Temperature and Moisture Meter

*

1st Sarp Acar
Computer Science Department of Watson Engineering School
Binghamton University
New York, USA
sacar1@binghamton.edu

***Abstract*—This paper present an enery-efficient temperature, humidity and soil moisture sensor system mainly for agricultural workers. This system's components includes ESP32 microcontroller, DHT22 for temperature and humidity sensing, and DHT11 for soil moisture sensing. This system's main purpose is to measure environmental parameters which mentioned above to be able to provide a system to agricultural workers and maximize their profit at the same time. This maximization provided with this system's paramenter values which will help their products to grow more wholesome and at the same time less energy consumption of the system will cost them less. The primary objective is to create a reliable and sustainable system for the workers.**

The system stands out for energy efficiency with the appropriate useage of ESP32's deep sleep mode. The system programmed to sleep and specific time intervals which those time intervals are set by the user, after the time period ends, it wakes up and performs a measurement of the temperature, moisture and soil moisture values of the system. Post-aggregation, the system transmits the data to Thingspeak platform via Wi-Fi usage, offering a robust and sustainable data preserve and display the data on a chart. With this feature of the system, live data can be observed on the chart and future actions can be taken by the user.

Moreover the paper states the design considerations of the system including system's architechture, sensor integration, data handling and energy efficiency strategies. Finally this paper will a give brief information about system's capability to operate with minimal energy while ensuring consistent data acquisition and transmission.

I. Introduction

In the domain of Internet of Things (IoT) the advanced microcontroller like ESP32 has played a huge role of this system's energy efficiency. It is because of the deep sleep mode of ESP32. This paper present a detailed analysis of the system and the implementation of software part. In situations where real-time data collecting is essential for decision-making processes, such as precision agriculture, environmental monitoring, and smart home applications, this system is especially pertinent. The motivation behind this project is to providing agricultural workers an energy efficient system inside all different measurement tools.

This system's measurement capability with the combination of energy efficiency aspect will provide agricultural workers more profit especially financially. With the less usage of energy will give them more reliable and sustainable system. The ESP32's dual processor, Wi-Fi capability and deep sleep mode functions offers and ideal way of energy efficiency. Moreover the DHT11 and DHT22 sensors for temperature, humidity and soil moisture measurement ensures comprehensive environmental data gathering.

As it mentioned above the most critical aspect of this project is to provide energy efficiency to the users. For this objection, the system mainly relies on the deep sleep function that ESP32 microcontroller offers. The function provides a circumstance to get the microcontroller into deep sleep mode where the system lowers the consumption of every activity that it was doing especially its energy consumption. While the sleep mode is not active, system wakes up which means it turn back to its old version of doing activites and energy consumption. During that state of actions, systems uses it edge tools which are DHT22 and DHT11 to measure the environmental values such as the temperature, humidity and soil moisture. These are crucially significant values for agricultural crops to grow, that is the reason why these sensors had chosed while creating the architechture. After accumulating an hour's worth of data, the system continuously connects to Wi-Fi and transmits the gathered data to the ThingSpeak platform for storage and analysis. At the ThingSpeak platform each data is stored in charts. By the usage of those charts every alteration of temperature, humidity and soil moisture values can be observed, tracked and can be used for future analysis of growing agricultural products. This ThingSpeak approach not only conserves power but also ensures to balance the load on network.

Besides of the newtwork connections the architectural design of the system is tailored to maximize the most efficient operational work and also maintian the data privacy, system reliablity and sustainability of the system. All those specifications which mentioned above involves careful planning of sensor integration, correct algorithms for both time interval determination, sensor readings and lastly communication protocols. All of those happened with the usage NVM(Non-volatile memory) in the ESP32. It is used for data retention between all sleep cycles which determined by the user. It is crucial to not to lost any data through the process of sleeping cycles. In addition , the implementation covers network connectivity

difficulties, data transmission correctness optimizations, and error handling mechanisms to enlarge the system's robustness.

Finally all those tools, mechanisms and software to complete the system design can provide insights into athmospheric and soil conditions helping farmers to predict what to do for future applications on their practices. This system is not only for the agricultural purposes, it can be used for home systems, urban planning, industrial purposes and with some slight alterations it can be used for Climate data centers as well.

II. Usage

The main usage aspect of this ESP32 based design is its user-friendly architecture and design. This features are making it easily accesible and wide spread of usage areas. The range of users are vary from hobbyist and educators to professional researchers and agricultural users. This system has simple instructions and ease on its use which enables every class of users to understand, analyze and manipulate the system's functions according to their own needs and purposes. Users can easily configure the device, because of its intuitive interface for adapting parameters such as data collection intervals and network settings. The integrating process of common and easily usable sensors like DHT11, DHT22, and soil moisture sensors more enhances its user control, as these components are not only cost-effective but also well-documented and supported by a tons of online resources.

Integrating the system with the Thingspeak data visualization platform increase the chance for users to analyze their data more precisely. ThingSpeak which is known for its user-friendly interface, has a significant function in enabling real-time interpretation and accessibility of the environmental data gathered by the system. The environmental analysis is crucially important especially in agriculture industry. It is because the alterations of the environment and can be rapid and unexpected. Users will become able to see their data on numerous charts and keep track of their process. This situation also examines the situation for future years products.

III. DESIGN

We explore the fundamentals of the environmental monitoring system in the third half of this paper, with a special emphasis on the reasoning behind the certain temperature, humidity, and soil moisture sensors as well as the complexities of the Wi-Fi physical layer signal structure. The choosing process of DHT11 DHT22 and soil moisture sensore are all based on their cost-efficient nature, accuracy, reliability and ease of integration with ESP32 microcontroller. These sensors provide a cost efficient solution while keeping a high quality of data integrity.

On the other hand, The IEEE 802.11 standard, which details communication over the physical and data link levels, is the foundation for the ESP32 module's Wi-Fi capabilities. This enables the device to safely and effectively send collected data to websites such as ThingSpeak. The specifications like

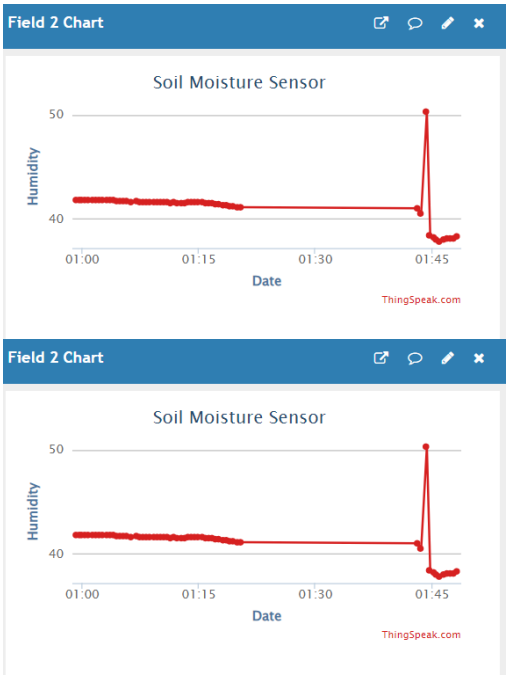

Fig. 1. ThingSpeak Platform Data Charts

modulation, data rate, and channel bandwidth gets significant approach on operational efficiency and data transmission trust.

In addition the connections of ESP32 and DHT11,DHT22 is also an extremely complicated and crucial aspect of the end result. The complete system requires many different aspects such as the wiring schematics, power management considerations, and the data flow process from acquisition to transmission.

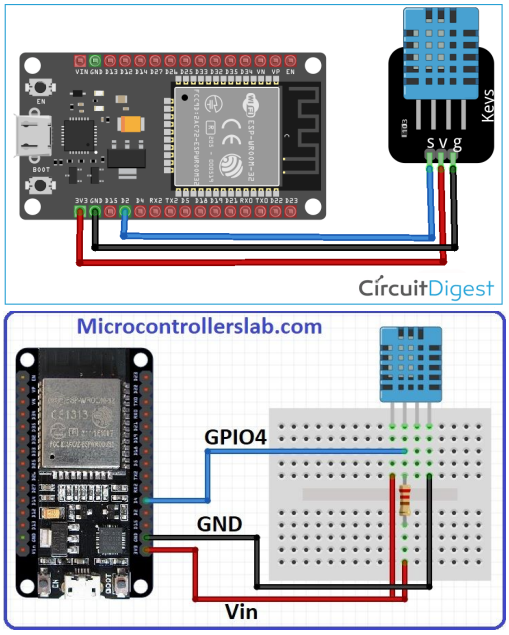

Fig. 2. The Connections between ESP32 and DHT11,DHT22

A. Connections Between ESP32 and Sensors

The system's main component is ESP32 microcontroller, it is a versite and powerful also the upgraded version of ESP8266. It facilitates both Wifi connection and also sensor integration. As the side components the DHT22 and DHT 11 are sensors for temperature and humidity measurement, and a soil moisture sensor for detecting moisture levels under the ground. The ESP32 which is the main component of the system is further mentioned by its many options of power supply options; it can be powered either through its USB port or an external battery connected to its VIN pin, accommodating various application requirements, from stationary to mobile deployments. Every sensor that used at the system has VCC, GND, and Data pins. These sensors are connected to the ESP32, with their VCC pins linked to a 3.3V pin on the ESP32, GND pins to a ground pin, and Data pins to GPIO pins, such as GPIO21. In addition the Wifi qualification of ESP32 module provides a circumstance not to need any other module for Wifi. It connects to the local networks for efficient and correct data transmission to ThingSpeak platform to dispaly the data on charts. As another aspect of the system, ESP32 periodically keeps the data and stamps it onto the terminal. Same data result can be seen at ThingSpeak charts as well. This data is temporarily stored in the ESP32's memory or in non-volatile memory (NVM) to preserve it across sleep cycles. However after every sleep cycle ends, since the next measurement will begin, the data which belongs the previous measurement will gone and the new one will held as the current values of environment. There is a time interval attached to the system's integration. This time period will determine the sleep mode time. At those time periods system will take itself to sleep mode and every consumption value of the system drops. By this way the system provides energy efficiency.

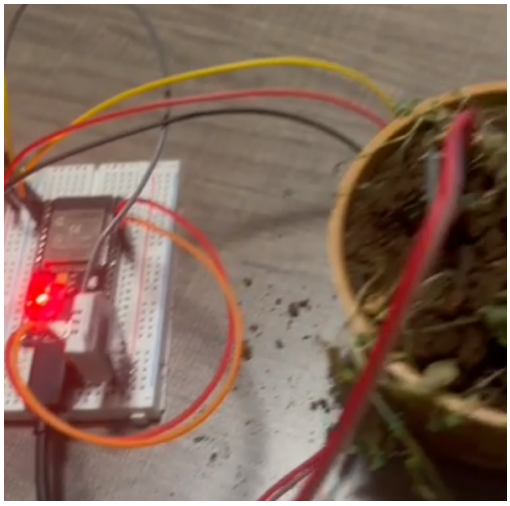

Fig. 3. Whole System

B. Software Section of System

The software section for powering the ESP32-based environmental monitoring system is completely designed to efficiently manage the collection and transmission of critical environmental data, specifically temperature, humidity, and soil moisture levels. At its core the system has ESP32 microcontroller to control the DHT11 and DHT22 sensors and its datas. It periodically gather readings and display those readings on the terminal and ThingSpeak. After those readings store in ESP32 non-volatile memory, the system gets into the sleep mode and during this time period, no significant actions are taking from the whole system. The wake-up after the time set sleep mode referred at the system as boot number. Every boot number is presented at the terminal. In order to understand how many times the system get into the sleep mode and got out later. This boot number count also helps the user to be able to measure how much energy has kept and how much has been used throughout the whole time of system's work. In addition every cycle has its own values of temperature, humidity and soil moisture percentage. They all are presented at ThingSpeak as the numerated charts. All of those are giving the full output of The sleep mode to active mode conversion, memory load data, Sleep mode warning and Current values of temperature, humidity and soil moisture.

As of the code, some specific libraries must be added firstly in order to run it successfully. Those are "DHT.H", "espsleep.h", "Prefherences.h","WiFi.h","HTTPClient.h" and "ThingSpeak.h". First DHT.h library's main objection is for the lines where reading data from sensors happen. It enables ESP32 to connect with the DHT11 and DHT22 libraries. With the usage of this library, the system can succesfully gathers the metrics and continue to make measurements successfully at every sleep cycle. The espsleep.h library is the core library and must be added to provide a sleep cycles and waking the system up afterwards. It is the main library that makes this system energy efficient. This is where the system sleeps during the given time intervals. The preferences.h library basically responsible for the data handling and preserving the data at non-volatile memory for a short time period. The library provides a simple usage for reading and writing key-value pairs in non-volatile memory. This makes it easier for the system users to store and retrieve data without dealing with lower-level details of non-volatile memory management. The wifi.h is basically for the connection of wifi by ESP32. It is crucial for sending the data to ThingSpeak and display the data on the charts. The HTTPClient.h facilitates HTTP communication, which is essential for interacting with the ThingSpeak. And lastly ThingSpeak.h library is for the ThingSpeak key and channel Id connections which is also crucial for the whole system.

```
1   #include <DHT.h>
2   #include <esp_sleep.h>
3   #include <Prefherences.>
4   #include <WiFi.h>
5   #include <HTTPClient.h>
6   #include <ThingSpeak.h>
```

Fig. 4. Libraries Included to Code

After including those needed libraries we set the correct

connections which has made on the board and set the pins to correctly determine system's integration. Also setting the time interval is also made at this section. Figure can be shown below.

```
#define DHT_SENSOR_PIN 21
#define DHT_SENSOR_TYPE DHT22
#define SOIL_MOISTURE_SENSOR_PIN 34
#define TIME_TO_SLEEP  10
const long uS_TO_S_FACTOR = 1000000;
```

Fig. 5. Libraries Included to Code

After setting the correct pins and making the physical connections the place where we make the wifi connections and ThingSpeak connections has to be done. In order to do that the wifi ssid and password must be set as a variable. The reason why we set those is just because we will use them later on the wifi connection function below of the code. Since the whole wifi connection system is for transmitting the data to ThingSpeak, we also need to make adjustment for ThingSpeak as well. ThingSpeak platform, gives every user a channel id and API key for adjust the connections and make the transmit the given data to their own platform. In order to do those corrections, ın the code two chars must be set and those values must be assigned as mentioned above. Below figure will show the exact implementation to how to do it.

```
const char* ssid = "SSİD";
const char* password = "pw";
const char* THINGSPEAK_API_KEY = "ABCDEF";
const unsigned long THINGSPEAK_CHANNEL_ID = 12345;
```

Fig. 6. Wifi and ThingSpeak Connections

Later, we are finally initializing the setup function. The function starts with initializing the Serial communication at a baud rate of 115200, ensuring real-time logging and debugging capabilities. A brief delay ensures the Serial Monitor is adequately prepared to display the output. The system's operational count, denoted by bootCount, is incremented and displayed in the terminal, it refers to the amount of times system gets into the sleep and reactivates itself again which is a straightforward indication of the number of wake-up cycles since the system's deployment. Moreover, The DHT sensor which's responsibility is to measure temperature and humidity, is initialized using dhtsensor.begin(), enabling the system's performance on environmental readings. In addition, the system's preferences are initialized with preferences.begin("env-data", false), setting up a non-volatile memory space for data keeping between deep sleep cycles. Network connectivity go through connectToWiFi() function which also written below the code part, it is a critical step that enables the ESP32 to communicate with ThingSpeak for data transmission. Subsequently, takeReading() is called to measure the current environmental data from the sensors. This data is then processed and transmitted to ThingSpeaks via checkAndSendData(), ensuring that the collected information is accessible for analysis and visualization.Finally, the ESP32 is set to enter deep sleep mode for a time interval defined above as TIMETOSLEEP, conversion to microseconds using uSTOSFACTOR. This approach is basic in decreasing the system's energy consumptionly significantly which is one of the critical challenges in continuous environmental monitoring. The work of espdeepsleepstart() then puts the ESP32 into a deep sleep state, effectively decreasing its energy consumption. This cycle of waking, processing, transmitting data, and returning to deep sleep forms the outline of the system's operation, balanced actions between consistent environmental monitoring and energy efficiency. Below is the code for this part.

```
void setup() {
  Serial.begin(115200);
  delay(1000); //Take some time to open up the Serial Monitor
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));
  dht_sensor.begin();
  pinMode(SOIL_MOISTURE_SENSOR_PIN, INPUT);
  preferences.begin("env-data", false);
  connectToWiFi();

  takeReading();
  checkAndSendData();

  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +" Seconds");
  Serial.println("Going to sleep now");
  delay(1000);
  Serial.flush();
  esp_deep_sleep_start();

}
```

Fig. 7. Setup Function

After completing the setup function, the system will take the readings from their sensors. Take reading function mainly designed for sensors, this function goes through by reading the humidity and temperature from the DHT sensor through dhtsensor.readHumidity() and dhtsensor.readTemperature(). These methods works as the retrieval of real-time humidity and temperature values, which are crucial parameters for system's environmental monitoring. After that, the system reads the soil moisture level using analogRead(SOILMOISTURESENSORPIN), which gets the moisture data from the soil moisture sensor connected to the ESP32. This reading is particularly significant in our system's success. Once these environmental parameters accurately measured, the function continues to keep this data in the ESP32's non-volatile memory (NVM) with the usage of preferences. The humidity, temperature, and soil moisture values are stored as float and integer values , ensuring that this vital data is kept to some point where system will reboot and continue on th next operation.

```
void takeReading() {
  float humidity = dht_sensor.readHumidity();
  float temperature = dht_sensor.readTemperature();
  int soilMoisture = analogRead(SOIL_MOISTURE_SENSOR_PIN);

  // Store data in NVM or RTC memory
  preferences.putFloat("humidity", humidity);
  preferences.putFloat("temperature", temperature);
  preferences.putInt("soilMoisture", soilMoisture);

}
```

Fig. 8. TakeReading Function

Before going into the wifi connection function, there is one more function that initialized in the code which is checkAndSendData. As looked at the name it does what it says. The function starts by initializing the ThingSpeak client with ThingSpeak.begin(client), which sets up the necessary conditions for data transmission with the usage of Wi-Fi. This

step is significant for making the ESP32 to communicate with the ThingSpeak.

Next, the function continues to collecy the humidity, temperature, and soil moisture values from the ESP32's non-volatile memory. The use of NVM ensures that these data points are kept in between the ESP32's deep sleep cycles. After the data has been successfully retrieved, it can be visible to the Serial Monitor or terminal. The main part of the function contains transmitting the data to ThingSpeak. This is made by use the temperature, humidity, and soil moisture values to specific prepared charts in the ThingSpeak using the ThingSpeak.setField() .

Each environmental parameter is assigned to a different chart. The ThingSpeak.writeFields() method then transmits this data to ThingSpeak channel, the identification of the THINGSPEAKCHANNELID and authenticated using THINGSPEAKAPIKEY.

Lastly, after completing the data transmission, the function clears the kept values in non valotile memory . This ensures that memory is ready to have new data in the next cycle. Figure of the function can be shown below.

```
65  void checkAndSendData() {
66    // Initialize ThingSpeak
67    ThingSpeak.begin(client);
68    // Retrieve data from NVM
69    float avgHumidity = preferences.getFloat("humidity", 0);
70    float avgTemperature = preferences.getFloat("temperature", 0);
71    int avgSoilMoisture = preferences.getInt("soilMoisture", 0);
72
73    // Send data to ThingSpeak
74    Serial.println(" avgTemperature  : " + String( avgTemperature));
75    Serial.println(" avgHumidity  : " + String(avgHumidity));
76    Serial.println(" avgSoilMoisture : " + String(avgSoilMoisture));
77    ThingSpeak.setField(1, avgTemperature);
78    ThingSpeak.setField(2, avgHumidity);
79    ThingSpeak.setField(3, avgSoilMoisture);
80    ThingSpeak.writeFields(THINGSPEAK_CHANNEL_ID, THINGSPEAK_API_KEY);
81    // Clear the stored values after sending
82    preferences.putFloat("avgHumidity", 0);
83    preferences.putFloat("avgTemperature", 0);
84    preferences.putInt("avgSoilMoisture", 0);
85  }
```

Fig. 9. Check and Send Data Function

Last lines of the code consists of wifi connection. This function starts by checking the current Wi-Fi connection status using WiFi.status(). If the ESP32 has not already connected to a Wi-Fi network (indicated by WiFi.status() != WLCONNECTED), the function continues to attempt a new connection using WiFi.begin(ssid, password).

This command tries to make the connection of the ESP32 to the Wi-Fi network using the parameters defined at the beginning of the code which are ssid and password. The function then enters a loop, continuously checking is the connection has been successfully made. This is done using the while (WiFi.status() != WLCONNECTED) loop, which continuosly delays the process by one second (delay(1000)) to allow time for the connection to be made. The loop ensures that the ESP32 does not continue with the code until a Wi-Fi connection has been made. This mechanism is important for ensuring that the ESP32 has access to a local network before trying to make any data transmission.

As for the output, previously given code is producing some specific information, current mode and states of the ESP32.

```
87  void connectToWiFi() {
88    if (WiFi.status() != WL_CONNECTED) {
89      WiFi.begin(ssid, password);
90      while (WiFi.status() != WL_CONNECTED) {
91        delay(1000);
```

Fig. 10. Wifi Connection Function

Firstly, The ESP32 sets a reset cause labeled as rst:0x5 (DEEPSLEEPRESET). This code give information that the ESP32 has been woken up from a deep sleep mode, a energy-saving mode where most the system achieves the research challenge. The boot:0x13 (SPIFASTFLASHBOOT) implies that the ESP32 is booting up using the SPI flash in a fast mode.

Next, The line `configsip 0, SPIWP 0xee` and the `clk_drv, q_drv, d_drv, cs0_drv, hd_drv, wp_drv` settings are connected to the application of the SPI flash and the ESP32's internal clocks. These settings are default values used for normal operation and memory interaction. The mode: DIO, clock div:1 means that the ESP32 is operating in Dual input output mode for fast communication, which is a common specification for faster data transfer, and the clock divider is set to 1, which means the normal clock speed.

Next to this, there are three load lines back to back, each displaying different memory addresses and the lengths of the code parts being loaded into the ESP32's memory. These lines shows that the process of loading from the flash memory into the ESP32's internal memory. The entry 0x400805f0 is the first entry point to the program.

Finally, Boot number: x, in the below example 26 is a ordinary print statement, from the user's code, means that this is the xth time the ESP32 has booted itself up. This is mostly used in situations to keep track of how many times the device has get into the sleep mode and woken up from the sleep, which can be very beneficial for debugging or tracking the device's operation.

```
rst:0x5 (DEEPSLEEP_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
Boot number: 26
```

Fig. 11. Output of the Code

IV. ENERGY EFFICIENCY

The main purpose of the whole system is to provide agricultural workers an energy efficient measurement tool to make their measurement process less cost. At this point, the ESP32 deep sleep mode integrated into the current system with the addition of time interval so that the system can get in and out of the deep sleep mode continuosly. While the system is in the sleep mode, less energy is going to be consumed than the normal activation mode. In this section of the paper. Numbers is going to be presented to be able to display how

much total energy consumption happened and how much cost efficiency made for the users. We will define all tool's energy consumption while in the sleep mode and active mode. The energy consumtion formula is below.

$$Power(P) = Current(I)xV(Volt) \tag{1}$$

A. ESP32

The ESP32 microcontroller has 260 microampers while in deep sleep mode and has 240 nanoampers while in active mode. Since we are using 3.3 volts at our system. So while in sleep mode ESP32 consumes;

$$0.000858nW = 260nAx3.3V \tag{2}$$

While in active mode ESP32 consumes;

$$792nW = 240uAx3.3V \tag{3}$$

B. DHT11

The DHT11 sensor has 100 microampers while in deep sleep mode and has 200 nanoampers while in active mode. Since we are using 3.3 volts at our system. So while in sleep mode DHT11 consumes;

$$0.00033W = 100nAx3.3V \tag{4}$$

While in active mode ESP32 consumes;

$$0.00066W = 200uAx3.3V \tag{5}$$

C. DHT22

To measure the power consumption of DHT22 In this system the min voltage is 3.3 . The used current in the measuring (active) mode is 1.5 mA, and in the stand by mode (not measuring) is 50 uA. Use the following formula to measure the power:

$$3.3V50uA = 0.25mW \tag{6}$$

While in active mode ESP32 consumes;

$$3.3V1.5mA = 4.95mW \tag{7}$$

For the given values, when the system gets in and out between being active for 5 seconds and sleeping for 5 seconds (which is approximate value of wake up seconds, with good conditions of wifi signal strength, and fast sensor readings) throughout an hour, the total energy consumption is approximately 396 watts. This cycle of going between active and sleep modes results in significant energy savings.

In conclusion of energy efficiency aspect, if the ESP32, DHT11, and DHT22 remain continuously active for an entire hour, the total energy consumption would be about 792 Watt-hour.

Therefore, by applying a strategy where the system is active for only 5 seconds and then sleeps for 5 seconds in each cycle, we can approximately decreace the energy consumption, saving about 396.00 watt in an hour. This approach explains the effectiveness of power management strategies in IoT devices, where periodic sleep modes can extremely reduce energy usage without significantly impacting performance.

ACKNOWLEDGMENT

I give my sincere gratitude and to Watson School of Engineering staff, Professor Dali Ismail and Zainab Altaweel for their contribution to this project. Special thanks to them for their guidance, and resources.

CONCLUSION

In conclusion, this paper has briefly demonstrated the design, implementation, and actions of an energy-efficient, ESP32-based environmental monitoring system. The system's performance capability to accurately calculate the values of temperature, humidity, and soil moisture, integration of the DHT11, DHT22, and a soil moisture sensor, gives an idea of its potential as a reliable system in environmental data gathering and using into the process of agricultural purposes. The innovative approach of integration of these sensors with the ESP32 microcontroller, applied by the significant use of deep sleep mode, plays a critical challenge in energy efficiency aspect. Significantly, the usage of Wi-Fi connectivity for transmitting data to the ThingSpeak platform has showed the system's ability to not only keep but also communicate data in an effective for real-time analysis and display on the chart. This project showed as a development cost-effective and sustainable IoT project for environmental monitoring.

Throughout the time period of this project, several points were achieved, mostly in the terms of power management, data integrity, and IoT connections. Some difficulties completed, such as ensuring sensor readings and keeping Wi-Fi connectivity, provided valuable results in system design . Moreover, the project's adaptability to different environmental areas , such as agricultural fields and urban homes, shows its wide spread of applicability and potential easy usage.

REFERENCES

[1] JC Alamilla-Magaña, E Carrillo-Ávila, J Jj Obrador-Olán, C Landeros-Sánchez, J Vera-Lopez, and JF Juárez-López. 2016. Soil moisture tension effect on sugar cane growth and yield. Agricultural Water Management 177 (2016), 264–273.

[2] Vasyl Cherlinka. 2018. Soil Moisture: How To Measure Monitor Its Level. Web. https://eos.com/blog/soil-moisture/

[3] Jingjin Wu, Yujing Zhang, Moshe Zukerman, and Edward Kai-Ning Yung. 2015. Energy-efficient base-stations sleep-mode techniques in green cellular networks: A survey. IEEE communications surveys tutorials 17, 2 (2015), 803–826.

[4] Jian Ding and Ranveer Chandra. 2019. Towards low cost soil sensing using Wi-Fi. In The 25th Annual International Conference on Mobile Computing and Networking. 1–16

[5] He, Y. Gong, and W. Lin, "Facility Agriculture Intelligent Monitoring System Based on Wireless Sensor Networks", Journal of Anhui Agricultural Sciences, vol. 38, pp. 4370-4372, 2010.

[6] Ibrahim Al-adwan, M S-D 2012 The Use of Zigbee Wireless Network for Monitoring and Controlling Greenhouse Climate International Journal of Engineering and Advanced Technology (IJEAT) 2 35