# HOWTO: VM, Cross compile, execute code on ARM/Linux

## Virtual Machine usage and folder structure

The most recent version of the virtual machine, that was used to cross compile code for the embedded Linux system and also run the code for the NIOSII system is located in:

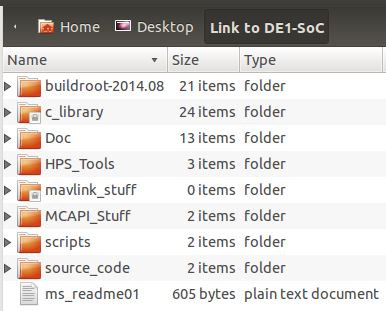
*C:\USERS\Public\QuadCopter\Ubuntu\_1204\_64bit\_de1soc\_v03*

on the CTLAB-08 computer. The VM can be opened with the VMware Player that is already installed on most of the computers in the lab. The VM itself is running a 64bit Ubuntu 12 distribution. The system can be accessed with the **username: de1soc** and the **password: hsu**. This user is not running as root natively, so keep in mind that for some actions administrative rights may have to be added (sudo).

All the relevant files are located in the folder:

*/home/de1soc/DE1-SoC*

This folder can also be accessed through a link that is placed on the desktop. It contains the following subfolders:



* **Buildroot-2014.08**
  + Files for execution and configuration of BuildRoot, changing the Linux Image, boot loader image and root file system image.
* **C\_library**
  + Test files (heartbeat, hello\_arm… ) and libraries for MCAPI
* **Doc**
  + Essential documentation files from Mr. Strahnen. This is the most important folder. Reading these documents is highly recommended!
* **HPS\_Tools**
  + Files for successfully booting up the DE1-SOC System (not interesting for development)
* **Mavlink\_stuff**
  + C-Code for testing MAVLINK communication (further information on *qgroundcontrol.org/dev/mavlink\_linux\_integration\_tutorial*).
* **MCAPI\_Stuff**
  + Everything that has to do with MCAPI communication.
  + It is highly recommended to read *howToStartMCAPI01.txt* in this folder
  + *MCAPI\_SYS* folder is for deeper understanding of the protocol, but not necessary for development.
  + *MCAPI\_Apps* contain a lot of test programs to test MCAPI functionality. A good first test would be to try to get *MCAPI\_packetTest\_receivingNode\_cpuS0* (running on NIOS!) and *MCAPI\_packetTest\_sendingNode\_cpuM* (running on ARM/Linux). This test exchanges some data with different package sizes. Do not get confused by the naming of the test programs. Both programs receive and send data! **The most important thing** is to run the receiving program on NIOS and the sending program on ARM/Linux.
  + The latest communication test is in the folder *XCopter\_files* and contains example code for sending a 512-byte package.
* **Scripts**
  + Contains two deprecated scripts. Feel free to insert useful scripts here.
* **Source\_code**
  + Network\_config contains three files that describe how the network is set up on ARM/Linux. These files can be taken as reference if wlan problems should occur.
  + *Test\_files* is an empty folder for temporary code

The cross compiler is only set up on this Ubuntu Virtual machine. You can copy the VM to other PCs if needed. As no cross compiler is set up for a windows system, it is recommended to use the VM exclusively for development for the ARM part of the DE1-SoC platform. At a first glance using a cross compiler seems a little unwieldy and complicated, but there are two main reasons to use a cross compiler in this scenario. Cross compilers are used widely by professionals and it is good to have seen how it works. The ARM CPU is not as fast as an Intel machine running the VM. Compiling directly on ARM would take a lot more time. Also development in an embedded Linux terminal is way harder than in Ubuntu.  
For development of NIOSII applications windows is the first choice, as altera provides a relatively solid development platform based on Ecplipse. There is a HowTo PDF-File from Group Bumblebee (2014) that goes into detail about how to develop for NIOSII with the tools provided by Altera on the DVD with the resources for this project.

## Cross compiling code on Ubuntu for ARM

To be able to compile code for ARM/Linux on another system (Ubuntu in this case), the correct cross compiler has to be set in the corresponding makefile. The cross compiler is located in:

*/home/de1soc/DE1-SoC/buildroot-2014.08/output/host/usr/bin/arm-buildroot-linux-uclibcgnueabihf-cc*

An example for how to compile a file with the cross compile:

*/home/de1soc/DE1-SoC/buildroot-2014.08/output/host/usr/bin/arm-buildroot-linux-uclibcgnueabihf-cc –o <filename of output> <filename of code> -I ./common/*

You can load your executable onto the ARM-system via SCP.

## Using SCP to upload files to the ARM-system

* Check if ARM/Linux is connected to the internet via Ethernet. This should happen automatically after a reboot if an Ethernet cable is plugged in.
* Start Kermit on Ubuntu (*sudo Kermit* in terminal, password is hsu)
* Log in to the embedded Linux (login:pass -> root:hsu)
* To copy via scp type: *scp <filename of file to copy>* [*root@141.59.43.62:/root*](mailto:root@141.59.43.62:/root) (the IP is the IP of the system you want to copy to). This copies the file into the root folder of your ARM/Linux.

### If copying doesn’t work

* The date of the ARM/Linux system might have to be set if copying the files doesn’t work. If the time difference between host and client is too big, scp might fail. The embedded linux system sets its time to 1970, which may hinder the usage of scp. Google provides tutorials for how to set time and date in linux.
* If copying still doesn’t work, you can try to delete old SSH fingerprints with at *~/.ssh/known\_hosts*
* Some root password has to be set before being able to use scp.
* Sometimes it helps to shut down wlan and starting it up afterwards (*ifdown wlan0*)
* Have you tried turning it off and on again?