, add the compiler to PATH variable.(do it properly)
    - $ cd /opt/
    - $ sudo mv gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/ gcc-arm-linux
    - $ `export PATH=$PATH:/opt/gcc-arm-linux/bin → vi ./profile`
    - you can verfiy the installation
        - arm-linux-gnueabihf-gcc --version
        - output: arm-linux-gnueabihf-gcc (crosstool-NG linaro-1.13.1-4.8-2014.04 - Linaro GCC 4.8-2014.04) 4.8.3 20140401 (prerelease)

step 2)lzop Compression:

- The Linux Kernel is compressed using **lzo**.
- Install the lzop parallel file compressor:
    - **sudo apt-get install lzop**

step 3)U-Boot → universal boot-loader

- The bootloader used on the BeagleBone black is **U-Boot.**
- U-Boot has a special image format called **uImage.**
- It includes **parameters** such as descriptions, the machine/architecture type, compression type, **load address**, checksums etc.
- To make these images, you need to have a **mkimage tool** that comes part of the U-Boot distribution.
- U-boot is an **open source universal bootloader** for Linux systems.
- It supports features like **TFTP, DHCP, NFS etc…**
- In order to boot the kernel, a valid kernel image **(uImage)** is required.

- It is not possible to explain u-boot here, which is beyond the scope of this post. So, we will see how to produce a bootable image using U-boot.

Building U-boot image → **MLO** and **u-image.bin** using tool sandbox

- Building U-Boot requires **libssl-dev** to be installed:
  - **sudo apt-get install libssl-dev**

  Download U-Boot, make and install the U-Boot tools:

  → wget ftp://ftp.denx.de/pub/u-boot/u-boot-latest.tar.bz2

  → tar -xjf u-boot-latest.tar.bz2

  → cd into u-boot directory

  → **make sandbox_defconfig tools-only**

  → sudo install tools/mkimage /usr/local/bin

  OR → Manually

  - Before, compiling U-Boot we need to configure it.
  - Thanks to the availability of configuration files in the **configs/** directory under **u-boot.**
  - We can configure using the **am332x_boneblack_defconfig file.**
  - All the configuration will be written to **.config file** located in u-boot/ directory.
  - By default you will not be able to view the .config file.
  - To view give ls -a command.
    - $ cd u-boot
    - **$ root am335x_boneblack_defconfig**

Cross-Compiling the u-boot image:

- **$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-**

- It will take around 10 to 15 minutes depending on the system configuration. Mine is Pentium dual core processor and it took 10 minutes for compilation.
- After **successful compilation**, several files will be produced in u-boot/ directory. Our prime concern is **MLO** and **u-boot.img** files.

step 4)Compiling the BeagleBone Black Kernel:

1)configuring:

Before compiling the kernel we need to configure it. Once again, thanks to the kernel developers for providing all configurations in a single file.

- Kernel configurations for ARM based SOC are available in the below folder.
    - → ls arch/arm/configs/*
- Kernel configuration for beaglebone black is :
    - bb.org_defconfig
- To apply this configuration, run below command.
- **$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-bb.org_defconfig**
- This command write it configurations to the **.config file** available in the kernel source root directory. You will get below log.

```
#
# configuration written to .config
#
```

Then compile the kernel.

2)Compiling:

- **git clone git://github.com/beagleboard/linux.git → download**
- cd linux → cd

- **git checkout 4.1** → get the 4.1 setting
  Make command compiles all drivers, modules, device trees, that are specified in the configuration file.
- This is for compiling the kernel →
- **(make ARCH=arm CROS S_COMPILE=arm-linux-gnueabi- -j4)**
  creating uImage
- **(make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage dtbs LOADADDR=0x80008000 -j4)**
- The above command will compile the kernel using the arm cross compiler having the load address as 80008000. "j4" corresponds to the number of process to be run during the compilation.
- After compiling you can find the image files in "arch/arm/boot/" directory.
- Copy "**uImage**" file from this directory and also "am335x-boneblack.dtb" file from **"arch/arm/boot/dts/"** directory to the BOOT partition.

step 5) creating uEnv.txt:

Write the following thing to file →

console=ttyS0,115200n8

netargs=setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext4 rootwait debug earlyprintk mem=512M

netboot=echo Booting from microSD ...; setenv autoload no ; load mmc 0:1 ${loadaddr} uImage ; load mmc 0:1 ${fdtaddr} am335x-boneblack.dtb ; run netargs ; bootm ${loadaddr} - ${fdtaddr}

uenvcmd=run netboot

step 6) Creating Partition:

- For creating partition and othere memory related tast we are going to use tool → **gparted**
  sudo apt-get install gparted
- Insert your **sd card** by means of card reader and open Gparted.
  - **sudo gparted**
- Select your sd card from the **top right corner.** it will be something like this, **"/dev/sdb"**

Note: Always use sd card of size greater than 2 GB. Although, 500 MB is more than enough for our task, having large free space will come handy at times.

- Right click on the **rectangular area** showing your sd card name and select **unmount** as we need to unmount the existing partitions.
- Then, delete the existing partitions by again **right clicking** and selecting "delete".
- This will delete all your files in sd card, so make sure you backed off any important files.
- We need two partitions in order to boot the kernel, one is for storing the **bootable images** and another one is for storing the **minimal RFS(Root File System).**
- Select new option by right clicking the partitions and provide the following details:
  - New size: 50MBembed journal
  - File System: FAT32
  - Label: BOOT
- Click Add button. Then, create another partition for storing RFS by entering the options below
    - New Size: 1000MB
    - File System: EXT3
    - Label: RFS
- Finally, click the **green tick mark** at the menu bar

- The partition will be created and you can see two partitions created as **BOOT and RFS.**

step7)Copying to the sd card:

- **Download RFS** :
- https://www.dropbox.com/s/k93doprl261hwn2/rootfs.tar.xz?dl=0

- Instead of downloading RFS, we can create own own custom RFS using BusyBox
- So, as of now download and de-compress the RFS.
- copy the rootfs to RFS partition in sd card which **may** be as follows
- **sudo tar -xvf rootfs.tar.xz -C /media/*rohit*/RFS/**
- **cd /media/rohit/RFS/rootfs/**
- **sudo mv ./* ../**
- **cd ../**

The above command will **de-compress** the tar file and will place it in the RFS partition of sd card. Just replace "host-name" with your username in the above command.

step 8)Installing kernel Modules:

$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules -j4

$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=/media/rohit/RFS/ modules_install

That's it. After completing the above steps, remove the sd card and place it in your BeagleBone.

Connect the Bone to your PC via USB to serial converter and open the serial console using minicom in PC. (Give baud rate as 115200).

_____XXX_____

Booting in beaglebone -black:
https://www.twam.info/hardware/beaglebone-black/u-boot-on-beaglebone-black

Booting sequence of BBB: → using MLO(spl) and u-boot.img

- The **AM335X** contains **ROM code** that can **load a bootloader** from external memory such as the on-board eMMC.
- On reset, this **ROM code searches for the bootloader** and then **copies** it to the **internal RAM** before executing it.
- The internal RAM on the AM335X is **128KB**, but due to various limitations, only 109KB is available for the **initial bootloader for program memory**, heap and stack.
- A fully featured version of U-Boot can be over **400KB**, hence it is **not** possible to **load this immediately**.
- For this reason, a cut down version of **U-Boot** called **U-Boot SPL** (**Second Program Loader)** is loaded **first**, and once it has initialised the CPU, it chain loads a fully featured version of **U-Boot (u-boot.img)**.

The BeagleBone Black provides by **alternative boot sequences** which are selectable by the boot switch (S2). In default mode (S2 not pressed) it tries to boot from

1. MMC1 (onboard eMMC),
2. MMC0 (microSD),
3. UART0,
4. USB0.

Usually it will find something in the onboard eMMC and boot from there. If S2 is pressed during power-up the boot sequence is changed to

1. SPI0,
2. MMC0 (microSD),
3. UART0,
4. USB0.

- As there is usually **nothing bootable** found on **SPIo** it will boot from the **microSD** card.
- The onboard **eMMC** or an external microSD card have to be formatted in a special way for the AM335x processor to find its boot file, which is described very good on the MMC boot format wiki page at TI.
- If the AM335x processor finds a **valid formatted MMC** it searchings for a file named **MLO** on the **first partition** and if it is **found it boots from that file**.

- U-Boot is a very **versatile boot loader** which can be used on the BeagleBone Black.
- U-Boot provides this **MLO file as a second-stage boot-loader** which then loads the actual **U-Boot** which has to be provided as a file named **u-boot.bin** in the same directory.
- U-Boot itself will then look for a file named **uEnv.txt** for further **configuration** and then act upon it.
-

uEnv.txt:

- With u-boot each partition of the **SD card will be searched** for an **environment file** under **/boot/uEnv.txt.**
- If an uEnv.txt file is found, its **contents are imported.**
- The name of a **dtb** file may be set with the variable '**dtb**' within the uEnv.txt file.
-

Booting sequence of ARM/beaglebone:

5 Stages to Booting:

Stage 1

- The ROM loads.This is on board read only(built in) and can't change. It looks for the MLO file and runs it.

Stage 2 (x-loader)

- MLO file runs and looks for zImage

Stage 3 (u-boot)

- ZImage loads & run with configuration uEnv.txt
- uEnv.txt files have info on where to find the **linux kernal plus** lots of other parameters
- Parameters can be passed from the uEnv.txt file into Linux kernal (provided the kernal was complied with the required modules for the parameters)

Stage 4

- Linux Kernal loads

Stage 5

- Root file system loads (e.g. debian, ubuntu, etc.)

Detailed Booting Sequence of Arm:

- In an ARM embedded system, at the time of power on, CPU is uninitialized and(reset)
- a basic clock setup, system specifics' setup is required before proceeding to the bigger and complex tasks.

- For this a piece of code is required at power on which does this basic system setup before handing over the control to the bootloader present in flash(already programmed) .
- Hence a **hardware bootloader** generally called as **Boot Rom** is provided by vendor (pre-loaded into the processors' internal ROM).
- This is hardwired at the manufacturing time.
- After a power on reset, this causes the processor core to jump to the reset vector, **Boot Rom is the first code to execute in the processor.**
- Responsibilities of Bootrom :
    - Bootrom performs the essential initialization including **programming the clocks,** stack initialisation, **interrupt set up** etc.
    - This is to determine where to find the software bootloader.
    - A booting device list is genertated by this ROM code.
    - The booting devices can either be **memory booting devices or a peripheral interface connected to a host**.
    - The ROM code goes through this device list, and tries to look for a valid booting image.
    - Take note that this is not the Linux kernel image.
    - It looks for a much smaller image, which can be of **109Kb at max**.
    - This image is the MLO image(x-loader) that is stored on either the eMMC/SD.
    - This is one of the many things that this ROM code does. It also does some other stuff like configuration of the clocks for executing it's own as well as other codes, sets up the stack and configures the watchdog timer(as discussed above).
    -
- **MLO(xloader):**
    - It will look for the the MLO, either on the eMMC, SD card or any other peripheral device and then loads it up into the public RAM. This MLO file is not part of the ROM, it is located on the boot media that you are using. In normal cases, it will be the eMMC or the SD card. You can check out the binary version of this in *boot/uboot* directory.
    - root@beaglebone:/boot/uboot# ls -lah MLO-rwxr-xr-x 1 root root 103K Mar 4 23:37 MLO

    - The MLO is "secondary program loader". It will run and configure off-chip memory and then load the U Boot image. The MLO and the u-boot.img that it loads, both have to be located on a FAT filesystem.
    -
    - The MLO is built when you build the U Boot bootloader.
- U-boot:
    - The U Boot image is located in the *boot/uboot* directory on our BeagleBone Black
    - root@beaglebone:/boot/uboot# ls -lah u-boot.img-rwxr-xr-x 1 root root 358K Mar 4 23:37 u-boot.img
    - You will notice that it's **bigger than the MLO** that we saw in the previous section.
    - **Das U Boot is the most commonly** used bootloader on embedded systems. More about the project can be found here
    - If you connect a 3.3V USB-Serial cable to the BeagleBone Black, you will see the following on a screen session when the BBB boots up.
        - U-Boot SPL 2013.10-00016-ga0e6bc6 (Feb 25 2014 - 10:27:54)reading argsspl: error reading image args, err - -1
    -

      reading u-boot.img
    -

      reading u-boot.img
- It loads the device tree, initializes pins as per the data structures there and then loads the Linux kernel.

The BeagleBone Black provides by **alternative boot sequences** which are selectable by the boot switch (S2). In default mode (S2 not pressed) it tries to boot from

5. MMC1 (onboard eMMC),
6. MMC0 (microSD),
7. UART0,
8. USB0.

Usually it will find something in the onboard eMMC and boot from there. If S2 is pressed during power-up the boot sequence is changed to

5. SPI0,
6. MMC0 (microSD),
7. UART0,
8. USB0.

- As there is usually **nothing bootable** found on **SPI0** it will boot from the **microSD** card.
- The onboard **eMMC** or an external microSD card have to be formatted in a special way for the AM335x processor to find its boot file, which is described very good on the MMC boot format wiki page at TI.
- If the AM335x processor finds a **valid formatted MMC** it searchings for a file named **MLO** on the **first partition** and if it is **found it boots from that file**.

- U-Boot is a very **versatile boot loader** which can be used on the BeagleBone Black.
- U-Boot provides this **MLO file as a second-stage boot-loader** which then loads the actual **U-Boot** which has to be provided as a file named **u-boot.bin** in the same directory.
- U-Boot itself will then look for a file named **uEnv.txt** for further **configuration** and then act upon it.