Author

Simón Pedro Aulet
simon.aulet@gmail.com

About the Report

This is a report on the project "Initial Configuration System for ESP32 Devices via Wi-Fi". It is a prototype of a machine configuration interface device using an ESP32 development board as a base.
Code, connections, and everything needed for this project can be found at https://github.com/SimonAulet/Configurador-ACySE

Operating Principle

Specific pins on the development board are connected to actuators, LEDs, sensors, etc. of the machine, according to the application. The state of each pin determines a type of configuration. For example; in the case of an oven, pin_13 high could mean interior light on by default.
The user connects to a WiFi network emitted by the microcontroller, which redirects all DNS requests to a captive portal; a web page stored in the device's memory. This way, when the user connects to the network, they are directed to a web interface from which they can view and modify the device's configuration. Then, when saved, the configuration is written to flash memory and persists once the user disconnects and even when the device restarts.

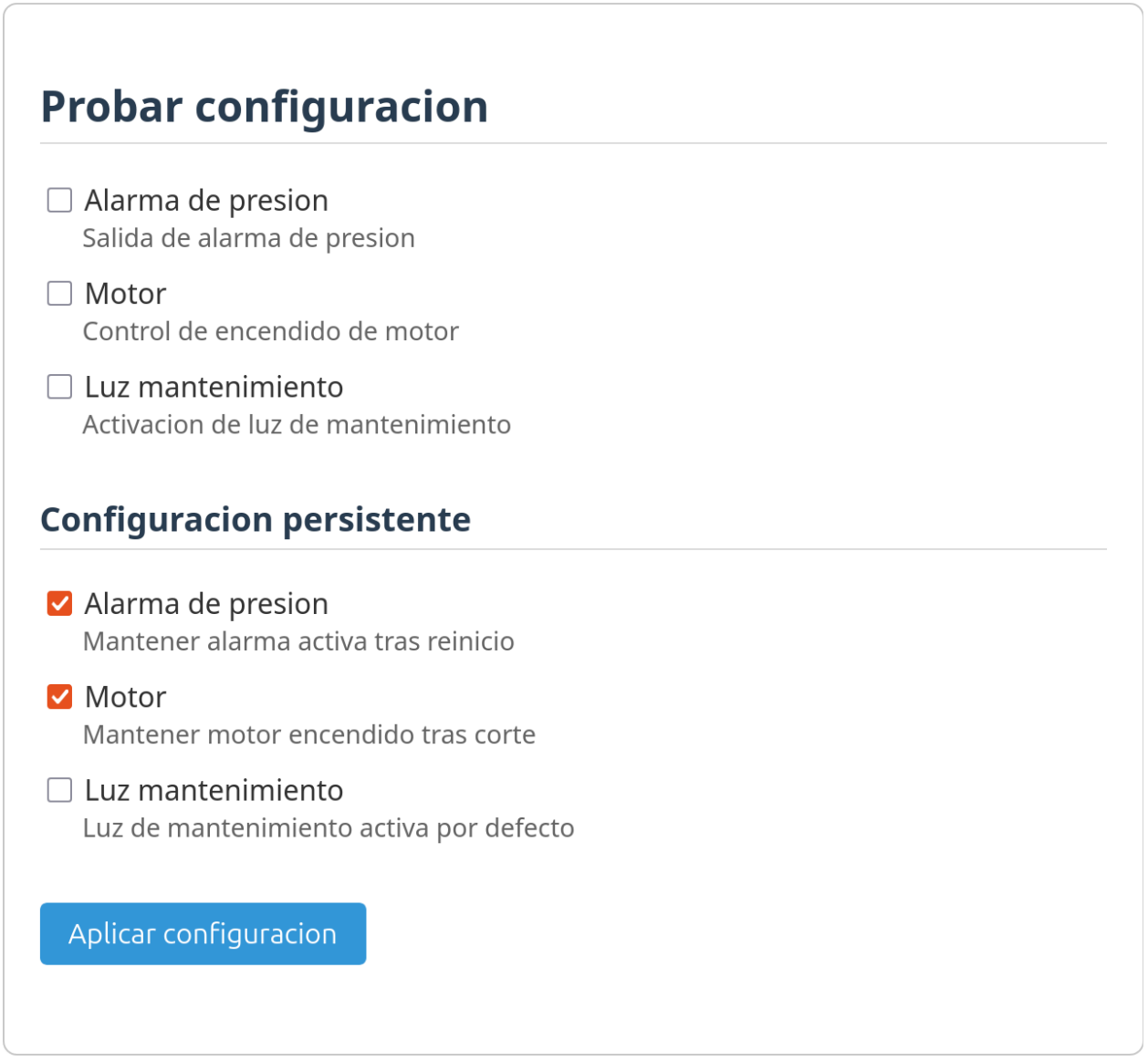
Motivation

Configuring certain devices is often limited or inconvenient to do. It is very common for machines to offer a 16x2 LCD with 3 buttons through which the user has to, manual in hand, make very complex configurations. This generates a propensity for errors and much frustration when configuring machines. To name a few examples, this happens in commercial refrigerators, cold rooms, ovens, dehydrators, boilers, etc.

- **Placing a large touch display to configure the device**: Not always feasible; mainly due to the costs of installing a display under these conditions. A touch display is very prone to breaking and does not survive in all conditions (outdoor use, inside kitchens with high temperature and flying flour, motor vibrations in certain machines, etc.) which complicates things greatly.
- **Designing a custom app**: There was a big trend, mainly in appliances, to design apps to control them. The problem with these apps is, mainly, the maintenance cost. Maintaining an app over the years for different versions of Android and iOS and adapting it to the constant changes of each ecosystem is extremely expensive, which leads to these types of applications typically being very neglected and prone to failures. Additionally, the general public is tired of having to install an app for everything, losing the simplicity of operating a machine.

- **Limiting options to pre-established configurations**: Usually the most used and although it works, it limits the potential of machines.

With this in mind, my solution proposes that configuration be done through the web page served by the microcontroller using simple and reliable HTTP protocols. The configuration is not accessed every time the device is to be used, only when something is to be modified. For example, configuring an external probe added to a boiler to increase its efficiency. Through the web interface, help and instructions about the device's use and the functionality of configurations can even be provided.



Scope

This system is designed for domestic or industrial appliances, operated by people with medium/high technical knowledge, where several parameters need to be modified clearly.

Operation

The following details the relevant functions of the code:

1. Starting from the captive portal example provided by Espressif (https://github.com/espressif/esp-idf/tree/v5.4.1/examples/protocols/http_server/captive_portal) which already has robust and well-maintained DNS redirection configured.
2. A data type representing a configuration item is generated using `key - value`:

```
typedef struct {
    char key[16];
    int32_t val;
} pin_config_T;
```

3. A default pin configuration is generated. If nothing is stored in flash, these values will be used. In this case, simply two pins low and one high were used. Note that this configuration will only be considered when the permanent flash is erased. Then, even if the device is turned off, the last saved configuration is used, not the default.

```
   const pin_config_T defaults[] = {
    {"pin_13", 0},
    {"pin_12", 0},
    {"pin_14", 1}};
```

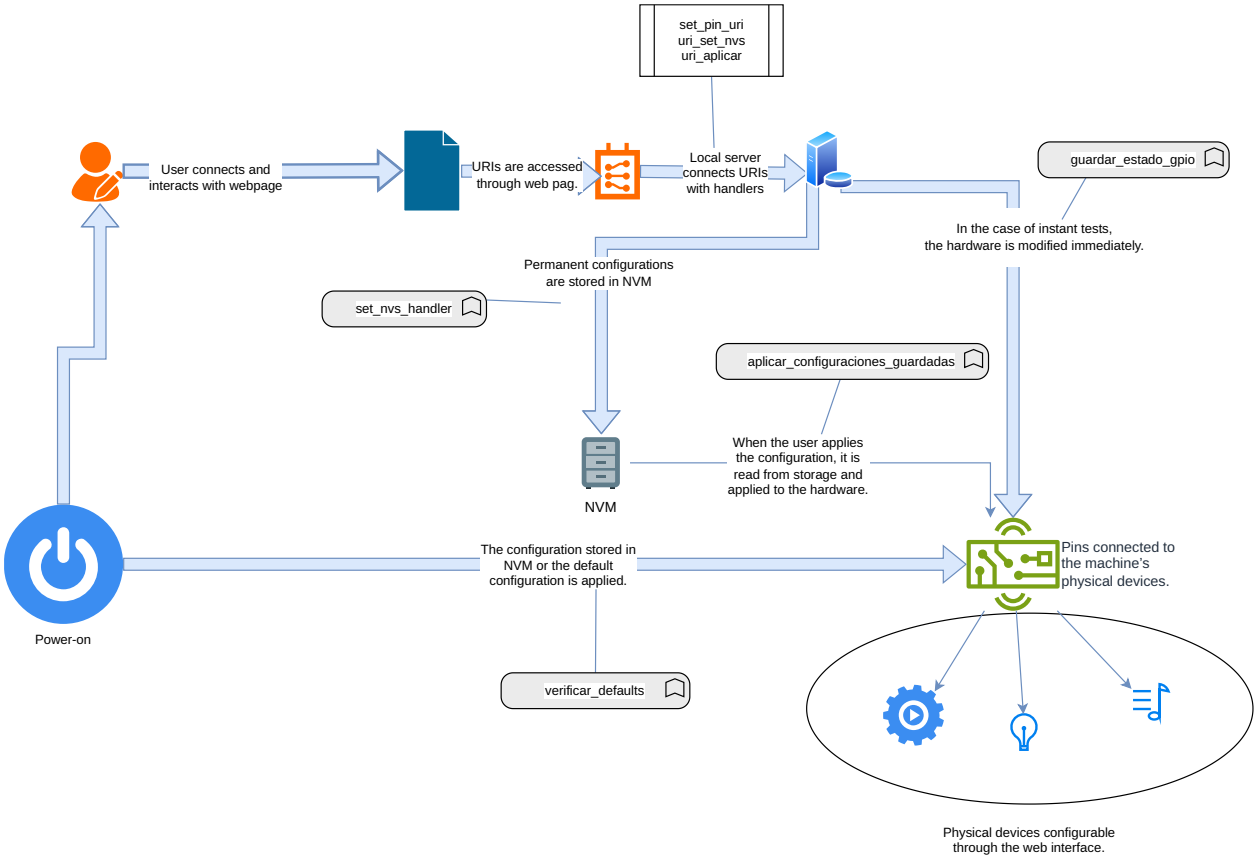
4. The following three functions are generated (complete bodies are in the code):
    1. `void verificar_defaults(const pin_config_T *defaults, int cantidad);`
    Verifies that all configuration options exist in NVM. If any does not exist, it creates it and assigns the default value.
    2. `void aplicar_configuraciones_guardadas(const pin_config_T *defaults, int cantidad);`
    Applies the configurations stored in NVM to the physical pins of the device. This is done on startup or when configurations are modified.
    3. `void guardar_estado_gpio(int pin, int value);`
    Assigns a specific state to a specific GPIO ignoring what is permanently stored. The purpose of this function is to test machine functionalities without affecting its configuration.
5. Since the interface will be web-based, handlers are generated to connect the web server with the microcontroller's internal functions:
    1. `esp_err_t set_pin_handler(httpd_req_t *req)`
    Handler that simply calls `guardar_estado_gpio` to store a state.
    2. `esp_err_t set_nvs_handler(httpd_req_t *req)`
    This handler contains the function that stores a `key` and its respective `value` in persistent memory. It does not affect the configuration, only the memory.
    3. `esp_err_t aplicar_config_handler(httpd_req_t *req)`
    Applies the configuration stored in NVM by calling `aplicar_configuraciones_guardadas`.
6. With the handlers defined, the URIs (of type `httpd_uri_t`) are defined that will link the actions performed on the web with the previously defined handlers and functions. These are: set_pin_uri (links to set_pin_handler), uri_set_nvs (links to set_nvs_handler), and uri_aplicar (links to

aplicar_config_handler) and then they are registered within the `start_webserver` function to take effect.

7. With the functions and connections between functions and the web defined, the web page (root.html) is designed, which is started by default by the web server. Now all traffic passing through the WiFi network emitted by the microcontroller is redirected to this page through which the device is configured.

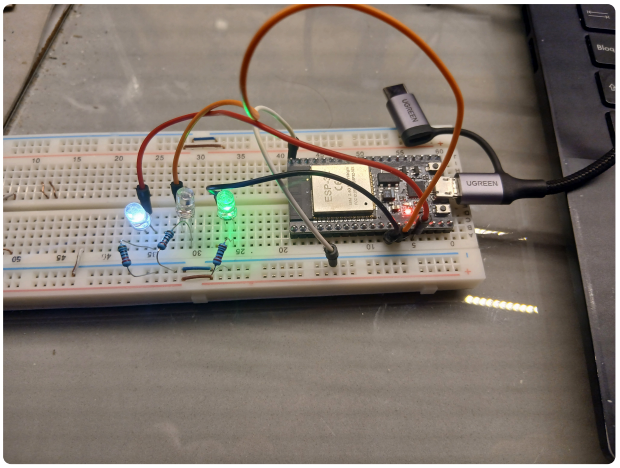
Usage Flow Diagram

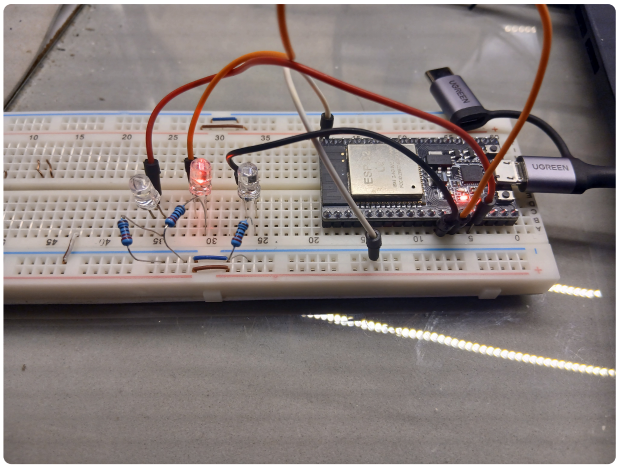
The following diagram shows the normal use of the device. Note that after startup, the device loads the stored configuration (or defaults if nothing is stored) and only when the user connects are modifications made to the configuration.

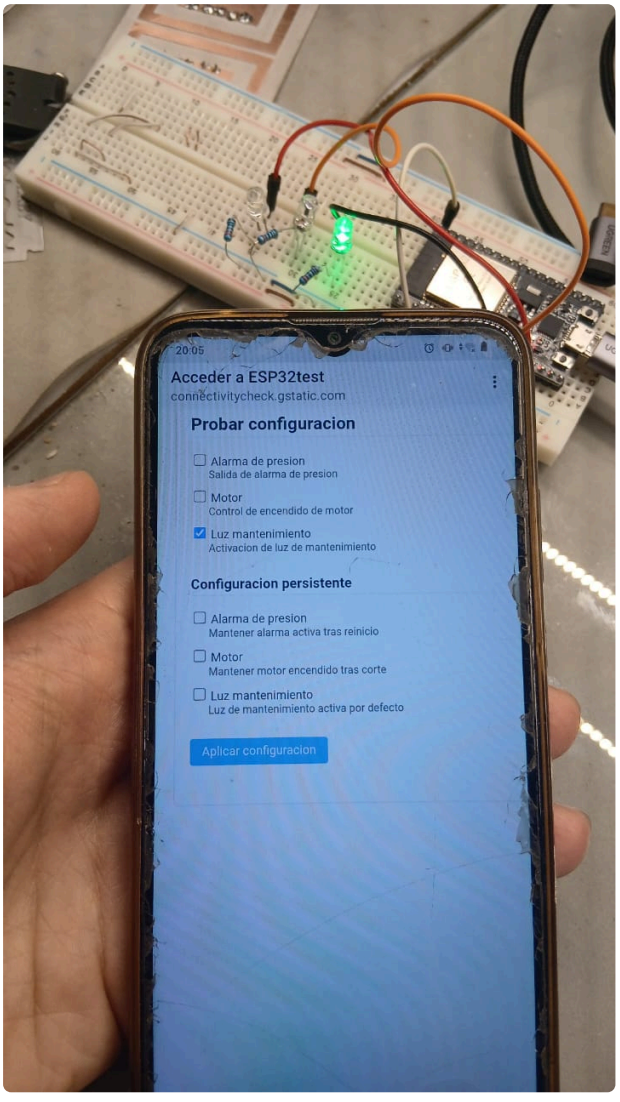


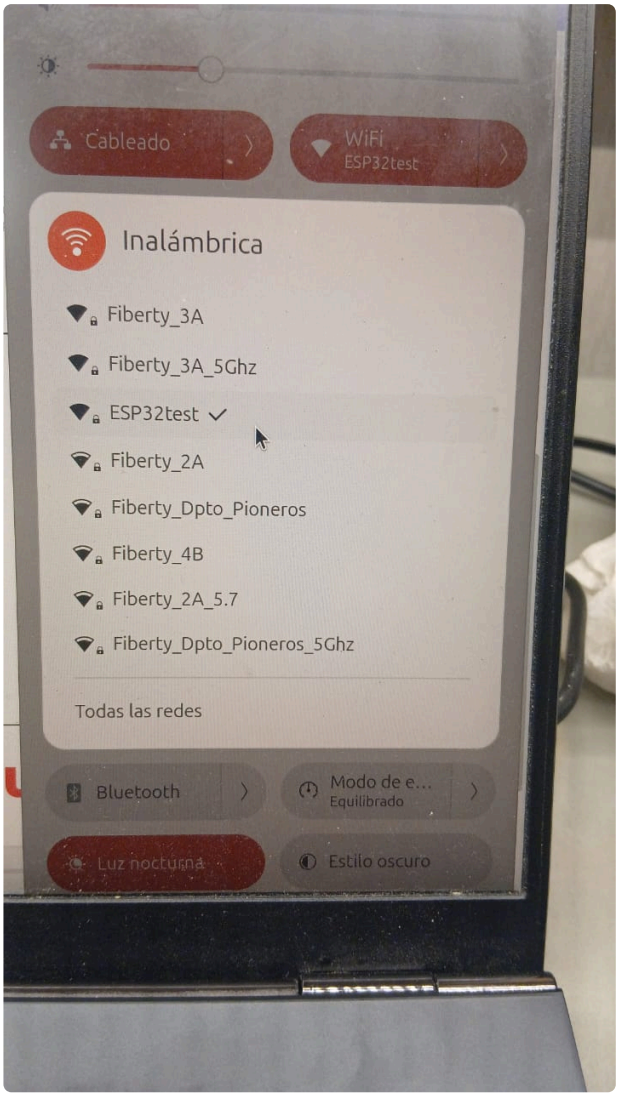
Usage Examples

Being an interactive web interface that stores information permanently, it is difficult to show its operation with images. These are photos of the development board on a protoboard with two different configurations applied. Each LED represents a physical configuration to apply to a real machine.

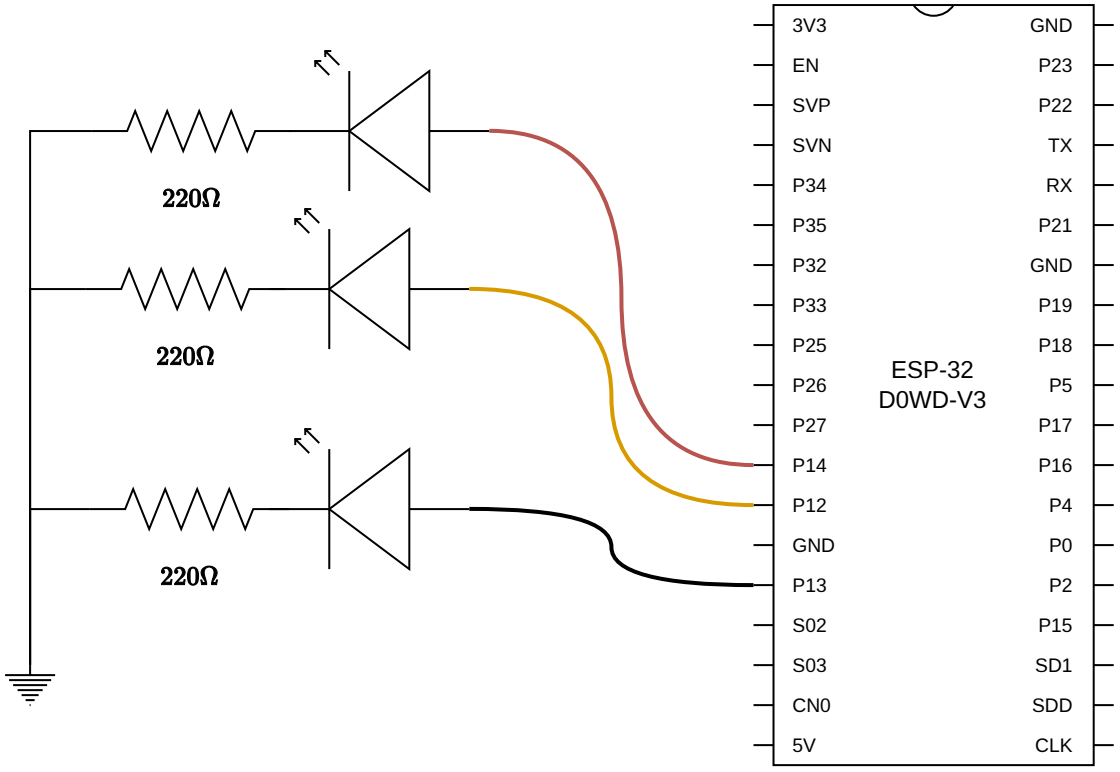








Then the connection diagram to test this same example with a 38-pin ESP32 board:



The complete code, this report, and the presentation poster can be found in the project's GitHub:
https://github.com/SimonAulet/Configurador-ACySE