# Cross Compile files on x86 Linux host for 96Boards ARM systems



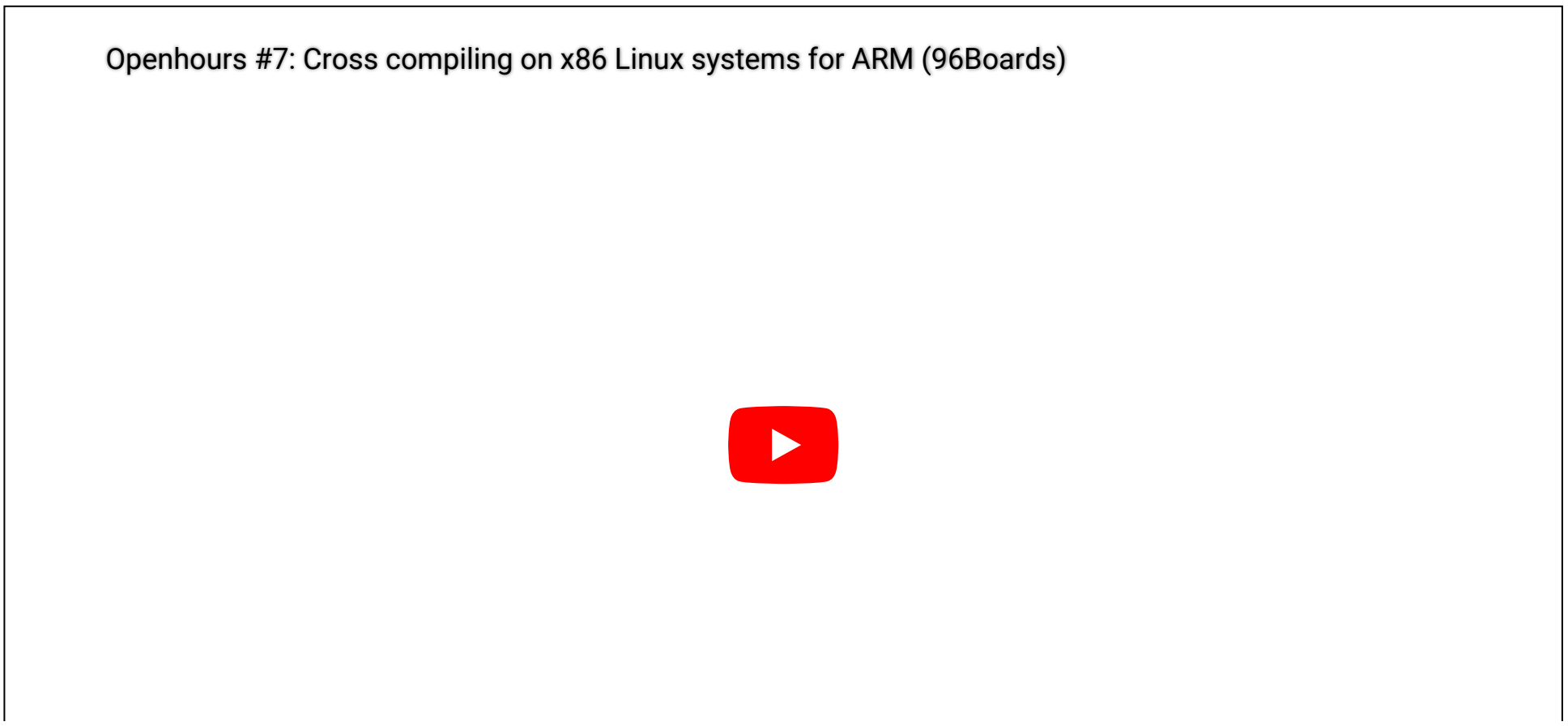


# Cross Compile files on x86 Linux host for 96Boards ARM systems

This three part set of instructions will walk you through basic commandline cross compilation on a Linux x86 system for ARM 96Boards devices.

## Assumptions

- Linux host system is used as the cross compiling station
- Examples were tested on fully updated Ubuntu 15.04 and 16.04 releases
- Examples depend on matching, latest libsoc and libmraa libraries to be installed on both devices (x86 machine, ARM machine)
  - Libraries should be built from source to ensure they are current and will match. Instructions can be found here
- Examples were tested on a DragonBoard 410c, but should work with all 96Boards



This material was covered in our 7th OpenHours session and can be paired with this blog.

# Part 1 - A simple application

Here you will learn to cross compile a simple application using Linux C and C++ toolchains. Cross compilation will happen on a Linux x86 machine for 96Boards ARM device.

## Step 1: Update 96Boards (ARM) system and Host (x86 Machine) computer

The image on your board/host computer might be out of date. This is possible even when using the stock images, recent downloads, or a newly flashed versions of any operating system.

A few useful commands will help us make sure everything on the board is current:

- **apt-get update:** Downloads package lists from online repositories and "updates" them to get information on the newest versions of packages and their dependencies.
- **apt-get upgrade:** Fetches and installs newest package versions which currently exist on the system. APT must know about these new versions by way of 'apt-get update'
- **apt-get dist-upgrade:** In addition to performing the function of upgrade, this option also intelligently handles changing dependencies with new versions of packages

**Commands:**

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

## Step 2: If you are using libsoc and or mraa make sure they are installed and up to date

**Installation libsoc:** Please go here for first time libsoc installation instructions.

**Update:** Change directory (cd) to your libsoc source and make sure you have latest code

**Commands:**

```
$ git pull
$ autoreconf -i
$ ./configure --enable-board=<your board name> --with-board-configs
$ make
$ sudo make install
$  sudo ldconfig /usr/local/lib
```

**Installation mraa:** Please go here for first time mraa installation instructions.

**Update:** Change directory (cd) to your mraa source and make sure you have the latest code.

**Commands:**

```
$ git pull
$ cmake .
$ make
$ sudo make install
$  sudo ldconfig /usr/local/lib
```

## Step 3: Install cross compilers on host machine

The following commands will install C and C++ cross compiler toolchains for 32bit and 64bit devices. You only need to install the toolchain that is the correct size for your board. If your 96Board is a 64bit SoC then only install a 64bit toolchain, if your 96Board is a 32bit board then only install the 32bit toolchain. This document will use the 64bit toolchain.

**For ARM 32bit toolchain**

```
$ sudo apt-get install gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf
```

**For ARM 64bit toolchain**

```
$ sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

## Step 4: Install package dependencies

```
$ sudo apt-get install build-essential autoconf libtool cmake pkg-config git python-dev swig3.0 libpcre3-dev nodejs-dev
```

Step 5: Create a workspace

```
$ mkdir hacking
$ cd hacking
```

## Step 6: Create a helloworld.c file with your favorite editor

**Example (using vim text editor):**

```
$ vim helloworld.c
```

Copy and paste the following into your `helloworld.c` file

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
        puts(“!!!Hello World!!!”);
        return(EXIT_SUCCESS);
}
```

Save and quit (`:wq`)

## Step 7: Compile, test, and run x86 file from the command line

**Compile:**

```
$ gcc helloworld.c -o helloworld.x86
```

**Test:**

```
$ file helloworld.x86
```

> *Print out should show a X86 binary file*

**Run:**

```
$ ./helloworld.x86
```

> *Print out should read `!!!Hello World!!!`*

## Step 8: Cross compile, test, and run ARM file from the command line

**Cross compile:**

```
$ aarch64-linux-gnu-gcc helloworld.c -o helloworld.arm
```

**Test:**

test.

```
$ file helloworld.arm
```

> *Print out should show an aarch64 ARM file*

Run (On 96Boards):

Copy file to 96Boards and run. It should run and say `!!!Hello World!!!`.

Retrieve 96Boards IP address with the following command:

```
$ /sbin/ifconfig
```

Commands(From host machine):

```
$ scp helloworld.arm linaro@{ipaddress of 96board}:.
$ ssh linaro@{ipaddress of 96board}
$ ./helloworld.arm
```

If you got this far congratulations, your basic cross compiling is working! Now let's make it more complex and add a C shared library. For the purpose of the rest of this document we will assume you have installed libsoc and mraa libraries on your 96Boards, they must be current and ready to use.

---

# Part 2 - Shared libsoc C library

Install libsoc, this will take a bit of doing, as we have to cross compile this library and then manually install it so it does not collide with X86 libraries. We use a staged Install process by using the DESTDIR environment variable (below) to redirect the install step into a temporary location so we can move it into the proper cross compile location.

## Step 1: Clone libsoc library and change directory

```
$ git clone https://github.com/JackMitch/libsoc.git
$ cd libsoc
```

## Step 2: Make library

```
$ autoreconf -i
$ ./configure --host aarch64-linux-gnu --enable-board=<your board name> --with-board-configs
$ make
$ sudo make DESTDIR=/tmp/stage install
```

## Step 4: Copy all files to the appropriate directory

```
$ sudo mkdir -p /usr/aarch64-linux-gnu/local/
$ sudo cp -a /tmp/stage/usr/local/* /usr/aarch64-linux-gnu/local/.
```

## Step 5: Change filename from within "test" directory

```
$ cd test
$ cp board_test.c compile_test.c
```

## Step 6: Cross compile, test, and run ARM file from the command line

**Cross compile:**

```
$ aarch64-linux-gnu-gcc -ggdb -O0 -I /usr/aarch64-linux-gnu/local/include/ -L /usr/aarch64-linux-gnu/local/lib/ compile_test.c -o compile_test.arm -lsoc
```

**Test:**

```
$ file compile_test.arm
```

```
$ file compile_test.arm
```

> *Print out should show an aarch64 ARM file*

**Run (On 96Boards):**

Copy file to 96Boards and run.

Retrieve 96Boards IP address with the following command:

```
$ /sbin/ifconfig
```

**Commands(From host machine):**

```
$ scp compile_test.arm linaro@{ipaddress of 96board}:.
$ ssh linaro@{ipaddress of 96board}
$ ./compile_test.arm
```

If you got this far you have command line cross compiling with shared library support installed and working, **congratulations**. Now lets move on to a more complex C++ library.

---

##Part 3 - Shared libmraa C++ library

## Step 1: Set up environment

The Intel mraa library uses the cmake build system which is totally different from the autotools system, so we need to make a control file which will tell it to do cross compiling. Since the library is also a C++ library with an extra C interface, we have to tell the build system the location of the C and C++ compilers.

```
$ cd ~/hacking
```

Create a file using your prefered editor named `aarch64.cmake` - This example will use vim text editor.

```
$ vim aarch64.cmake
```

This file will tell the cmake build system that it is to cross compile the code and to use the cross compile toolchain.

Copy and paste the following into the `aarch64.cmake` file

```
# this one is important
SET(CMAKE_SYSTEM_NAME Linux)
#this one not so much
SET(CMAKE_SYSTEM_VERSION 1)

# specify the cross compiler
SET(CMAKE_C_COMPILER   /usr/bin/aarch64-linux-gnu-gcc)
SET(CMAKE_CXX_COMPILER /usr/bin/aarch64-linux-gnu-g++)

# where is the target environment
SET(CMAKE_FIND_ROOT_PATH /usr/aarch64-linux-gnu)

# search for programs in the build host directories
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
# for libraries and headers in the target directories
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
# end of the file
```

Save and quit (`:wq`)

## Step 2: Clone mraa library, cross build it, and install it.

Doing this will allow you to cross compile apps which use this library. You won't need any dependencies such as: swig, python or node is on

Doing this will allow you to cross compile apps which use this library. You won't need any dependencies such as: swig, python or node.js on your cross compile machine, instead we will use command line flags to not attempt to build them.

First, make sure `/tmp/stage` directory does not exist by executing the following command:

```
$ sudo rm -Rf /tmp/stage
```

Clone libmraa library and change directory

```
$ git clone https://github.com/intel-iot-devkit/mraa.git
$ cd mraa
```

Cross build library using cmake and make commands with the appropriate tags

```
$ cmake -DBUILDSWIG=NO -DBUILDSWIGNODE=NO -DBUILDSWIGPYTHON=NO -DCMAKE_TOOLCHAIN_FILE=../aarch64.cmake .
$ make
```

Install library using the `make install` command

```
$ sudo make DESTDIR=/tmp/stage install
$ sudo mkdir -p /usr/aarch64-linux-gnu/local/
$ sudo cp -a /tmp/stage/usr/local/* /usr/aarch64-linux-gnu/local/.
```

## Step 3: Test an application which uses the shared library mraa.

The mraa library builds all of it's example files in the process of building the library, this means you don't need to invoke the compiler it's already been done.

First change to the `examples` directory

```
$ cd examples
```

**Test:**

```
$ file hellomraa
```

> *This print out should show an aarch64 ARM file*

**Run (On 96Boards):**

Copy file to 96Boards and run.

Retrieve 96Boards IP address with the following command:

```
$ /sbin/ifconfig
```

**Commands(From host machine):**

```
$ scp hellomraa linaro@{ipaddress of 96boards}:.
$ ssh linaro@{ipaddress of 96boards}
$ ./hellomraa
```

> *Print out should read as follows:*

```
“Hello mraa”
“ Version: <some version info>”
“ Running on a <board type>”
```

**Congratulations** you have correctly installed your cross compiler and built your first cross compiled mraa library application.