Upgrade of the humidity extraction system: optimization with Arduino Nano and DHT22-AM2302 for advanced energy management control.

Let's get back to the topic of DIY electronics in this new article on officinasottocasa.it, completely revisiting our humidity extraction system that you read and appreciated in our previous article on the "Solar-Powered Humidity Extractor," updating the hygrostat and the energy management system using a temperature/humidity sensor with DHT22-AM2302 connected to an Arduino Nano.

The reason we decided to revise the system is related to the solar panel's inability to provide sufficient energy to the battery and thus ensure the storage of the necessary energy to operate the extraction fans in the colder and less illuminated months of the year. Moreover, these are also the rainiest months with more humidity problems.

An alternative to upgrading the entire system could have been expanding the photovoltaic panel and accumulator capacity, but the current placement does not allow for it.

We have therefore decided to activate, when necessary, an additional charging source using the domestic power grid. We could have simply inserted a charge maintainer purchased on Amazon for a few tens of euros, but it wouldn't have been in the style of officinasottocasaDIY.

Hardware Components

Let's start with the components of the extraction system, which, as in the previous version, are as follows:

- Lead-acid battery 12V 7.2 Ah
- Voltage regulator for solar panels 20A
- Fan 12V 2W
- Solar panel 12V 20W

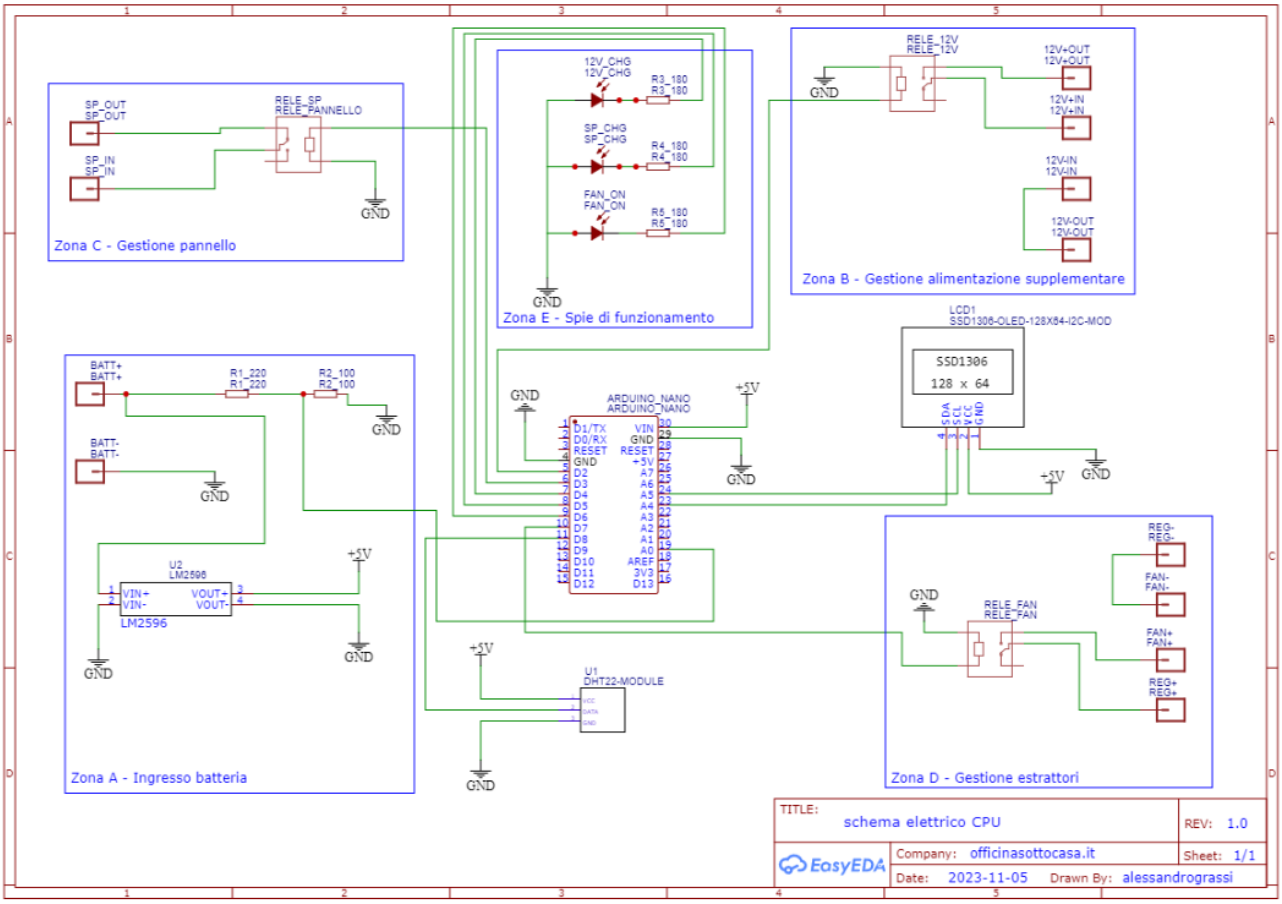
Now, let's move on to the components of the CPU:

- Elegoo Nano
- OLED SSD1306 128X64 Screen
- DHT22-AM2302 Module
- Relay SRD-05VDC-SL-C
- Variable Step-down with LM2596

We designed the CPU using the circuit designer available on this website link and created the PCB using the connected service link. The designer allows automatic generation of the gerber file for the PCB and sending it to the production site.

Electrical Diagram CPU

Let's begin with the overview:



Electrical Diagram of the CPU

To facilitate a detailed analysis of the diagram and maintain a coherent order, we have divided the components into specific zones, each corresponding to its function.

Zone A – Battery Input

In this zone, we have the contacts for connecting the battery, the voltage divider, and the step-down DC-DC converter for powering the board. The two functions of this group are as follows:

Powering our circuit through a 5V step-down with the integrated LM2596.

Connecting the battery to the voltage divider to check its charge status and activate additional charging through the grid.

The LM2596 is a popular integrated switching voltage regulator. Equipped with high performance, it is widely used to convert DC voltages, ensuring optimal stability and energy efficiency. Suitable for power applications in various industries, it provides a reliable and compact solution for voltage control. We practically use it in every project. In particular, you saw it in this article: "GSM Alarm for Low Battery – Part 1: Hardware."

The voltage divider is an essential electronic circuit that divides the voltage in a circuit in proportion to the values of the resistors used. This versatile component finds common applications in electronic design, allowing precise control of voltage in various contexts, from sensors to amplifiers. For further details, we refer to Raffaele Ilardo's website, which you can find in the references.

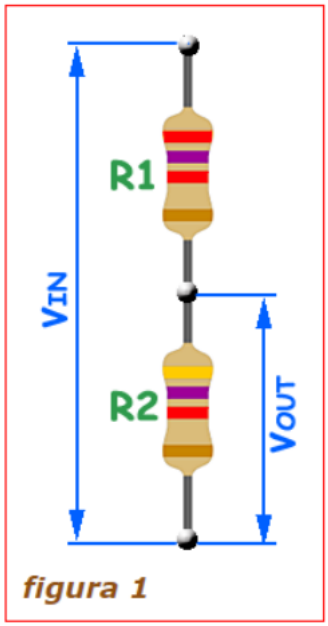


figura 1

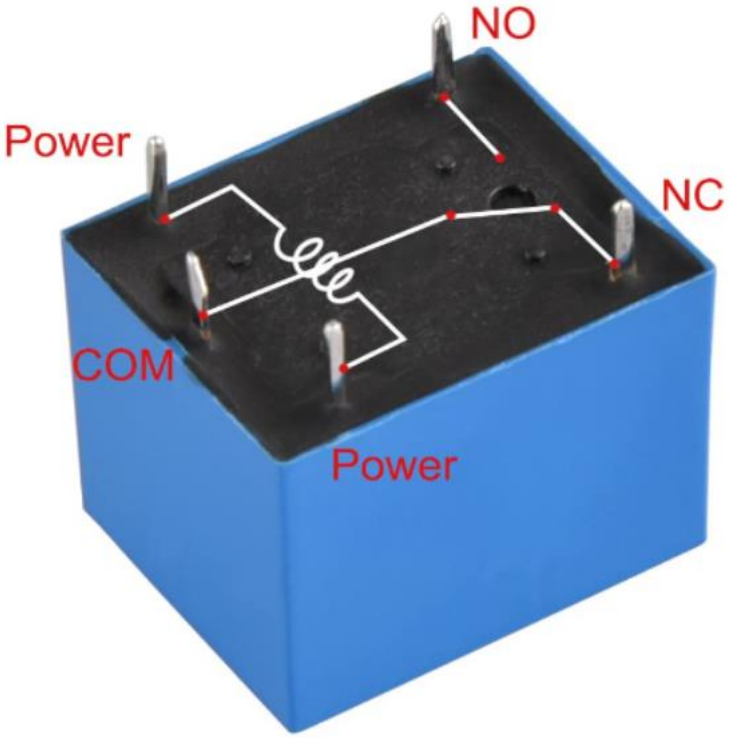
In this project, we use it to lower the battery voltage to a level compatible with the analog input of the Arduino board. In particular, we have inserted a voltage divider (R1=220KΩ and R2=100KΩ) that causes a voltage drop from the 12V input from the battery (Vin) to 3.75V (Vout) across R2.

Zone B – Management of Additional Power Supply

In this group, we have the contacts for the power supply from the electrical grid and the control relay managed by the microcontroller.

The SRD relay is a versatile electronic component that controls the opening and closing of electrical circuits. Its reliability and switching capability make it ideal for controlling loads in various electronic devices, ensuring efficient management of electrical connections and increased durability.

In this project, we use SRD relays with a 5V primary coil as they are controlled through the digital ports of the Arduino board.



You will notice that, with the use of the relay, we have completely isolated our circuit from the 12V charging system coming from the electrical grid. The relay operates on the positive pole, while the negative pole has only two mounting holes for wire soldering, as it is not in common with the ground of our integrated circuit.

Zone C – Photovoltaic Panel Management

In this group, we connect one pole of the photovoltaic panel to another relay. The purpose of this connection is to exclude the panel when the system is powered by the electrical grid. This way, we avoid overlaps and use exclusive sources without interference between the panel and the power supply.

Once again, the relay guarantees the isolation of the panel current from that of the circuit.

Zone D – Extractors Management

Here is where we manage the activation of the load, which consists of fans for extracting humid air outside. The current to operate the fans comes from the charge controller that manages the panel and charges the battery.

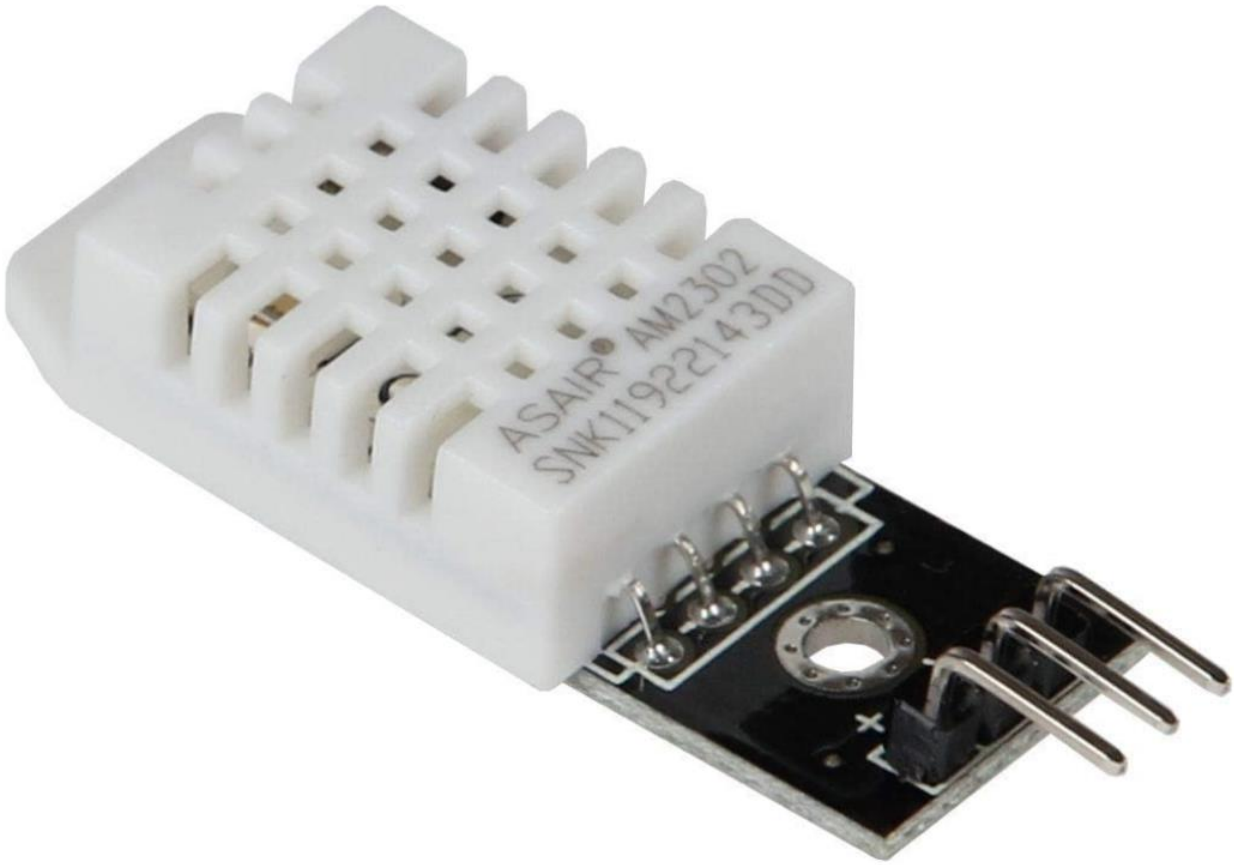
Zone E – Operating Indicators

We have inserted three LEDs that serve as indicators of the relay status. The LEDs, controlled by the microcontroller's software, will provide us with information about the relay's excitation status. The excitation status of a relay indicates whether it is activated or deactivated. When excited, the relay closes the contacts, allowing the passage of current. Upon deactivation, it interrupts the connection.

DHT22-AM2302 Module

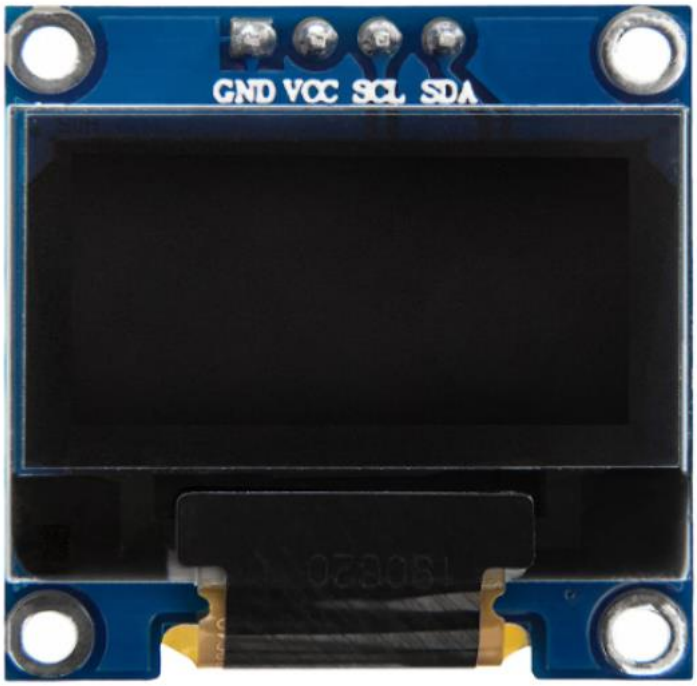
The DHT22-AM2302 module is a high-performance temperature and humidity sensor, renowned for its accuracy in environmental measurements. Characterized by a compact design and a simplified interface, it

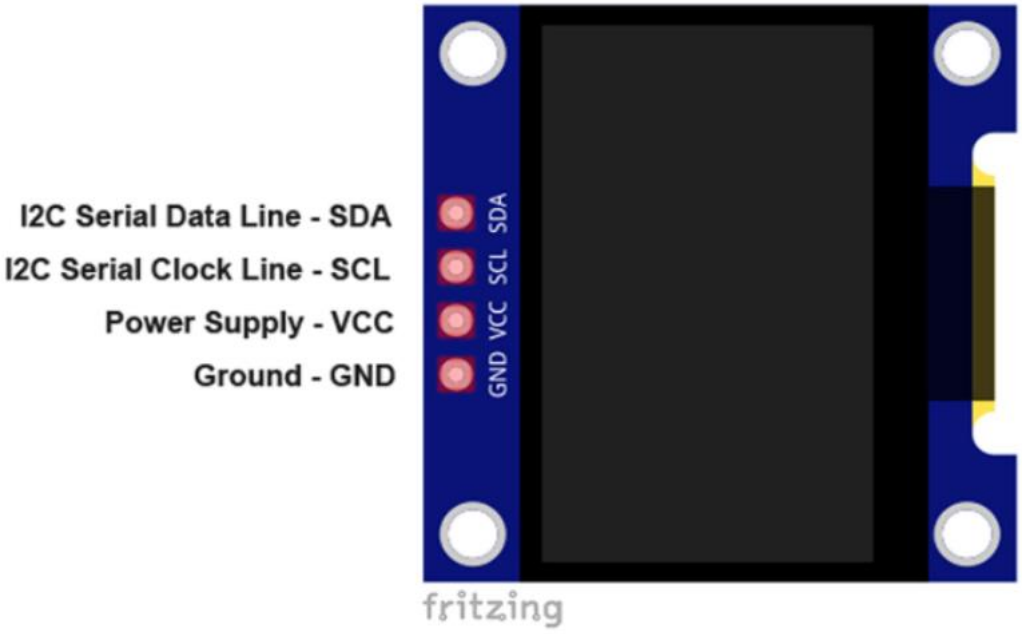
is commonly employed in DIY projects and industrial settings. Its integration with Arduino Nano allows us to monitor the humidity level in the air and, based on the parameters defined in the software, activate the extractors.



OLED Screen 128X64

To have all the operating information and any alerts, we have included an OLED display, an advanced electronic component with a resolution of 128x64 pixels. Thanks to OLED technology, it provides superior sharpness and contrast, making it ideal for displaying text and graphics. The controller is an SSD1306 with an I2C interface, which makes it suitable for this type of project.

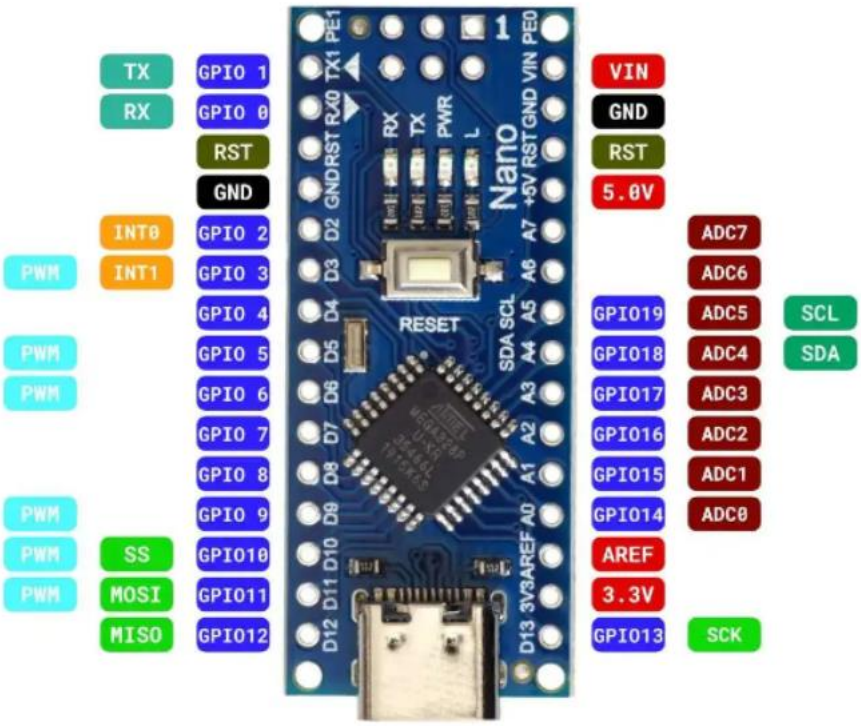




Arduino Nano

Now let's get to the beating heart of the project. The microcontroller hosting the software we've created, governing all the various components.

This is a Nano clone, not a genuine Arduino, although it's practically identical in terms of pinout, operation, and programming to its original counterpart.



They share the same ATMEGA328P chip. The only difference between the originals and clones is the USB-Serial converter. Clones typically use the CH340G chip, and drivers for this chip are not included in the Arduino IDE installation. Therefore, they need to be installed separately. In this article link, you'll find information and links to download and install drivers for the CH340G.

For its usage, we'll discuss it more in terms of software in the continuation of the article.

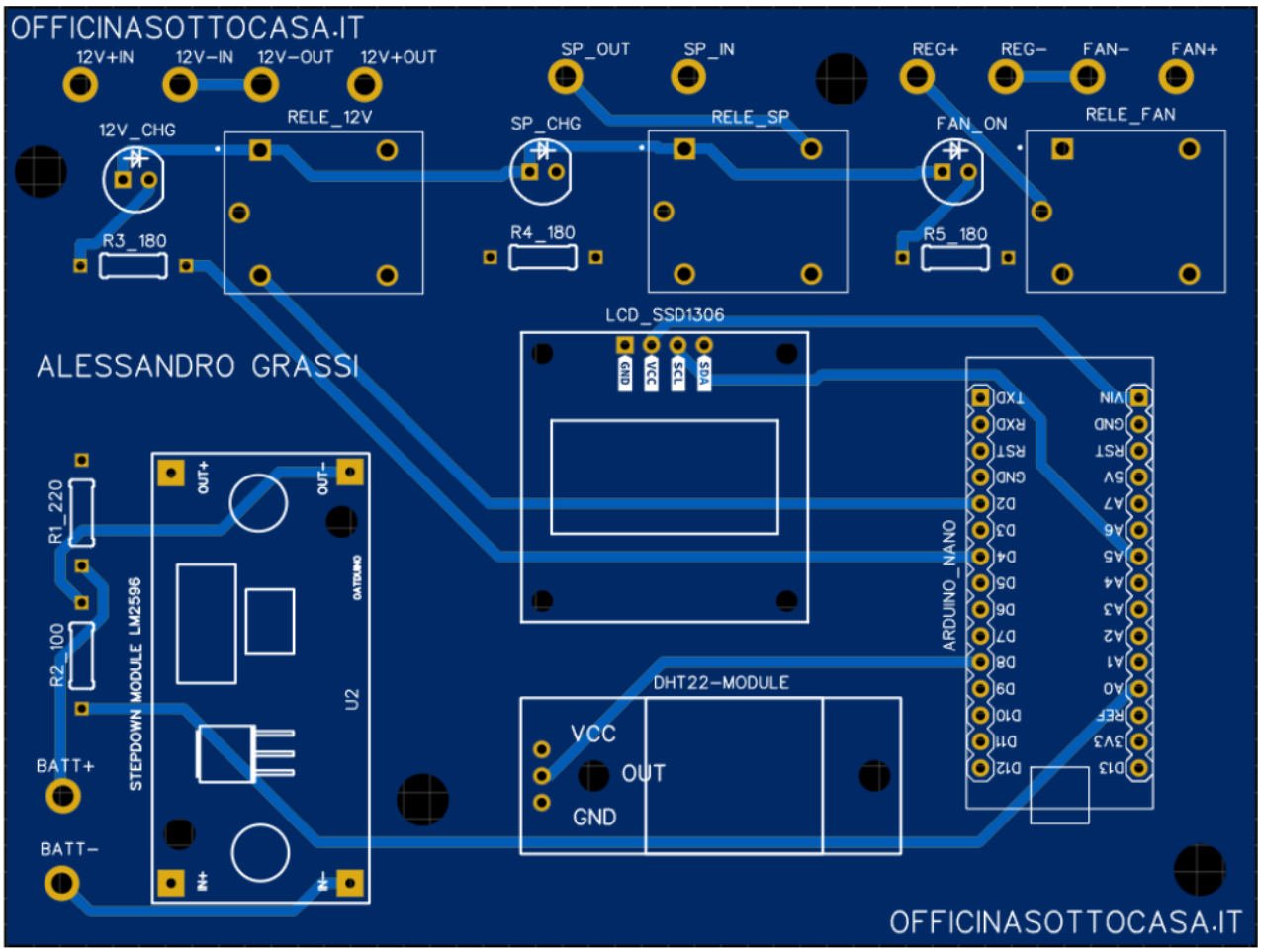
PCB

To create the PCB, we started with the electrical schematic. Through EasyEDA's online designer, it's possible to translate the electrical schematic into components on the PCB. The arrangement of the components is at our discretion. As you can see, we made an effort to maintain the separation between the groups identified in the electrical schematic, mainly for aesthetic and organizational reasons.

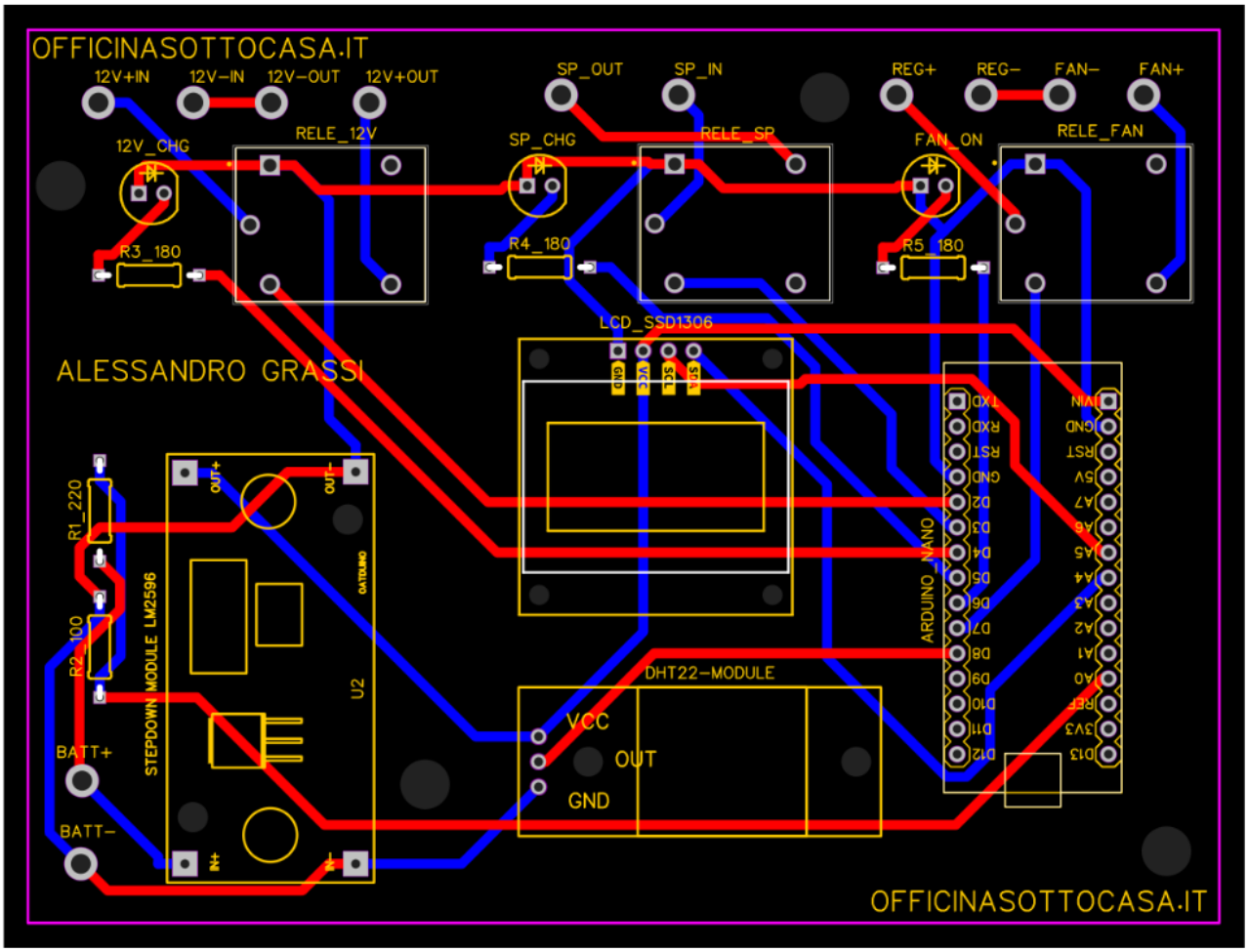
We purchased a two-layer FR-4 board with dimensions of 120 by 90 millimeters to integrate it into our electronic projects, ensuring reliability and precision in supporting complex circuits.

For the curious ones, FR-4, an acronym for "Flame Retardant 4," is a composite material often used for PCB (Printed Circuit Board) manufacturing. It consists of a base of epoxy glass reinforced with fiberglass,

impregnated with low-flammability epoxy resin. This material offers excellent dielectric properties, mechanical strength, and good workability, making it a common choice for electronic board production.



PCB Layout. In red, the top tracks, and in blue, the bottom ones.



Programming the ATMEGA328P Microcontroller Software

Until this point, our focus has been exclusively on the hardware aspect of the system. However, it's the software that will play the key role in orchestrating the entire system. The correct functioning, effective management, and synchronization of hardware components will be guided by our software.

We use the Arduino prototyping platform (and its clones) because Arduino programming offers intuitive access to creating custom electronic projects. The syntax is simple, and there is a wide range of libraries, allowing even beginners to quickly develop interactive devices.

Preliminaries for Programming

As always, since we need to manage recurrence in the Loop() function, we'll use some binary "dummy" variables (0 and 1) that will allow us to cyclically control environmental variables derived from sensor readings (humidity and charge level) but perform operations linearly (to clarify, without turning the relays on and off every 10 seconds).

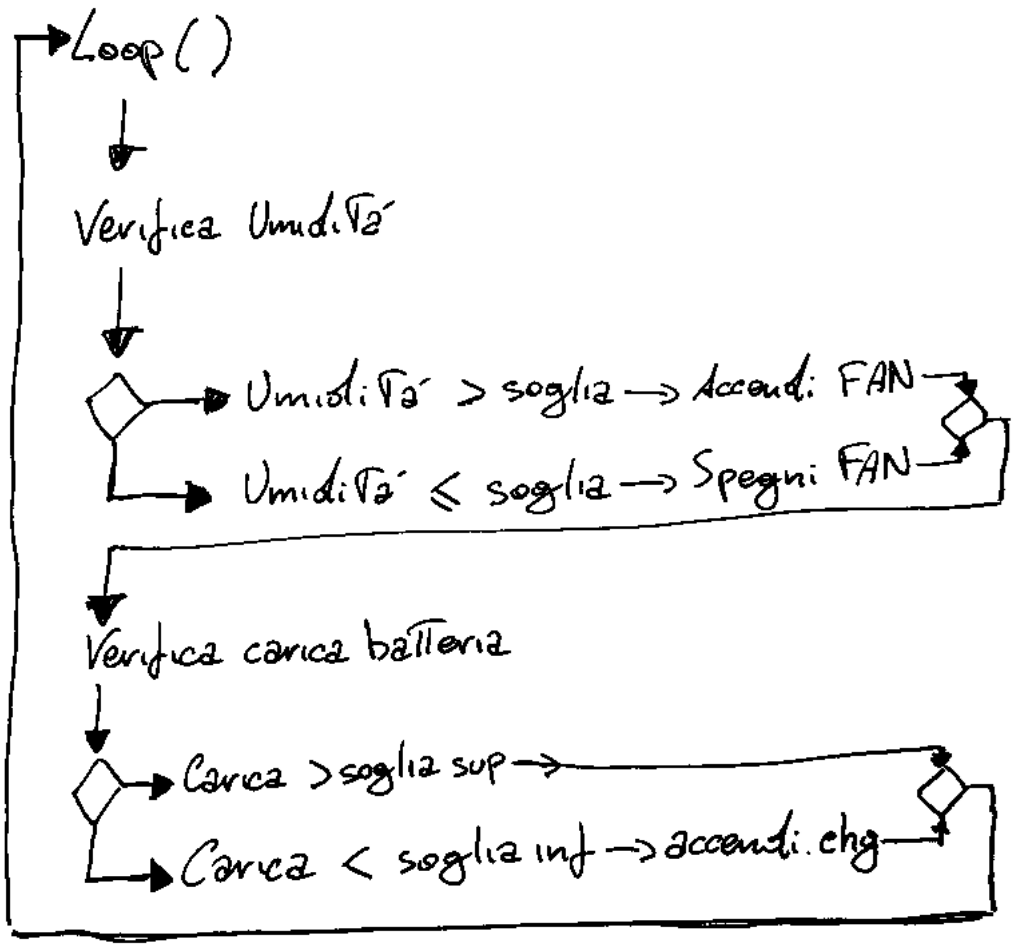
"Dummy variables" are generally used as temporary placeholders or to comply with syntax requirements without having a real functional utility in the context of the code. In practice, they are only used to meet the structure required by the programming language without being involved in significant operations or calculations in the program. The term "dummy" indicates that these variables are used only for temporary or formal purposes.

We will also use the watchdog function. This Arduino function refers to the integrated Watchdog Timer in AVR microcontrollers (Advanced Virtual RISC), such as those used in Arduino boards (or clones, as in our case). This timer is a hardware counter that can be programmed to generate a reset of the microcontroller if it is not "fed" regularly by inserting specific instructions in the code (restarting the timer before the set countdown ends). The watchdog is used in this project as a safety mechanism to automatically restart the microcontroller in case of malfunctions or program freezes.

Flowchart

Although the program might seem slightly complex, in essence, it is limited to a few simple steps:

- Check the humidity level, and if it is above a threshold value, turn on the extraction fans.
- Check the battery charge level, and if it is below a threshold value, turn on the external power to charge the battery.



In the charge control section, we have introduced an upper and lower threshold value. This way, within the range between the two threshold values, the relays remain in the previously set position to avoid annoying continuous state changes when the values are around the threshold value.

Libraries

To enable communication with various I/O peripherals, we first load some libraries.

In Arduino, libraries are sets of predefined code and functions that can be reused to simplify programming and interaction with specific hardware components. Libraries contain implementations of common or complex routines that can be called in your programs without having to write the code from scratch.

Libraries in Arduino are essential to make the development environment more accessible and allow users to focus on the specific logic of their project without having to manage low-level details. They offer several advantages:

- Code Reusability: You can easily use libraries to manage common functionalities in different projects without rewriting the code every time.
- Code Simplification: Libraries abstract complexity and implementation details, simplifying your main code.
- Simplified User Interface: Libraries often provide a high-level interface that simplifies interaction with hardware components.
- Community and Sharing: The Arduino developer community actively contributes to creating and improving libraries, allowing quick sharing of solutions and ideas.
- Development Speed: Using libraries can speed up the development process, allowing you to focus on the specific details of your project.

For example, if you are working with an OLED screen, you might use the Adafruit_SSD1306 or U8x8 library to simplify display management. Include the library in your code, and you can call functions defined by the library without writing the low-level code needed to control the OLED screen.

Let's now take a look at the code:

```
#include <avr/wdt.h> //includi libreria per utilizzo watchdog

#include <DHT.h>

#define DHTPIN 8 // Imposta il pin al quale è collegato il sensore DHT22

#define DHTTYPE DHT22 // Imposta il tipo di sensore (DHT22 o DHT11)

#include <U8x8lib.h>
// Imposta il pin di comunicazione I2C
#define OLED_SDA 4
#define OLED_SCL 5

//Inizializza lo schermo OLED128x64
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(OLED_SCL, OLED_SDA, U8X8_PIN_NONE);

// Inizializza il sensore DHT
DHT dht(DHTPIN, DHTTYPE);
```

The code snippet just provided begins with the inclusion of essential libraries for the watchdog, DHT22 sensor, and OLED display. Subsequently, connection pins for the DHT22 sensor and the OLED display are declared. An OLED 128x64 screen and a DHT22 sensor are then initialized, establishing the necessary I2C communication for the correct operation of the system.

Variable Definitions

In the following code snippet, constants and variables used for system control are declared. Pins for relays and LEDs, the reference voltage, the voltage divider pin, and threshold values for voltage and humidity readings are defined. The boolean variables fan_state and charge_state indicate the status of the fan and battery charge (on/off). They are modified whenever a condition occurs that results in a state change and

will serve as inputs to display information on our OLED screen. These parameters are crucial for adjusting the system's behavior based on environmental and power conditions.

```
const int REL_FAN = 7;
const int REL_CHG = 2;
const int REL_SPA = 3;

const int LED_FAN = 6;
const int LED_CHG = 4;
const int LED_SPA = 5;

const float tensioneDiRiferimento = 5.0; // Definisci la costante per la tensione di riferimento (5V p
er Arduino Uno)


const int pinPartitoreTensione = A0;// Definisci il pin analogico al quale è collegato il partitore di
tensione

// Specifica i valori delle resistenze del partitore di tensione
const float resistenzaR1 = 220000.0;  // 220 kohm
const float resistenzaR2 = 100000.0;  // 100 kohm

// specifica le variabili valori soglia per le letture della tensione
const float batt_low_limit = 11.0;
const float batt_high_limit = 13.0;

// specifica le variabili valori soglia per le letture dell'umidità
const float hum_low_limit = 70.0;
const float hum_high_limit = 75.0;

bool fan_state = false;   // Stato del fan (acceso/spento)
bool charge_state = false; // Stato della carica della batteria (acceso/spento)
```

Setup() Function

The setup() function initializes the Arduino program. It sets the output mode for pins controlling relays and LEDs, essential for system control. The DHT sensor is initialized using dht.begin(), and the U8x8 library for the OLED display is started with u8x8.begin(). Finally, the watchdog timer (wdt_enable(WDTO_8S)) is activated with a timeout of 8 seconds to monitor the correct operation of the program.

```
void setup() {
  pinMode(REL_FAN, OUTPUT);
  pinMode(REL_CHG, OUTPUT);
  pinMode(REL_SPA, OUTPUT);
  pinMode(LED_FAN, OUTPUT);
  pinMode(LED_CHG, OUTPUT);
  pinMode(LED_SPA, OUTPUT);

  dht.begin();// Inizializza il sensore DHT

  u8x8.begin(); // Inizializza la libreria U8x8
  wdt_enable(WDTO_8S);
}
```

Loop() Function

The loop function of the code cyclically monitors the system. After a 2-second waiting period, it calls the functions battery_level, humidity_level, and update_screen. The first function checks and manages the battery charge state, the second one monitors and adjusts humidity, while the third one updates the OLED display with current values. Additionally, it resets the watchdog timer to prevent a microcontroller reset.

```
void loop() {
  delay(2000);

  battery_level();

  humidity_level();

  update_screen();


  wdt_reset();  // Resetta il timer del watchdog

  }
```

Update_screen() Function

The update_screen function reads the battery voltage and humidity from the DHT22 sensor, calculates the results, and displays them on an OLED screen. It shows the voltage with two decimal places, indicating whether the battery is charging from a solar panel or the electrical grid. It displays the humidity with two decimal places and indicates whether the extraction fans are in operation or turned off.

```
void update_screen(){

  // Leggi il valore analogico dal pin A2
  int valoreLettura = analogRead(pinPartitoreTensione);

  // Calcola la tensione letta in base al partitore di tensione
  float tensioneLetta = (valoreLettura / 1023.0) * tensioneDiRiferimento * (resistenzaR1 + resistenza
R2) / resistenzaR2;

  // Leggi l'umidità
  float umidita = dht.readHumidity();
  u8x8.clearDisplay();
  u8x8.setCursor(0, 0);
  u8x8.print("Voltage: ");
  u8x8.print(tensioneLetta, 2); // Mostra la tensione con due decimali
  u8x8.print(" V");
  if (charge_state = true){
  u8x8.setCursor(0, 1);
  u8x8.print("220 charge");
  }
  else if (charge_state = false){
  u8x8.setCursor(0, 1);
  u8x8.print("Solar charge");
  }
  u8x8.setCursor(0, 2);
  u8x8.print("Humidity: ");
  u8x8.print(umidita, 2); // Mostra l'umidità con due decimali
  u8x8.print(" %");
  if (fan_state = true){
  u8x8.setCursor(0, 3);
  u8x8.print("fan running");
  }
  else if (charge_state = false){
  u8x8.setCursor(0, 3);
  u8x8.print("fan off");
  }
  u8x8.display();

}
```

battery_level() Function

The battery_level function measures the voltage of a battery through a voltage divider and checks if it is outside the specified thresholds. If the voltage is below the minimum threshold, it activates the relay and LED for charging from the electrical grid, turns off the relay and LED for solar charging. If the voltage exceeds the maximum threshold, it reverses the operations. The function tracks the charge state with a boolean variable.ion fans are in operation or turned off.

```
void battery_level(){

// Leggi il valore analogico dal pin A2
  int valoreLettura = analogRead(pinPartitoreTensione);

  // Calcola la tensione letta in base al partitore di tensione
  float tensioneLetta = (valoreLettura / 1023.0) * tensioneDiRiferimento * (resistenzaR1 + resistenzaR
2) / resistenzaR2;

  if (tensioneLetta < batt_low_limit) {
    charge_state = true;
    digitalWrite(LED_CHG, HIGH);
    digitalWrite(REL_CHG, HIGH);
    delay(200);
    digitalWrite(LED_SPA, HIGH);
    digitalWrite(REL_SPA, HIGH);
  } else if (tensioneLetta > batt_high_limit) {
    charge_state = false;
    digitalWrite(LED_CHG, LOW);
    digitalWrite(REL_CHG, LOW);
    delay(200);
    digitalWrite(LED_SPA, LOW);
    digitalWrite(REL_SPA, LOW);
  }

}
```

humidity_level()

The humidity_level function reads the humidity from the DHT22 sensor and checks if it is outside the predefined thresholds. In case of an error in sensor reading, it displays the error on the screen. If the humidity exceeds the maximum threshold, it turns on the relay and the fan LED for extraction. If the humidity falls below the minimum threshold, it turns off the relay and the fan LED. The function keeps track of the fan state with a boolean variable that is used to report the information on the LED screen.

```
void humidity_level() { // Verifica l'umidità
  float umidita = dht.readHumidity();  // Leggi l'umidità
  if (isnan(umidita)) {
    // Serial.println("Errore nella lettura del sensore DHT22!");
    u8x8.setCursor(0, 3);
    u8x8.print("Hum Read ERR");
    return;
  }
  if (umidita > hum_high_limit) {  // Verifica se l'umidità è al di fuori delle soglie
    digitalWrite(LED_FAN, HIGH); // Accendi il LED per il relè fan estrazione
    digitalWrite(REL_FAN, HIGH); // Accendi il relè fan estrazione
    fan_state = true; // imposta stato variabile estrattore su ON
  } else if (umidita < hum_low_limit) {
    digitalWrite(LED_FAN, LOW); // Spegni il LED per il relè fan estrazione
    digitalWrite(REL_FAN, LOW); // Spegni il relè fan estrazione
    fan_state = false;
  }
}
```