ESP8266: Setting up native SDK's on Linux and exploring some applications

The information for installing and setting up the native SDK for the esp8266 from Expressif is readily available in the Internet:

With the native SDK we can develop our own firmware using the Espressif SDK and the C or C++ language. Then we can use directly the internal esp8266 processor, allowing to do interesting things in a more direct way, like implementing MQTT, driving hardware, and so on.

So why this post? Basically is to put into writing some of my own annotations to the issues that I had installing the SDK and the cross compiler tools on my Linux machines.

Starting up:
The original instructions that I've followed where these:
https://github.com/esp8266/esp8266-wiki/wiki/Toolchain
I installed the tool chains in two computers, both running Linux: an Arch Linux distribution and a Kubuntu 14.04 distribution (and time is up for this one because I'm moving it also to Arch Linux. All I'm missing is to install a SSD).

1st) Installed the required pre-requisites that are documented on the above link. On Arch Linux the package installation tool is **pacman** and probably the package names can be different. I didn't need to install anything on Arch, but on Kubuntu I needed to install the 64-bit pre-requisites.

2nd) Create the **/opt/Espressif** directory. The owner should already be your user, except if you have no permissions on the /opt directory.

3rd) Install the Xtensa Crosstool-NG that has the cross compiler and tools for the esp8266 processor:

```
cd /opt/Espressif
git clone -b lx106-g++
https://github.com/jcmvbkbc/crosstool-NG.git
cd crosstool-NG
./bootstrap && ./configure --prefix=`pwd` && make &&
make install
./ct-ng xtensa-lx106-elf
./ct-ng build
```

Some notes regarding this:

- We are going to clone the git branch **lx106-g++** that has the **C compiler** and the **C++ compiler**. There is another branch named **lx106** that only has the **C compiler.** There is some software out there like Ardunet that need the c++ compiler and so that is the reason why I've chosen this git branch.
- I've also changed the url for the git repository from **git://** to **https://** because in I'm behind a web proxy that needs authentication. Just make sure that the **http_proxy**, **https_proxy** and **ftp_proxy** variables are set. The ftp_proxy variable is needed because at the **ct-ng build** step will connect to a ftp server to retrieve files. One example for setting these environment variables: **export https_proxy=http://myproxyuser:myproxypass@proxyip:port.** If you have an internet direct connection, like at home, no need to set these variables.

- The **./ct-ng build** step will take a long time, around 30 minutes in one of my computers and 15 in another.
- The tool will connect to the ftp server on IP: http://208.118.235.20/ where are located the Gnu compiler sources.

At the end we need to add the path to the crosstools to our PATH variable.

`export PATH=/opt/Espressif/crosstool-NG/builds/xtensa-lx106-elf/bin/:$PATH`

You may want to add this line at your **.bashrc** file. In my case (I'm using KDE desktop, I've created a Konsole profile for this type of work and add the new settings to the PATH variable,)

Installing the Expressif SDK

The instructions on the above link are for the 0.9.3 version of the SDK. I've installed the most recent version at this time that was 0.9.4:

```
cd /opt/Espressif
wget -O esp_iot_sdk_v0.9.4_14_12_19.zip
https://github.com/esp8266/esp8266-
wiki/raw/master/sdk/esp_iot_sdk_v0.9.4_14_12_19.zip
unzip esp_iot_sdk_v0.9.4_14_12_19.zip
mv esp_iot_sdk_v0.9.4 ESP8266_SDK
mv License ESP8266_SDK/
cd /opt/Espressif/ESP8266_SDK
wget -O lib/libc.a
https://github.com/esp8266/esp8266-
wiki/raw/master/libs/libc.a
wget -O lib/libhal.a
https://github.com/esp8266/esp8266-
wiki/raw/master/libs/libhal.a
wget -O include.tgz
```

```
https://github.com/esp8266/esp8266-
wiki/raw/master/include.tgz
tar -xvzf include.tgz
```

Except the change of the SDK version, the instructions are the same as the original reference post.

Installing the esptool

The esptool will pick up the compiler output and extract/create the files needed for flashing the esp8266. The instructions for this step are the same as in the original reference post. In my case, I've got the tool from the sources and not installed the deb file:

```
cd /opt/Espressif
wget https://github.com/esp8266/esp8266-
wiki/raw/master/deb/src/esptool_0.0.2.orig.tar.gz
tar xvzf esptool_0.0.2.orig.tar.gz
cd esptool
```

Let's put the tool on the search path. In my case I've choose the Xtensa bin directory. (Edit: it should be put at /usr/bin. See below at building the blinky example)

```
cd /opt/Expressif/esptool
chmod +w /opt/Espressif/crosstool-NG/builds/xtensa-
lx106-elf/bin
ln -s /opt/Espressif/esptool/esptool ../crosstool-
NG/builds/xtensa-lx106-elf/bin
sudo cp /opt/Espressif/esptool/esptool /usr/bin
```

Installing the firmware uploader tool esptool-py

```
cd /opt/Espressif
git clone https://github.com/themadinventor/esptool
esptool-py
chmod +w /opt/Espressif/crosstool-NG/builds/xtensa-
lx106-elf/bin
ln -s /opt/Espressif/esptool-py/esptool.py
crosstool-NG/builds/xtensa-lx106-elf/bin/
```

With this tool we can upload the generated firmware into the esp8266.

Building
So now we are ready and we can follow the instructions here: https://github.com/esp8266/esp8266-wiki/wiki/Building to starting building things.

There are some code examples, one of them named **blinky** but the make file assumes that the **esptool** is located **/usr/bin.** This is because I've not installed the esptool through the deb package. So we can copy the esptool command to the /usr/bin directory: **sudo cp /opt/Espressif/esptool/esptool /usr/bin.**

We can now build the AT demo and the IoT demo. Just follow the instructions. For example for the IoT example just do the following (Make sure that the path is set: export PATH=/opt/Espressif/crosstool-NG/builds/xtensa-lx106-elf/bin/:$PATH):

```
cd /opt/Espressif/ESP8266_SDK
sed -i -e 's/xt-ar/xtensa-lx106-elf-ar/' -e 's/xt-
xcc/xtensa-lx106-elf-gcc/' -e 's/xt-objcopy/xtensa-
lx106-elf-objcopy/' Makefile
mv examples/IoT_Demo .
cd /opt/Espressif/ESP8266_SDK/IoT_Demo
```

```
make
cd .output/eagle/debug/image
esptool -eo eagle.app.v6.out -bo
eagle.app.v6.flash.bin -bs .text -bs .data -bs
.rodata -bc -ec
xtensa-lx106-elf-objcopy --only-section .irom0.text
-O binary eagle.app.v6.out
eagle.app.v6.irom0text.bin
```

Uploading:
For uploading just follow the instructions here: https://github.com/esp8266/esp8266-wiki/wiki/Uploading but basically is using the esptool.py to upload the firmware to the esp8266. Just make sure tha GPIO0 is grounded at boot time.

The IoT RTOS based example:
There is a beta SDK for the esp8266 based on the freeRtos according to this: https://github.com/espressif.

To use this SDK (Make sure that the PATH environment variable is correctly defined):

```
cd /opt/Expressif
git clone
https://github.com/espressif/esp_iot_rtos_sdk.git
git clone
https://github.com/espressif/esp_iot_rtos_sdk_lib.g
it
cp esp_iot_rtos_sdk_lib/lib/*  esp_iot_rtos_sdk/lib
cd esp_iot_rtos_sdk

make
```

We should have now the IoT RTOS main image ready. To generate the firmware files:

```
cd app/.output/eagle/debug/image/
esptool -eo eagle.app.v6.out -bo
eagle.app.v6.flash.bin -bs .text -bs .data -bs
.rodata -bc -ec
xtensa-lx106-elf-objcopy --only-section .irom0.text
-O binary eagle.app.v6.out
eagle.app.v6.irom0text.bin
```

And we can flash it:

`/opt/Espressif/esptool-py/esptool.py --port /dev/ttyUSB0 write_flash 0x00000 eagle.app.v6.flash.bin 0x40000 eagle.app.v6.irom0text.bin`

The default baud rate for the IoT example is 74880…

Exploring some application:

Now that we have all the tools for developing natively we can check out some applications:

- esp-mqtt – A native MQTT client for the esp8266
- Ardunet – A work in progress that mimics the Arduino way of making applications on the esp8266.

The download links are: