Lab 1

— *Getting started with the* ESP32 —

This first lab concentrates on getting you familiarized with aspects related to the `ESP32`, the development board that we will be using during these series of labs. You will initially install the development environment and then design small programs that you can flash into the development board in order to run them.
For those who wish to work with an integrated development environment (IDE) such as Eclipse or PlatformIO, please refer to the corresponding documentation. In these labs we will only need a simple text editor ; Vim, emacs, etc. and a terminal.
N.B

- All bugs that you will encounter should be filled as issues in this repository `https://github.com/efrei-paris-sud/Lab-One/issues` ;
- The more non-trivial issues you fill and more generally the more acitve you are in GitHub, the more you get good appreciation for your final mark from us ;
- This being said, before submitting a bug, try to resolve it by "google"-ing or "stackoverflow"-ing it and don't hesitate to resolve your own or other's issues ;
- You will find the format for issuing a bug here `https://github.com/efrei-paris-sud/Lab-One/issues/1`.

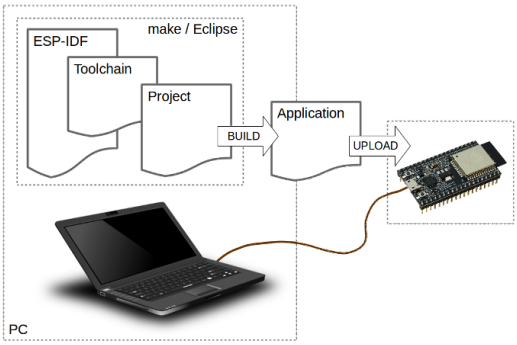

FIGURE 1.1 – Development of applications for ESP32. Source : `https://docs.espressif.com/projects/esp-idf/en/latest/get-started/`

1.1 Set up cross compilation environment

1.1.1 Installing dependencies

For Ubuntu and Debian :

```
sudo apt-get install gcc git wget make libncurses-dev flex bison \
     gperf python python-pip python-setuptools python-serial \
     python-cryptography python-future python-pyparsing
```

1.1.2 Installing Xtensa toolchain

Download tarball

You can download the Xtensa toolchain from the following link : `https://dl.espressif.com/doc/esp-idf/latest/get-started/linux-setup.html#toolchain-setup`

Linux 64-bit :

```
xtensa-esp32-elf-linux64-1.22.0-75-gbaf03c2-5.2.0.tar.gz
```

Linux 32-bit :

```
xtensa-esp32-elf-linux32-1.22.0-75-gbaf03c2-5.2.0.tar.gz
```

Extraction

Extract the contents of the previously downloaded tarball into `/opt` (recommended folder) :

```
sudo tar -xvf xtensa-esp32-elf-linux64-1.22.0-75-gbaf03c2-5.2.0.tar.gz \
        -C /opt
```

Make the toolchain available across terminal sessions

Now, in order to make the toolchain components available across terminal sessions, you need to add the path to the location of the toolchain into your `~/.bashrc` file. Add the following command to the end of your `~/.bashrc`, modify the path according to the specific location where you extracted the tarball previously :

```
export PATH=$PATH:/opt/xtensa-esp32-elf/bin
```

1.1.3 ESP IoT Developpement Platform (ESP-IDF)

ESP-IDF is a framework which ease and speed up building applications of the internet of things. The framework offers a rich API that allow developers to exploit the great capabilities of the ESP32 development board, such as, WiFi, Bluethooth Low Energy (BLE), energy management, etc.

Getting the ESP-IDF framework

In order to get the ESP-IDF framework, you need to clone the git repository using the following command. This repository corresponds to a fork of the *Espressif*'s official repository which is dedicated specifically to this series of labs. Do not forget the `--recursive` option.

```
git clone --recursive https://github.com/efrei-paris-sud/esp-idf.git
```

Setup the IDF_PATH

Similar to the toolchain setup done previously, you need to make the location of the ESP-IDF framework accessible from your terminal sessions. For this, add the following command to the end of your `~/.bashrc`. Adapt the path to the location where you cloned the repository.

```
export IDF_PATH=/path/to/the/location/of/your/repository
```

Testing the installation

In order to verify that the installation was carried correctly, we will try to compile a first program for the `ESP32`. The folder `examples` inside the ESP-IDF framework contains some examples that you can compile and test. These programs come with a `Makefile`; you only need to issue the command `make` in your terminal.

1.1.4 Communication with the development board

Now that the environment is finally set up properly and that you are able to compile one of the examples provided in the `examples` folder, you have to communicate with the development board in order to upload the binary image of your program into the flash memory of the `ESP32`. Communication with the development board is carried out via serial communication, *i.e.* USB peripheral which is specified by `/dev/ttyUSB`. It follows that so as to use this peripheral,

you need to have read and write permissions on that file, which you don't have by default. To cope with this, you need to add the current user to the `dialout` group by means of the following command :

```
sudo usermod -a -G dialout $USER
```

Where `$USER` corresponds to the current user.

Check if you, effectively, belong to the `dialout` group using the command `groups $USER`. Hang on ! when you add a given user to a group, these changes don't take effect immediately, you have to logout then login, or launch a new terminal session with `su $USER`.

1.1.5 Serial bootloader

The serial bootloader is the tool that allows you to upload your executables into your development board's flash memory. *Espressif*, the company that develops and maintains the `ESP32` dev. boards makes available the `esptool.py`, which is written in python, and distributed under the GPLv2 licence. This tool allows you to communicate with the ROM of the `ESP32` bootloader. In order to install it, use the Python Package Index (PyPI) :

```
pip install esptool
```

Upload your first program

By issuing `make flash` in one of the provided examples, you will upload the generated binaries into the development board. Check that everything works fine, *i.e.* no error messages are displayed in the `esptool.py` output.

Interacting with the development board

You have the ability to interact with the development board via serial communication. To start the serial monitor, just issue `make monitor`. The standard output as well as the starndard input of the programs being run on the development board are redirected to the serial monitor, thus, when for eg. a call to `printf` in your program is executed, the message will be displayed by the serial monitor on your terminal. The same principle applies for the standard input.

The following commands are usefull to handle the serial monitor properly :

- `Ctrl-]` to stop the monitor ;
- `Ctrl-T Ctrl-H` will display a help menu with all other keyboard shortcuts ;
- Any other key apart from `Ctrl-]` and `Ctrl-T` is sent through the serial port ;

1.1 EXERCICES

1. Read the previously flashed programs and try to infer a typical structure for these kind of programs. More specifically, take a look at the included headers and try to find their location in the ESP-IDF source tree. Nothing concrete is asked in this exercise but feeling free to explore the various components of the ESP-IDF.

2. In order to have data comming from the development board, the program being run on it has to be designed in that sense. Try to flash the example inside `examples/get-started/hello_world` into the development board. This example return information concerning the various features embedded in the `ESP32` (have a look into the source code to know how the program works in details).
By inspiring you with the `hello_world` example, extend this program so as to allow a user to interact with the development board via the serial monitor in order to display, according to the user's commands [1], the features embedded in the `ESP32`.

3. By referring to the `esp_system.h (link)` header file, extend the previous program in order to display more information about the development board. Extend, at the same time the commands formats you've defined.

1. Commands format that you have also to define, it has not to be necessarily complex.