Lab 2

General Purpose Input/Output (GPIO)

The goal of this second lab (and the third one) is to get you familiarized with the various APIs exposed by the ESP-IDF framework. Firstly, you will discover the pinout of the development board and program with the GPIOs API. After that, you will discover by yourself the Serial Peripheral Interface (SPI) protocol used for transmission. The main goal is to correctly connect the BME280 sensor to the ESP32 via SPI and get measures from it. Another goal of the lab is to put you in the context of reading documentation from different sources in order to get devices communicate with each other.

Reminder

- All bugs that you will encounter should be filled as issues in this repository `https://github.com/efrei-paris-sud/Lab-Two/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`.

2.1 Introduction

The ESP32 chip features 40 physical GPIO pads. Some GPIO pads cannot be used or do not have the corresponding pin on the chip package(refer to technical reference manual). Each pad can be used as a general purpose I/O or can be connected to an internal peripheral signal.

Note that GPIO6-11 are usually used for SPI flash. GPIO34-39 can only be set as input mode and do not have software pullup or pulldown functions [1].

1. Take a look at Section Pin Description of the ESP32 datasheet p.7 (link)

2.2 GPIO API [2].

GPIO set direction

Configure GPIO direction,such as `GPIO_MODE_INPUT`, `GPIO_MODE_OUTPUT`, `GPIO_MODE_INPUT_OUTPUT`, etc.

```
esp_err_t gpio_set_direction(gpio_num_t gpio_num, gpio_mode_t mode)
```

Return
- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO error

Parameters
- `gpio_num` : Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16) ;
- `mode` : GPIO direction

GPIO set output level

```
esp_err_t gpio_set_level(gpio_num_t gpio_num, uint32_t level)
```

Return
- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO number error

Parameters
- `gpio_num` : GPIO number. If you want to set the output level of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16) ;
- level : Output level. 0 : low ; 1 : high

GPIO get input level

```
int gpio_get_level(gpio_num_t gpio_num)
```

Return
- 0 the GPIO input level is 0
- 1 the GPIO input level is 1

Parameters
- `gpio_num` : GPIO number. If you want to get the logic level of e.g. pin GPIO16, `gpio_num` should be `GPIO_NUM_16` (16) ;

2.3 Breadboard

From now on, you will be using a breadboard to hookup all the wires. Figure 2.1 depicts how energy flows through a typical breadboard. More precisely, energy in blocks A and D is distributed horizontally while in blocks B and C, it is distributed vertically. Note that all blocks are independant.

2. For a complete API reference, please refer to the official documentation of the ESP-IDF `https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/gpio.html`(link)

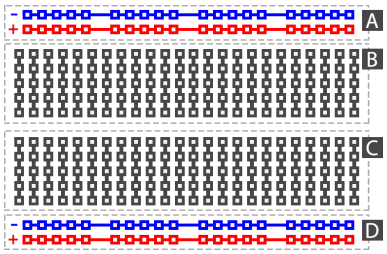

FIGURE 2.1 – Basic breadboard layout. Source : `https://www.tweaking4all.com/wp-content/uploads/2013/12/basic_breadboard_layout.png`

2.1 EXERCICES

1. Create a new project by following this structure (`main.c` containing all the stuff you are implementing) :

```
- src/
        - main.c
        - component.mk
- Makefile
```

   Note that, the ESP-IDF has a particular build system which is based on Makefiles. The file `component.mk`, which is by the way empty, is used to specify to the build system that all files in this subdirectory have to be included into the project and thus compiled using the toolchain. [3]

2. Write a function `blink_task` which turns on then off the integrated LED of the ESP32 indifinitely. Note that the integrated LED is connected to `GPIO_2`. You can make the function call `vTaskDelay(1000/portTICK_RATE_MS);` in your program so as to add a delay of 1000ms between pull-up and pull-down. Do not forget to include `freertos/FreeRTOS.h` as well as `freertos/task.h` headers.
3. The entry point of ESP-IDF programs is not the usual `main` but `app_main`. Define an entry point for your program and launch the previously defined function in a `freeRTOS` task [4].
4. Make the necessary hook-ups with many LEDs on the breadboard in order to make light effects of you choice. Modify you program to operate these light effects.

3. If you are interested to know more about the ESP-IDF build system, you can dig into the contents of `esp-idf/make/` folder.

4. The tiny microcontrollers that you are handling run a true operating system which features multithreading ! We will discover freeRTOS in more details during the next labs.

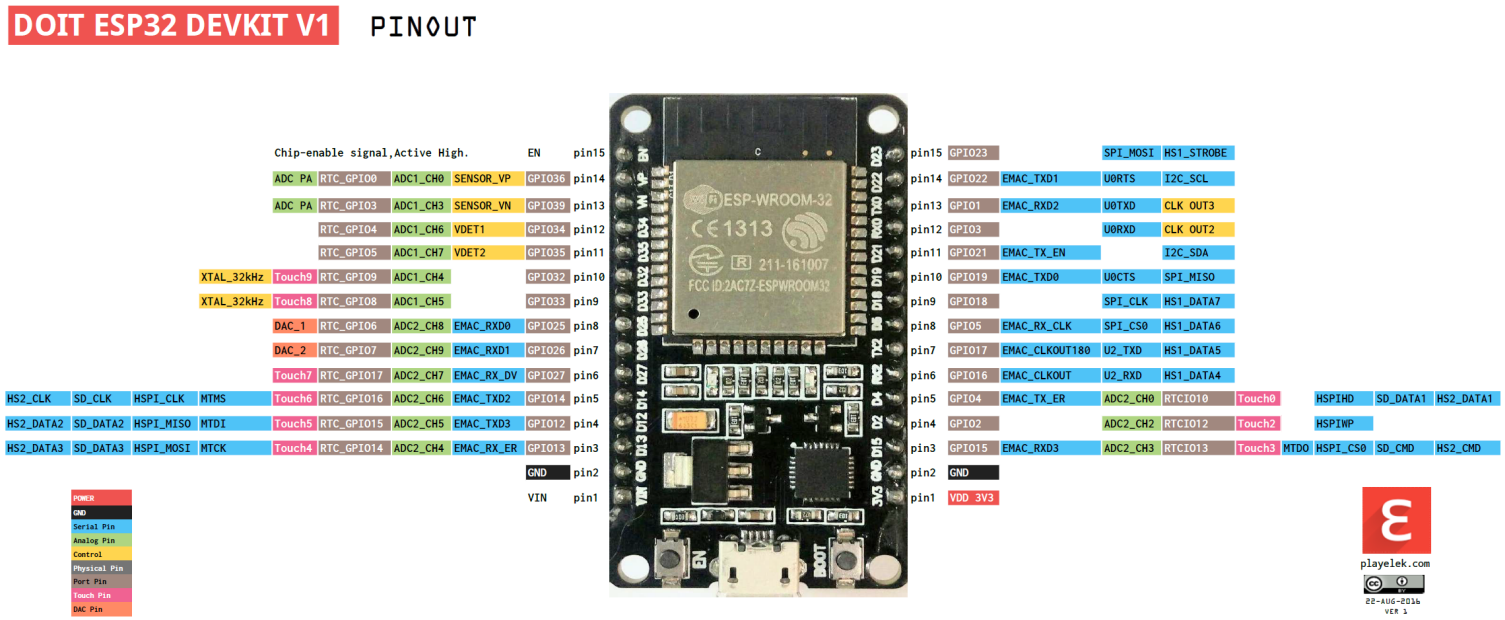

FIGURE 2.2 – ESP32 development board pinout.