There are two types of Compilation:
1.Native Compilation
2.Cross Compilation

Native Compilation:

What this means is that you are trying to build an object file or a module for the same underlying hardware that you are using on your host.By default Linux builds for the same architecture the host system is running. This is called “native compiling”. An x86 system building an x86 kernel, x86-64 building x86-64,or PowerPC building PowerPC are all examples of native compiling.

ex:
Assuming you are trying to run a C-Code(test.c) in Ubuntu(linux) on Intel Hardware machine.

- gcc -o test test.c

The result of this compilation will generate an executable file which will work on the x86 processor but will not work on ARM.

Cross Compilation:

It is the process in which user will try to generate the binaries or the executable files for other architectures.You use a cross compiler to produce executable (or objects) for a platform other than the local host.Cross compiling is hard. The build system for the Linux kernel supports cross compiling via a two step process:
1) Specify a different architecture (ARCH) during the configure, make, and install stages.
2) Supply a cross compiler (CROSS_COMPILE) which can output the correct kind of binary code.
An example cross compile command line (building the “arm” architecture) looks like:

```
make ARCH=arm menuconfig
make ARCH=arm CROSS_COMPILE=armv5l-
```

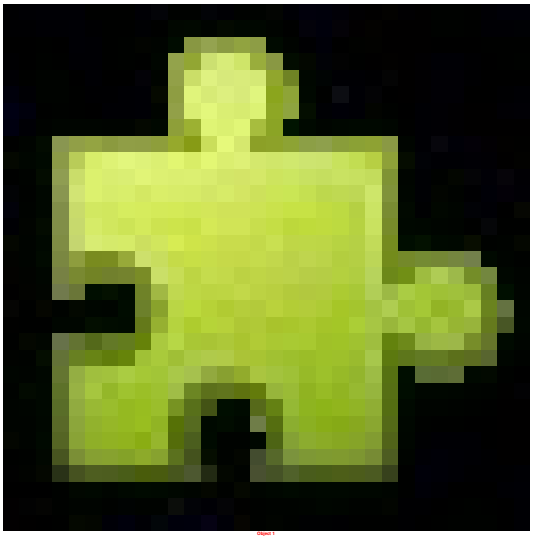
Don't bother about the above two statements,you will get to know about that later.

Steps to Make your Setup ready:

Things you require:

1.PC with Ubuntu in it.
2.A Raspberry pi with Raspbian OS(Jessie) in it.

Connect your raspberry pi to the PC and try to remotely login into it using ssh.
This step can be done by following the video below:



The next step is to download the required source code files and the build tool chain for arm.

Step 1:
create a directory named devicedriver inside a directory named Work in your home directory.

- cd ~;mkdir Work;cd Work;mkdir devicedriver;cd devicedriver

Step 2:get the source files into the devicedriver directory

- git clone --depth=1 https://github.com/raspberrypi/linux

After the execution of above command you will find a directory named linux inside the devicedriver directory.

Step 3:get the build tool chain in the devicedriver directory

- git clone https://github.com/raspberrypi/tools

After the execution of above command you will find a directory named tools inside the devicedriver directory.

Step 4:Look into the tools directory

In the arm-bcm2708 folder in the tools directory ,you'll see four other folders, each containing a separate tool chain:

- arm-bcm2708-linux-gnueabi
- arm-bcm2708hardfp-linux-gnueabi
- gcc-linaro-arm-linux-gnueabihf-raspbian
- gcc-linaro-arm-linux-gnueabihf-raspbian-x64

So what this folders Mean?
arm--Architecture of the target(Raspberry pi have arm)
linux--os used host(Laptop)
gnuabihf--application binary interface(Don't bother about it at this point)
raspbian--OS of the target(Raspberry pi)
x64--Architecture of the host(Laptop-64 bit host)

So in this experiment i am using the gcc-linaro-arm-linux-gnueabihf-raspbian-x64 as my tool chain.You can use different tool chain as well but for time being i used the one mentioned above.

Step 5 :Adding the tool chain to your system PATH

- export PATH=$PATH:$HOME/Work/devicedriver/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin

While this step is not strictly necessary, it does make it easier for later command lines!

Type the below command to execute the bashrc file with the PATH variable updated

- source .bashrc

After this command you can even check whether your tool chain path is added to your system path or not just by typing $PATH

Step 6: Generate the .config File
In most of the cases we will generate the .config file with the default configuration.
So first we have to navigate to linux directory in the devicedriver directory and then

For Pi 1 or Compute Module:type

- KERNEL=kernel
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig

For Pi 2/3:Type

- KERNEL=kernel7
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig

Then for both type the below :

- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs

Note: To speed up compilation on multiprocessor systems, and get some improvement on single processor ones, use -j n where n is number of processors * 1.5. Alternatively, feel free to experiment and see what works!

After this step is done ,just check the linux directory ,you will be able to find the file named **Module.symvars** and some other files,and in the arch/arm/boot/ folder you will find an uncompressed image file named **Image** and a compressed image named **zImage.**

But for cross compilation there is no need of Image file or even the zImage file,what you need are the intermediate files in the build process of the kernel that will be generated after the successful kernel build.

What you are going to read in the next paragraph is the most important step that needs to be grasped,because i found that many resources in the Internet are quite confusing mixing the kernel source compilation and the cross compilation.

Cross Compilation means, building your kernel and modules on a different platform rather than on the Raspberry Pi hardware directly. I did NOT have any luck compiling my kernel module directly on Raspberry Pi hardware.

In detail, first you'll want to cross-compile the Raspberry Pi

Linux kernel(which you did in this step) so it builds Module.symvars and other intermediates. You won't actually use the kernel image that is being cross-compiled, just the intermediates to compile your module. Next, you'll cross-compile your module using those intermediates. Once your module is cross-compiled, copy the .ko module to your running Raspberry Pi and you should be able to load it along with the stock kernel.

But just for being safe side,do the following steps also,or else you may land in having a module which works for one version of kernel and your raspberry pi might me having a different kernel version.

Remove the SD card from the raspberry pi and insert into your computer using a card reader,ensure that the hardware write lock is removed,then type the below commands:

Check for the mount point of your SDcard using **sudo fdisk -l** command.In my case the boot partition is mounted to **/dev/sdb1** and my root partition is mounted to **/dev/sdb2**..But in your case it might be different.

- mkdir /mnt/fat32
- mkdir /mnt/ext4
- sudo mount /dev/sdb1 mnt/fat32
- sudo mount /dev/sdb2 mnt/ext4

Next, install the modules into the SD card:

- sudo make ARCH=arm CROSS_COMPILE=$(HOME)/Work/devicedriver/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf- INSTALL_MOD_PATH=mnt/ext4 modules_install

Finally, copy the kernel and Device Tree blobs onto the SD card, making sure to back up your old kernel:

Before executing the below commands ensure that KERNEL variable is set to kernel7 by using $KERNEL command,if the output is kernel7,then it is fine,otherwise type KERNEL=kernel7 and press enter.

- sudo cp /mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
- sudo scripts/mkknlimg arch/arm/boot/zImage /mnt/fat32/$KERNEL.img
- sudo cp arch/arm/boot/dts/*.dtb /mnt/fat32/
- sudo cp arch/arm/boot/dts/overlays/*.dtb* /mnt/fat32/overlays/
- sudo cp arch/arm/boot/dts/overlays/README /mnt/fat32/overlays/

- sudo umount /mnt/fat32
- sudo umount /mnt/ext4

Finally, plug the card into the Pi and boot it!

So if the build is successful then our cross compilation is done.

Step 7: Time to Test

Now make a directory named **code** in the devicedriver directory. In that create two files named test.c and Makefile.

Content of test.c:

```
#include <linux/module.h>

static int __init simple_init(void)
{
    pr_info("Hi ManojKiran %s ..\n", __FUNCTION__);
    return 0;
}

static void __exit simple_exit(void)
{
    pr_info("Bye ManojKiran %s ...\n", __FUNCTION__);
}

module_init(simple_init);
module_exit(simple_exit);
MODULE_LICENSE("GPL");
```

Contents of MakeFile:

```
COMPILER:=$(HOME)/Work/devicedriver/tools/arm-bcm2708/gcc-linaro-arm-linux-
gnueabihf-raspbian-x64/bin/arm-linux-gnueabihf-
SOURCE:=$(HOME)/Work/devicedriver/linux/
PWD:= $(shell pwd)
obj-m := test.o
default:
  make ARCH=arm CROSS_COMPILE=${COMPILER} -C $(SOURCE) M=$(PWD) modules
  rm *.mod.c *.o
clean:
 -rm *.mod.c *.o *.ko modules.order Module.symvers 2>/dev/null
```

Then type make in the code directory.If every thing works fine

,then you should see test.ko file being generated in the test directory.

Step 8:

Test it by insmod command
In Laptop it should generate an error because that module is built for arm architecture.
Type below command in laptop(Ubuntu) in code directory ,and you should get an error like this

Command:

- sudo insmod test.ko

Output:
insmod: ERROR: could not insert module dr2016f1.ko: Invalid module format

Step 9:

Now time to test the test.ko module in the raspberry pi and it should work.

Use scp to send the test.ko module to raspberry pi using the below command in the code directory.

- sudo scp test.ko pi@IPADDRESS:/home/pi

Then you can find the test.ko file lying in the home directory of raspberry pi.Then type this command in the home directory of raspberry pi.

- sudo insmod test.ko

if the above command is successful then nothing will be visible.To see if it actually worked type the following command

- dmesg

The last line of the output of the dmesg will be Hi ManojKiran

****

Thanks For reading my post.Hope you guys enjoyed it and learnt

deltax amount today.Fell free to comment my efforts.