Mini Smart Weather Station

Khayman Fernandes Carneiro (29478) - Bibi Zeenat Malika Choolun (30797) - Sultana Shamsunnahar (33852)

1.Introduction

(by Sultana Shamsunnahar)

In the sectors of agriculture, climate analysis and public health environmental monitoring plays a really important and potential role. However, A usual weather station is so costly, bulky, and not so accessible to smaller-scale users. It is advance with the Internet of Things (IoT) technology, so there is an opportunity to develop low-cost, compact and more accessible solutions that can help with real-time environmental data collection and analysis. The main purpose of this project is to design and build a prototype of a Smart Mini Weather Station which will be capable of monitoring the key environmental parameters such as air temperature, humidity, sunlight intensity and soil moisture. Our system utilizes an ESP32S3 microcontroller to collect and transmit data by using the MQTT protocol, making it more suitable for remote monitoring. In addition, the project idea is to incorporate a live web-based dashboard for real-time visualization, and also for the historical analysis it supports data logging via a backend database. With our project we will try to explain practical applications of measurement and control systems and highlight the integration of sensors, microcontrollers, and communication protocols within an IoT-based monitoring framework.

2. Materials and Methods

(by Sultana Shamsunnahar)

2.1 System Overview

The system architecture is composed of:

•A microcontroller unit (MCU) for control and communication

•A set of environmental sensors for data acquisition

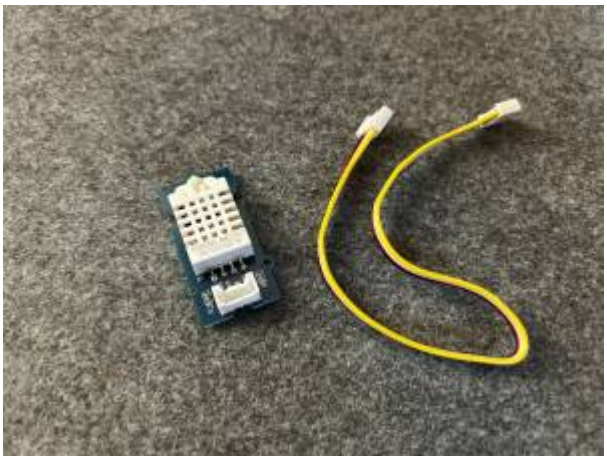
•A communication protocol for data transmission

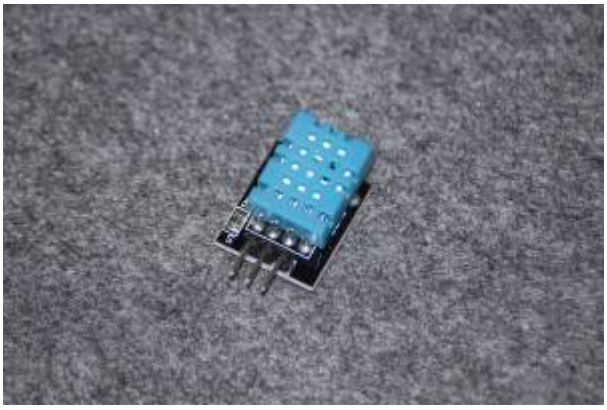
•A web interface for visualization

•An optional database for data storage

2.2 Materials

Component	Purpose
ESP32 Microcontroller	Central processing unit with built-in WiFi
DHT11 / DHT22 Sensor	Measurement of air temperature and humidity
Light Dependent Resistor (LDR)	Measurement of ambient light intensity
Capacitive Soil Moisture Sensor	Detection of soil moisture content
Power Supply (Rechargeable Battery)	Portable power source for outdoor operation
MQTT Broker (Ubidots)	Lightweight message server for data transmission
Web Dashboard (Ubidots)	Visualization and storage of data
Arduino IDE	Development and upload environment for ESP32 code
EasyEDA	Circuit design and diagram creation tool





2.3 Methods

2.3.1 Sensor Integration

The ESP32 was linked to each sensor via GPIO (General Purpose Input/Output) pins. Digital measurements of temperature and also humidity are provided by the DHT sensor that we used in the project. Moreover, a voltage divider circuit was used to connect the LDR in order to measure the intensity of light as an analog input. The analog signal produced by the capacitive soil moisture sensor is proportionate to the amount of water in the soil.

2.3.2 Data Acquisition and Transmission

The Arduino IDE was used to program the ESP32 to read the data from every sensor at predetermined intervals. To facilitate wireless transmission over WiFi, the gathered data was converted into MQTT messages and published to a distant MQTT Broker; in this case, Ubidots.

2.3.3 Data Visualization

Ubidots was selected to be the platform for the dashboard since it provided both MQTT broker and dashboard services simultaneously, as well as some limited database storage. Both web and mobile and devices can access more easily and use the dashboard.

2.3.4 Power Supply and Portability

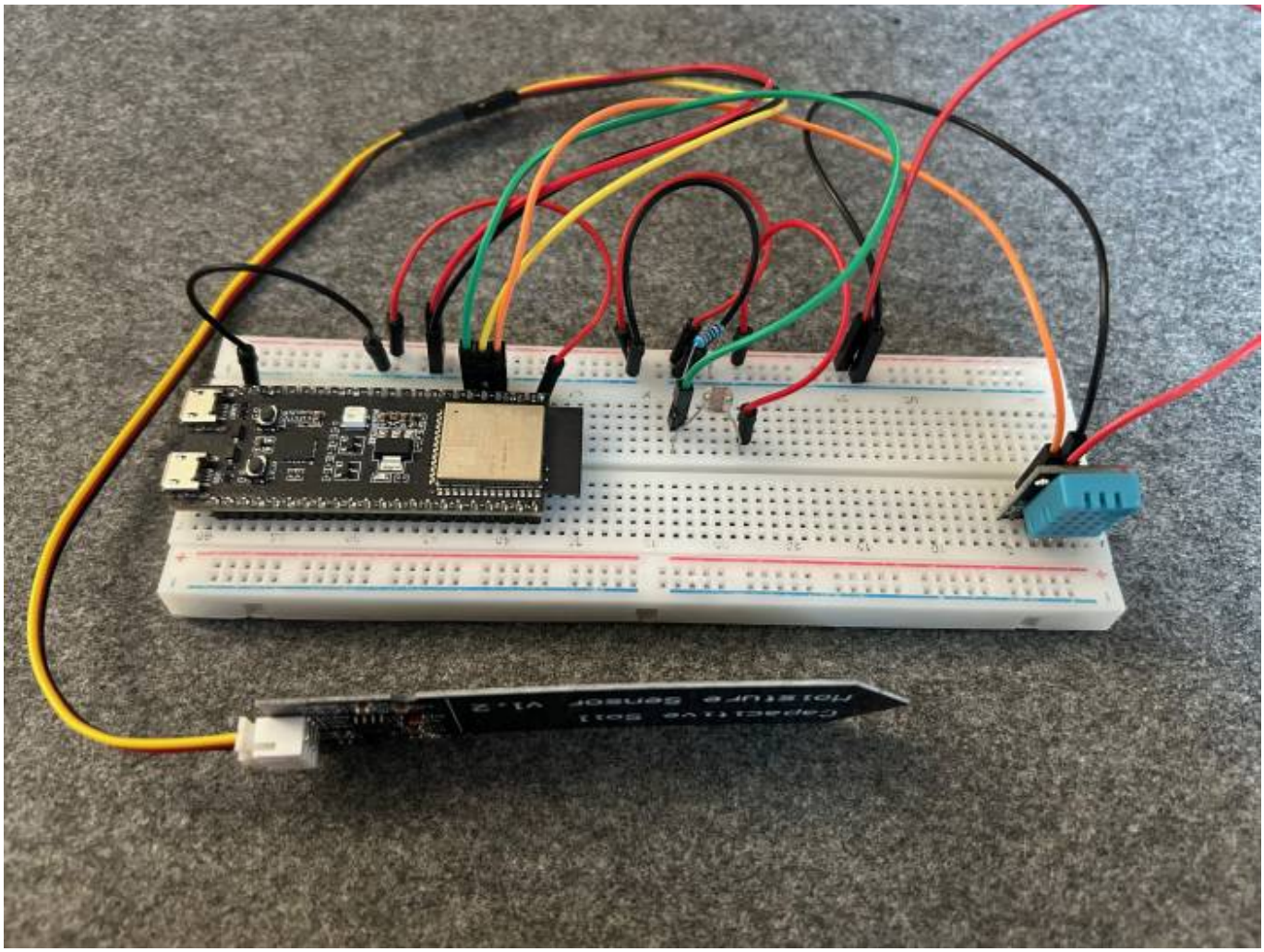
The system has two power modes: battery pack function for adoption in remote or outdoor areas, and USB power for indoor use. Longer battery life is guaranteed by the ESP32's low power consumption.

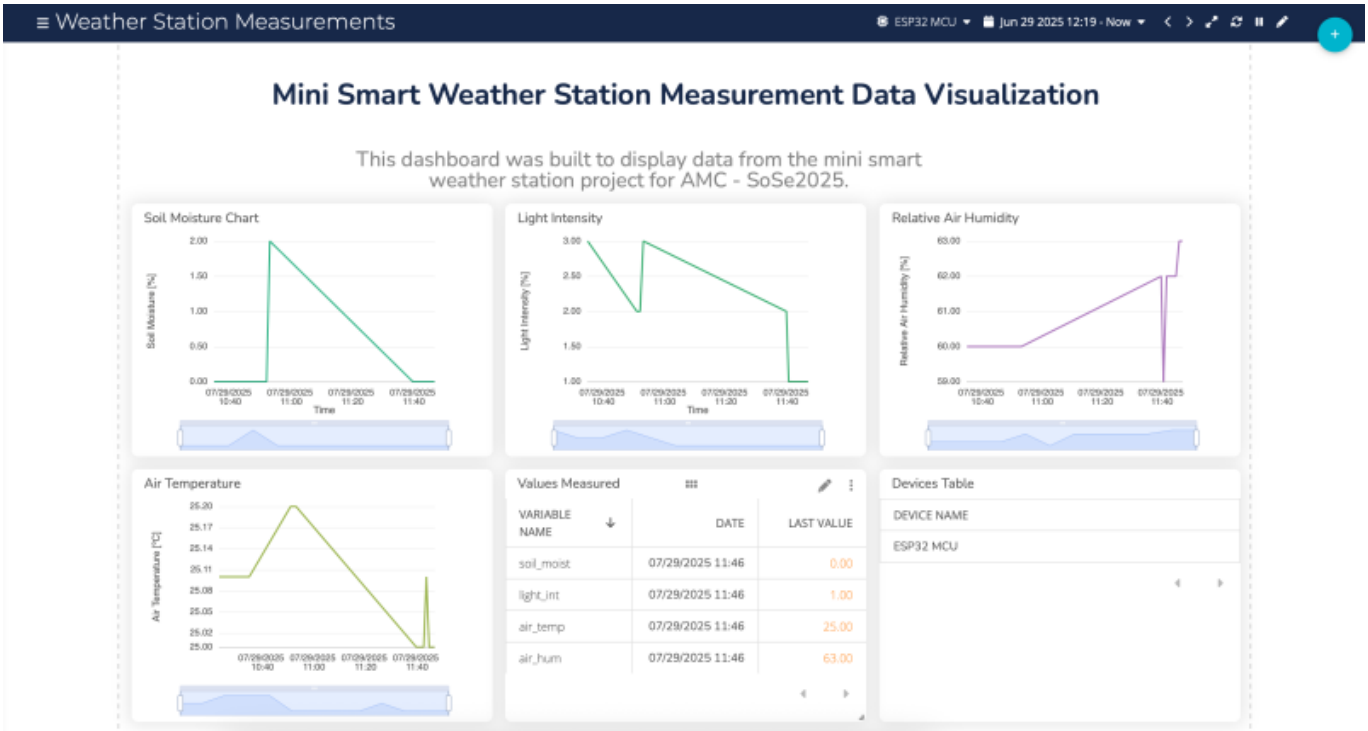
3. Results

(by Khayman F. Carneiro)

The prototype (Figure 3.1) successfully read the data collected from the sensors and sent it over MQTT to the Ubidots MQTT broker; the data received by the server was then available to be displayed in the dashboard (Figure 3.2). The air temperature and humidity sensor used was a DHT11 unit since it showed more accurate results than the DHT22 unit available when compared to a more expensive reference unit. The microcontroller used was the ESP32S3 connected directly to the host computer, since the more specific board did not function as intended, as will be discussed in the next chapter.

The code developed for it is shown below with comments where needed.






```
#include <DHT.h>
#include <WiFi.h>
#include <MQTTClient.h>
#include <UbidotsESPMQTT.h>

// Defining sensor variables
#define DHTPIN 4
#define DHTTYPE DHT11
#define SOILPIN 5
#define LDRPIN 6
DHT dht(DHTPIN, DHTTYPE);

// Defining MQTT/Dashboard variables
#define TOKEN "BBUS-elnIUM3JpLzaEsk3gpvkznDG8GCZUV"
#define WIFINAME "FRITZ!Box 7520 TE_EXT"
#define WIFIPASS "Coffee.Luna"
#define DEVICE_LABEL "esp32-mcu"
#define VAR_LABEL_1 "air_temp"
#define VAR_LABEL_2 "air_hum"
#define VAR_LABEL_3 "soil_moist"
#define VAR_LABEL_4 "light_int"
Ubidots client(TOKEN);
bool connected = false;

// Defining sleep variables
#define uS_FACTOR 1000000
#define PUBLISH_FREQUENCY 60
RTC_DATA_ATTR int bootCount = 0;

const int airStd = 3550;
const int waterStd = 1560;
```

```
int soilMoistPerc = 0;
int soilMoist = 0;
int hum = 0;
float tempC = 0;
int light = 0;
int lightPerc = 0;
const int minLight = 0;
const int maxLight = 3409;


// Auxiliary Function for Ubidots connection
void callback(char *topic, byte *payload, unsigned int length)  {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++)  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

void soilMoistPrint(int soil_moisture, int air_standard, int water_standard)
{
  soilMoistPerc = map(soil_moisture, air_standard, water_standard, 0, 100);

  if (soilMoistPerc < 0) {
    soilMoistPerc = 0;
  }
  else if (soilMoistPerc > 100) {
    soilMoistPerc = 100;
  }
  Serial.print("Soil Moisture: ");
  Serial.print(soilMoistPerc);
  Serial.println("%");
  Serial.println("---------------------------");
}

void dhtOutPrint(int humidity, float temperature_C) {
  if (isnan(humidity) | isnan(temperature_C)) {
    Serial.println("Reading Failed!");
  }
  else {
    Serial.print("Humidity: ");
    Serial.print(humidity);
    Serial.print("%");
    Serial.print(" | ");
    Serial.print("Temperature: ");
    Serial.println(temperature_C);

  }
}
```

```
void lightOutPrint(int light, int minimum, int maximum)  {
  lightPerc = map(light, minimum, maximum, 0, 100);
  Serial.print("Light Intensity: ");
  Serial.print(lightPerc);
  Serial.println("%");
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  delay(1000);
  dht.begin();

  ++bootCount;
  Serial.println("Boot number " + String(bootCount));
  esp_sleep_enable_timer_wakeup(PUBLISH_FREQUENCY * uS_FACTOR);

  client.setDebug(true);
  client.wifiConnection(WIFINAME, WIFIPASS);
  client.begin(callback);

  if (!connected)  {
    Serial.println("Not connected, attempting to connect");
    connected = client.connect();
  }
  Serial.println(connected);

  light = analogRead(LDRPIN);
  //lightOutPrint(light, minLight, maxLight); Debugging method for LDR
  hum = dht.readHumidity();
  tempC = dht.readTemperature();
  //dhtOutPrint(hum, tempC);  Debugging method for DHT sensor

  soilMoist = analogRead(SOILPIN);
  //soilMoistPrint(soilMoist, airStd, waterStd);  Debugging method for soil
moisture sensor
  if (connected) {
    client.add(VAR_LABEL_1, tempC);
    client.add(VAR_LABEL_2, hum);
    client.add(VAR_LABEL_3, soilMoistPerc);
    client.add(VAR_LABEL_4, lightPerc);
    client.ubidotsPublish(DEVICE_LABEL);
    client.loop();
  }

  Serial.println("Going to sleep");
  Serial.flush();
  esp_deep_sleep_start();

}
```

```
void loop() {
  // put your main code here, to run repeatedly:
}
```

4. Discussion

(by Khayman F. Carneiro)

4.1 Discussion of Results and Issues Encountered

The first prototype worked as intended; data was collected from the sensors, organized and sent to the Ubidots server and then displayed in a dashboard. This however had limitations: the board was connected to a host computer for power and the humidity sensor was not the ideal option since the DHT22 is a lot more accurate and has better range of measurements, according to Adafruit. In the scope of this project, where the project is supposed to be set up outside and data to be sent over MQTT to a server, wireless operation is of utmost importance. Based on this, and the fact that the frequency of measurement did not need to be very high, the use of the low power modes available on the ESP32 chips is very advantageous to the battery life of the project, allowing for it to be on the field taking measurements for a lot longer times since, when in deep sleep, the microcontroller and its peripherals can draw currents as low as 8.4uA (Espressif, n.d.).

In order to fulfill these requirements a few components were acquired:

- a Firebeetle ESP32 IoT microcontroller;
- a DHT22 sensor unit;
- a Lithium battery pack (4000mAh).

The microcontroller by Firebeetle is optimized for low power usage, making it ideal for such a project. Asides that the MCU has a JST1.25 connector for a battery pack and built-in battery charging via USB, which settled it as the perfect candidate for the weather station. The new board however did not work, as there was an error of communication between the computer and the board and, although the board was recognized by the computer correctly, with the right drivers installed, communication could not be set up even if the board was forced into bootloader mode by connecting GPIO0 to ground and pressing the reset button. That said, after extensive research and testing, the microcontroller appears to be faulty and needs to be returned, so proper evaluation of the performance of the project in the field so far is not possible.

As previously mentioned, the DHT22 sensor showed more deviation in its measurements than the DHT11 unit used in the first version of the prototype (when compared to a reference measurement unit), so the latter sensor was kept in place. Even after testing with a pull-up resistor to keep the signal coming from the sensor high, as it is mentioned in the material for AMC 2020 (available here), the measurements were deviating similarly, at around 20% less relative humidity measured and around 4ºC over the reference values. This could be due to sensor self heating or a malfunction, since the DHT units in general tend to vary a lot regarding their reliability and quality control.

Using Ubidots as a platform for MQTT transmission and storage proved to be really convenient since there was no separate configuration for the MQTT Broker and the dashboard and everything was

seamlessly integrated. The modularity of the platform allows for many devices to be added and managed, with each having their own separate variable spaces. Some of the issues encountered in the platform however are the lack of a truly secure way to connect the device to the broker, as it requires that the token is hardcoded. In addition to that there is the limitations of the free account, which, although minor compared to other options such as Amazon Web Services or FlowFuse, affected the testing procedures of the prototypes, specially the limitation on the amount of information that can be sent per day. This is not as limiting however for actual project operation since the measurements and information sent would be done hourly, meaning the maximum limit should not be reached.

4.2 Possibility for Future Improvements

There are many improvements that could be done to the project, both in hardware and software. Regarding hardware, the quality of the sensors could use improvements, since the ones used now are low cost and not very accurate, specially suitable for home projects and proofs of concept. The DHT sensor could be substituted for a more reliable BME280, which would also allow for measurement of pressure; the LDR could be replaced by a BH1750 sensor for greater precision in values, although for the purposes of this project, where measurements of light do not need to be as precise, the sensor would be far better than necessary. The same can be said about the soil moisture sensor, which could be substituted by a more advanced probe soil sensor for readings such as pH as well as soil moisture content. All of these improvements, specially the probe sensor, would increase the price of the project but also decrease the error in the measurements. Another possibility of improvement, regarding battery life, is the use of a miniature solar panel as to recharge the battery during the day; in conjunction with a microcontroller with appropriate low power modes, the devices could last for months, provided the battery is sufficiently large and there is enough sun to recharge it.

Regarding software, there are some different options that could also work well, such as the use of Node-RED, a SQL or Non-SQL database, and another dashboard web server such as Grafana, as to make the project more substantial and improve historic analysis capabilities, as well as scalability if multiple devices are used in different locations at the same time. Another idea would be to use instead of a timer to turn on the MCU from deep sleep, leaving the Wi-Fi connection on (using modem sleep) and sending a trigger to the board remotely when the measurement is needed.

This prototype mostly is a proof of concept of the viability and ease of use of such a device. If developed and refined, an enclosure could be designed as to protect the contents of the box while at the same time allowing for the proper measurements to be done, meaning an opening for the light sensor and concerns with heating of the components which could affect the temperature readings.

5. Conclusion

(by Bibi Zeenat Malika Choolun)

The case study of the Smart Mini Weather Station illustrates the integration of sensors, communications of IoT devices, and real-time data display as a low-cost and practical system for environmental monitoring. The prototype was able to gather the requisite data like air temperature, humidity, light intensity, soil moisture, and winds using the ESP32 microcontroller and DHT sensors and using the MQTT protocol, transmitted the information to the web-based dashboard. This project

was useful in demonstrating core concepts of measurement and control systems, as this device literally put those concepts to work. It showed the viability of using low-powered electronics and simple sensors for better environmental monitoring. This system has the potential to be adapted for smart agriculture, in weather forecasting, or in educational institutions with further enhancements. In a nutshell, this project was useful in creating real-world simulations with the Internet of Things, sensor networks, and data systems, as the knowledge is very useful in a technology-driven economy.

The prototype Smart Mini Weather Station had some noteworthy features, but the following limitations should also be considered.

* Power Source: An obvious concern is the device's first iteration, powered by a computer's USB port, which makes the device unsuitable for outdoor or remote usage. * Sensor accuracy: The older version of the device was outfitted with a DHT11 sensor, which had inferior accuracy. * Management of Battery Life and Power: Integrate a rechargeable battery while utilizing the low power modes of the ESP32 to make the device portable and conserve energy. * Protective Casing: Create a protective enclosure of the device that will guard it from rain, dust, and extreme temperatures. * Alerts: Notify users of issues such as dry soil and perform advanced data analysis, suggesting resolutions based on the collected data. * Enhanced Dashboard: Add mobile compatibility, as well as increase interaction via charts, trends, and data download capabilities to the dashboard. * Enhanced Functionality: Connect multiple stations to monitor wider regions such as farms or school campuses.

Explanation Video

The videos showing the explanation and demo are zipped below. Also included is the schematic file showing how everything is connected. Uploaded here for the size of the zipped file is still too large for Moodle.

final_project_folder.zip

References

(by Sultana Shamsunnahar)

1.Anemometer Working Principle Circuit Digest. (2019). *Anemometer - Wind Speed Sensor Working and Applications*.[https://circuitdigest.com/tutorial/anemometer-working-and-applications]

2.Fritzing – Circuit Design Tool Fritzing.org. (n.d.). *Fritzing: Open-source Electronics Design Software*.[https://fritzing.org/]

(by Bibi Zeenat Malika Choolun)

3.Pandey, S. K., Srivastava, S., Singh, V. K., & Kumar, V. (2025, May 24). ESP32-Based Weather Station. ResearchGate. ResearchGate

4.Ganesan, S., Lean, C. P., Li, C., Yuan, K. F., et al. (2024). IoT-Enabled Smart Weather Stations: Innovations, Challenges, and Future Directions. Malaysian Journal of Science and Advanced Technology. ResearchGate+1mjsat.com.my+1

(by Khayman F. Carneiro)

5.Smarthon. (n.d.). Digital light sensor. Smarthon Docs. Retrieved July 29, 2025, from https://smarthon-docs-en.readthedocs.io/en/latest/Sensors_and_actuators/Digital_Light_sensor.html

6.Espressif Systems. (n.d.). Boot mode selection. Espressif Documentation. Retrieved July 29, 2025, from https://docs.espressif.com/projects/esptool/en/latest/esp32/advanced-topics/boot-mode-selection.html

7.Espressif Systems. (2023). ESP32 datasheet (Version 3.6). https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

8.Last Minute Engineers. (n.d.). ESP32 sleep modes & power consumption. Retrieved July 29, 2025, from https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

9.Espressif Systems. (n.d.). Current consumption measurement for ESP32 modules. Espressif Documentation. Retrieved July 29, 2025, from https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/current-consumption-measurement-modules.html

10.Adafruit. (n.d.). DHT11/22 temperature and humidity sensors. Adafruit Learning System. Retrieved July 29, 2025, from https://learn.adafruit.com/dht/overview

11.Espressif Systems. (n.d.). ESP32 sleep modes. Espressif Documentation. Retrieved July 29, 2025, from https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html

—