Internet of Things and Wireless Sensor Networks

# Newcastle University

# School of Engineering

# EEE8121: Internet of Things and Wireless Sensor Networks

# Group No. 24

**Group Members:** 230611159 - **Sadhana Ramu** (Group Leader)

230340248 - **Siddharth Balu Pagare**

230649244 - **Spoorthy Matada Siddalingaswamy**

230319598 - **Shashank Ramachandra Maidur**

**Module Leader**: Dr Gui Yuan Tian

# Abstract

The history of humankind has always been a good teacher, as shown by the fact that people learn from their mistakes. Learning about the history of a country is made much easier via the study of its national heritage. As a direct consequence of this, these intangible cultural artefacts get a prominent level of attention.

The challenge of effectively preserving the works of art shown at museums and galleries, while maintaining a low level of energy usage is one that is of the utmost importance. It is a responsibility that we owe to both the current and the next generation. When these artefact collections are not adequately managed, they are vulnerable to the effects of severe environmental conditions. Because of this, it is very necessary to always keep a close eye on the climatic conditions, whether it be in storage or during displays.

The preservation of artefacts is significantly impacted by the degree to which environmental elements such as brightness, temperature, and humidity are controlled and managed. Due to the widespread use of new digital technologies, the recording, storing, and showing of cultural heritage assets have become more dependable and cost-effective.

By using a Wireless Sensing and Monitoring Platform that is made possible by the Internet of Things (IoT), we can monitor the temperature, relative humidity, and light in the setting of museums and artefacts. This enables us to better preserve and protect these important collections. The use of an IoT driven monitoring system, as opposed to the conventional measuring tools and techniques that are utilised in art galleries, may assist in the undertaking of measurements in a manner that is continuous, on a real-time basis, as well as in a manner that is much simpler and more cost-effective. The newly created technology is capable of automatically adjusting the light intensity surrounding the artefact to the appropriate range. The Android application not only tracks temperature and humidity but also has the capability to manually adjust the level of light intensity.

# CONTENTS

**List of tables**



**List of figures**



**Key words**: ESP32 Microcontroller; Xiao ESP32S3; Dust Sensor; Humidity Sensor; RFID reader; Temperature and Light sensor; artefacts conservation; Internet of Things.

# Chapter 1 - Introduction

## 1.1 Background

Art galleries, museums, and historic sites are major draws for visitors in every nation, and huge economic advantages are expected to be given or obtained from the numerous historical assets that have a tremendous potential. In today's world, there are several factors that may contribute to the deterioration of art and artefact collections over time, including environmental factors, acts of vandalism committed by humans, and natural deterioration. Art collections are required to live in an appropriate microclimate of temperature, humidity, and light in order to halt or at least slow down the pace of deterioration of artefacts and prolong the life of the artefacts themselves. Therefore, the preservation of historical arts and artefacts in an atmosphere that is both stable and regulated is the best option. In order to guarantee the arts will be preserved for the long term, it is a necessity that these influences affecting the artefacts be controlled, or at least, reduced.

The preservation of artefacts by preventing and restricting the degrading process of the artefacts in the first place is the goal of monitoring and regulating the interior environment in a gallery. At the same time, the monitoring and control of the indoor environment in a gallery is intended to ensure that the occupants are provided with a pleasant atmosphere. The accurate measurement and monitoring of these characteristics by manual methods is laborious, takes a lot of time, is not dependable, and is, to some degree, impossible. These methods depend on the computations and records made by humans. Because it is entirely dependent on human beings, there is a significant risk of making mistakes. There is a significantly reduced risk of mistakes when the monitoring process is mechanized to the extent that it is independent of human involvement. This results in a significant lengthening of the time artefacts may be used. In order to do this, it is necessary to have an IOT (Internet of Things) and wireless sensor network that can gather and keep track of the data remotely.

The term "Internet of Things" (IoT) refers to a hypothetical future in which commonplace objects and people are able to intelligently interact with one another and their immediate surroundings, as well as with other people and organizations, in order to provide contemporary services via the medium of the Internet. The Internet of items enables remote monitoring and administration of items as well as the environment in which they are located by using actuators and sensors.

This information may be used to better regulate the atmosphere of the gallery and, as a result, enhance the preservation of cultural artefacts.

The use of sensors like dust sensor, humidity sensor, temperature and light sensor and RFID reader and tag in the monitoring of gallery ambiance provides a number of benefits over more conventional monitoring. Some of the advantages of this technology are the cheap cost, the minimal visual effect, minimal energy utilization, the great flexibility, the absence of the requirement for infrastructure, the simplicity with which the sensor nodes may be deployed, and the ability to continually monitor and regulate the environment of the museum.

## 1.2 Aim & Objectives

The project involves the development of ESP32 sensor connected to various sensors, transmitting data to the cloud via Thing Speak for IoT-Driven Artifact Conservation. Additionally, a security system was implemented to regulate room access. The main objectives include understanding the working principles of each sensor, calibrating sensors for precision, exploring cyclic redundancy checks (CRCs) and Time Division Multiple Access (TDMA), and writing software for sensor interfacing with either ESP32 or Arduino. Subsequently, the collected sensor data will be sent to the IoT platform. Finally, IoT-Driven Artifact Conservation will be realized through integration with IFTTT, synchronizing with Thing Speak.

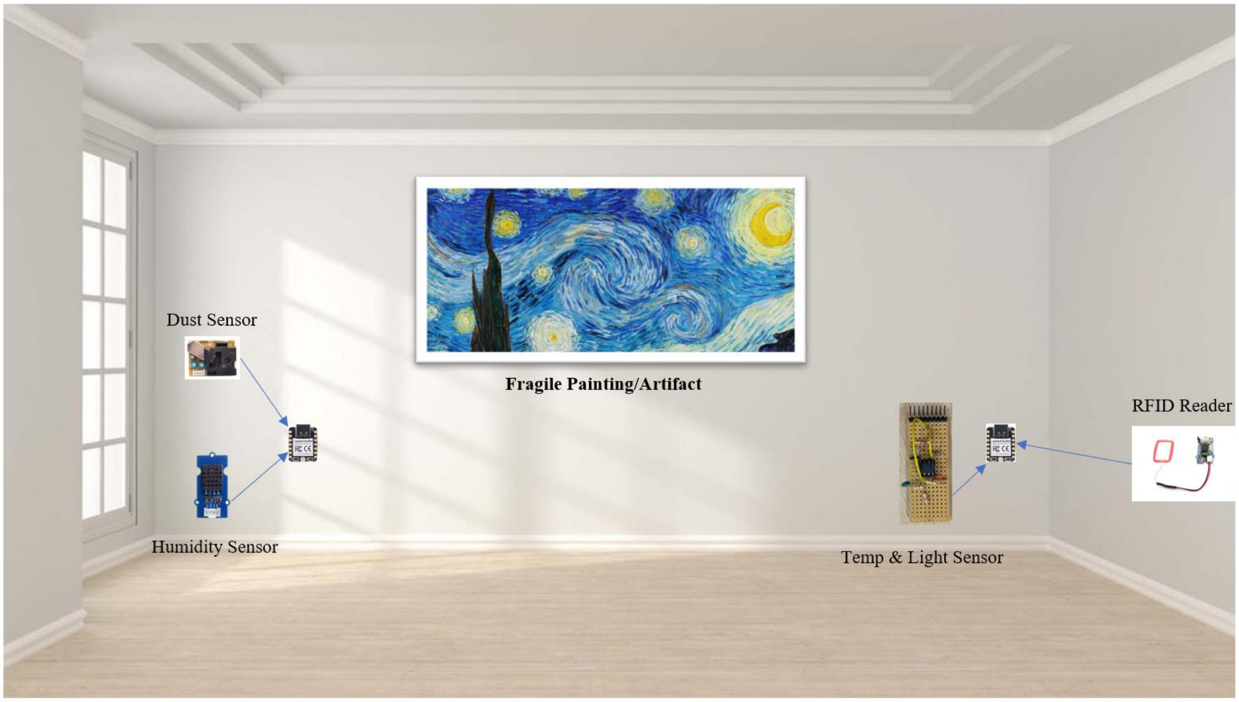



Figure 1: Visualization of our storyline

## 1.3 Group Contribution and Effective Management

The team, comprising four members, effectively divided responsibilities between hardware and software aspects of the project. Sadhana, as the Group Leader, focused on interfacing and calibrating the Dust sensor. Shashank interfaced and calibrated the Humidity sensor and contributed to the writing of report. Siddharth led the hardware and interfacing management, overseeing the integration of sensor boards fabricated during lab sessions, also prepared presentation for the demonstration. Spoorthy took charge of the RFID reader and its interfacing and worked on the final report along with Siddharth. Although challenges emerged with hardware connections and software code, collaborative efforts, assistance from demonstrators, and individual research successfully resolved these issues. The collective goal was to ensure each member contributed to at least one sensor, allowing for seamless integration with ESP and Thing Speak, thereby creating a cohesive IoT-driven Artifact Conservation system.

# Chapter 2 - Literature Review of IoTs (Internet of Things), WSNs, Sensors and Communications

The preservation of art and artefacts has been a subject of study for an extended period, where conventional approaches of classifying and conserving artefacts have depended on manual categorization and examination by specialists. Nevertheless, the latest developments in Internet of Things (IOT) and Wireless Sensor Network (WSN) have shown significant promise in improving the classification and conservation of cultural heritage artefacts. The paper [10] introduces a tailored IoT-Wireless monitoring system that tracks temperature, humidity, and light levels. This system is crucial for overseeing the gallery environment, which in turn plays a vital role in preserving the artwork in the museum. The piece has been verified by Shreyas Folk Art Museum, an Indian Heritage site located in Ahmedabad, India. The system employs a tailored hopping technique to send data from the transmitter node to the reception node. The data received is seen and logged onto a personal computer (PC) using a Graphical User Interface (GUI) created in LabVIEW. In order to safeguard our artefacts from theft and damage, as well as avoid deterioration caused by dust, we decided to install a security system. This system consists of an RFID reader and tag for protection against theft, and a dust sensor to prevent degradation caused by dust particles.

Preserving and conserving cultural heritage holds significant importance due to the inevitable degradation of artworks over time. Deterioration is influenced by a range of factors, including material composition, exposure to weather conditions, and human interactions. Ideally, artworks should be safeguarded within stable, controlled climates, necessitating vigilant monitoring and meticulous record-keeping of environmental conditions. [1]

In reference [9], a demonstration highlights a prototype of an indoor air quality monitoring system integrating an ESP32 microcontroller with a MQ-136 gas sensor and a dust sensor. This system effectively tracks temperature, humidity, dust particle levels, and harmful gases, presenting the collected data on the ThingSpeak platform.

In reference [5], focuses on creating a people access control system utilizing the ESP32 module. This system is engineered to collect data from a triple access mechanism, enabling individuals to either make direct payments, display store consumption receipts, or allow workers to present their identifications through RFID technology.

# Chapter 3 - Project Specifications and Implementations

## 3.1. Design and Development of Sensors

### 3.1.1 Hardware working of DHT Sensor

The Grove - Temperature & Humidity Sensor utilizes the DHT20 sensor, an enhanced iteration of the DHT11. Compared to its predecessor, the DHT20 boasts increased accuracy in temperature and humidity measurements, alongside a broader measurement range. Notably, it incorporates I2C output, simplifying its usability. Connecting these sensors to various microcontrollers like Arduino or ESP32 allows for immediate measurement of humidity and temperature values. They are equipped with an NTC thermistor and a capacitive humidity sensor to facilitate this functionality seamlessly.

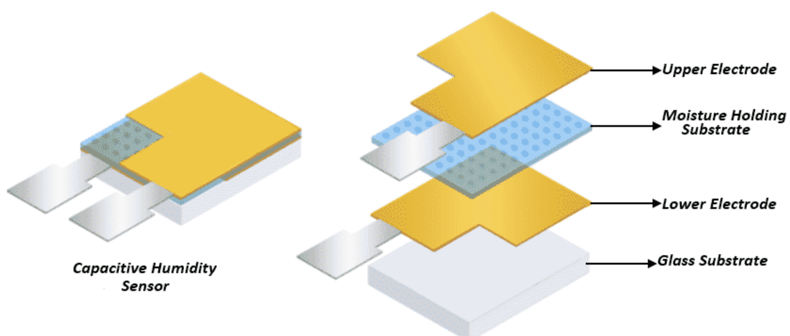



Figure 2: Internal Structure of Humidity Sensor

The DHT20 integrates a specialized sensor chip featuring a high-performance semiconductor-based capacitive humidity sensor and a standard on-chip temperature sensor. Its data output is in the standardized I2C format. Significantly surpassing the reliability of its predecessor, the DHT11. The new generation of upgraded products have been improved to make their performance more stable in elevated temperature and high humidity environments; at the same time, accuracy, response time, and measurement range of the product have been greatly improved. [11]

To gauge humidity, they utilize a humidity sensing element containing two electrodes and a moisture-holding substrate, typically composed of a conductive plastic polymer or salt. This configuration forms a capacitive humidity sensor, akin to capacitor structures.

As environmental humidity fluctuates, the substrate's conductivity changes, altering the resistance between electrodes proportionally. Higher relative humidity reduces the resistance, while lower humidity increases it. The Integrated Circuit (IC), housed on a small PCB in an 8-bit SOIC-14 package, detects, and processes these resistance changes. This IC performs analog-to-digital conversion on the signal, utilizing stored calibration coefficients, ultimately generating a digital output containing temperature and humidity data for simple reading.

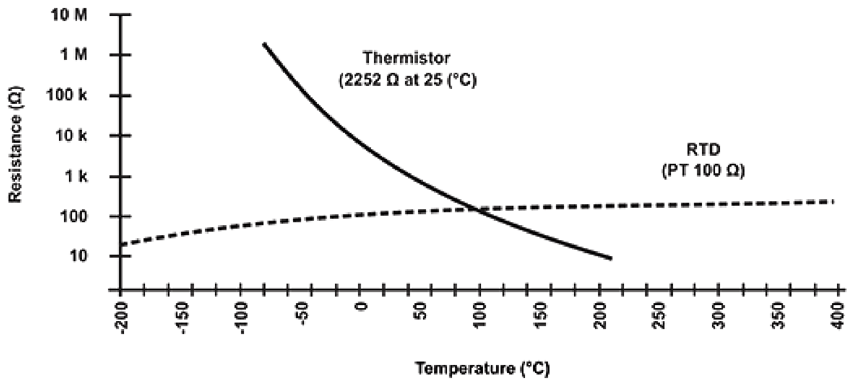



Figure 3: Variations in Resistance of NTC Thermistors Across Temperature Ranges

A thermistor operates as a variable resistor, adjusting its resistance in response to environmental temperature changes. With rising temperatures, the resistance of a Negative Temperature Coefficient (NTC) thermistor diminishes.

Standard equation or NTC thermistor's resistance as a function of temperature is. The equation

$$R_T = R_{25C} * e^{\left(\beta * \left[\left(\frac{1}{T+273}\right) - \left(\frac{1}{298}\right)\right]\right)}$$

can be written as:

R25C is the thermistor's nominal resistance at room temperature (25 °C). This value is normally provided in the datasheet.

β (beta) is the thermistor's material constant in Kelvin. This value is normally provided in the datasheet.

T is the thermistor's actual temperature in Celsius.

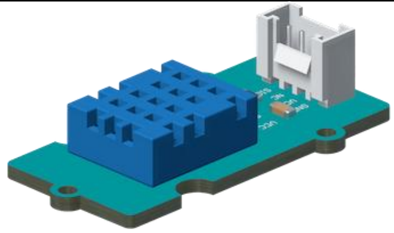
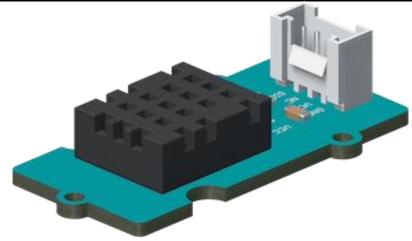
| Sensor Name | DHT11 | DHT20 |
|---|---|---|
| |  |  |
| Difference | Blue Color | Black Color |
| Input Voltage | 3-5V | 2.0V – 5.5V |
| Measuring Humidity Range | 20-80% | 0 ~ 100% RH |
| Humidity Accuracy | 5% | ± 3 % RH (25 ℃) |
| Temperature Accuracy | 0-50°C / ± 2°C | -40 to 80°C /± 0.5 ℃ |

Table 1. Comparison between DHT11 and DHT20

### 3.1.2. DHT20 Protocol

DHT sensors utilize a simplified single-bus communication method, employing a solitary data line for transmitting and controlling received data. In instances where the device cannot release the bus, it connects an open drain or tri-state port to the data line, enabling data transmission while allowing other devices to access the bus. Typically, a single bus incorporates an external pull-up resistor of around 5.1kΩ, resulting in a high state when the bus is inactive. Operating in a master-slave structure, only the host initiates communication with a slave, necessitating strict adherence to the single bus sequence for devices accessing the host.

Data serves as the means for communication and synchronization between the microprocessor and DHT20. This involves a single-bus data format, comprising a 40-bit data transfer with a high signal at the outset, alongside an 8-bit parity bit used to validate data accuracy.

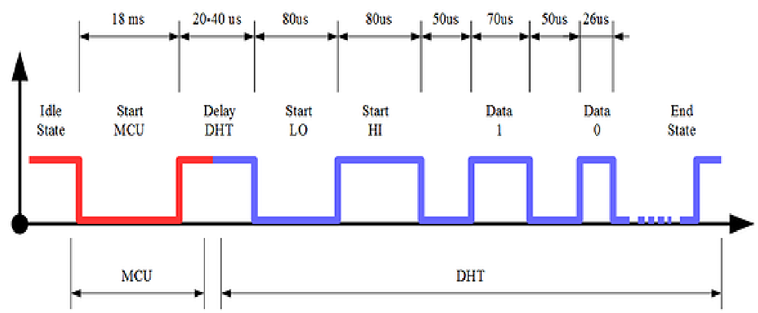



Figure 4: Single Bus Communication in DHT

The data structure for the DHT sensors comprises: an 8-bit integer for humidity, an 8-bit integer for decimal humidity, 8 bits for temperature and humidity.

data, an 8-bit integer for decimal temperature, and an additional 8-bit parity bit.

Communication between the master and slave occurs through specific steps: Initially, the microprocessor's I/O output transitions to a low state, a duration that must be at least 18ms. Subsequently, the microprocessor's I/O enters a state that activates the pull-up resistor, causing the data line from the DHT11 to become high. This transition signals the sensor to switch from low power consumption to a higher consumption state as it awaits a signal from the microcontroller. The completion of this start signal is crucial, as the sensor will not respond without it.

The detection of the start signals triggers an 80us low voltage pull from the DHT11 as it prepares to transmit data. It then follows by sending information to the microcontroller at a low voltage lasting 50us. Each data bit begins with a 50us signal. The bits represent either "1" or "0" based on the duration of the signal; a "1" maintains a 70us high state, while a "0" sustains a 26-28us high state.

While one transmission typically takes almost 5 milliseconds to complete, it is advisable to wait at least 2 seconds between transmissions due to the characteristics of the sensors being used. This extended waiting period ensures a more reliable and stable operation, allowing sufficient time for the sensors to reset and avoid potential interference or inaccuracies in subsequent readings.

### 3.1.3. RFID Reader and Tag

Art galleries often have difficulties in managing and safeguarding their invaluable art collections due to the intricate nature of the task. The authentication system of the art gallery is constructed using a Grove 125kHz RFID reader and tag as its primary components. The implementation seeks to improve security and monitoring capabilities in art galleries or museums by using RFID technology for effective inventory management and safeguarding against theft or unauthorized handling. The use of Radio-Frequency Identification (RFID) technology is extensive due to its capacity to provide distinct identification to artworks and artefacts.

The Grove 125KHz RFID reader operates based on the idea of low-frequency radio wave transmission. Some of the key features and specifications that helped us to choose this for our implementation are:

**Grove 125kHz RFID Reader:**

- **Frequency:** Operates at 125kHz frequency.
- **Compatibility**: Usually designed to be compatible with EM4100 or comparable 125kHz RFID tags.
- **Communication Interface**: Can make use of UART (Universal Asynchronous Receiver Transmitter), I2C, or different serial communication protocols.
- **Voltage Requirements**: Operates at 5V
- **Antenna**: An in-built antenna is provided with the reader
- **Read Range**: The device has a sensitivity capable of detecting objects up to a maximum distance of 7 centimeters (about 2.76 in).
- **Form Factor**: Compact and often designed for seamless interface with plenty of development environments.
- **Output Format**: Generates tag data in a legible format for seamless interaction with microcontrollers.

**Grove 125kHz RFID Tag:**

- **Frequency:** Operates at 125kHz frequency.
- **Chip Type:** EM4100 or related chips are often used.
- **Material:** Usually constructed from PVC or other resilient substances.

- **Size and Shape:** Available in several dimensions and forms, often resembling cards or key fobs.
- **Unique ID:** Every tag is assigned a unique identification that is encoded into the chip.
- **Read-Only or Read-Write:** Certain tags have read-only functionality, whilst others may have the ability to be both read and written. We have used read-only.
- **Durability:** Unaffected by environmental elements such as water and dust.
- **Encoding:** It has the capability to be pre-encoded with distinct information or may be programmed to suit certain purposes.

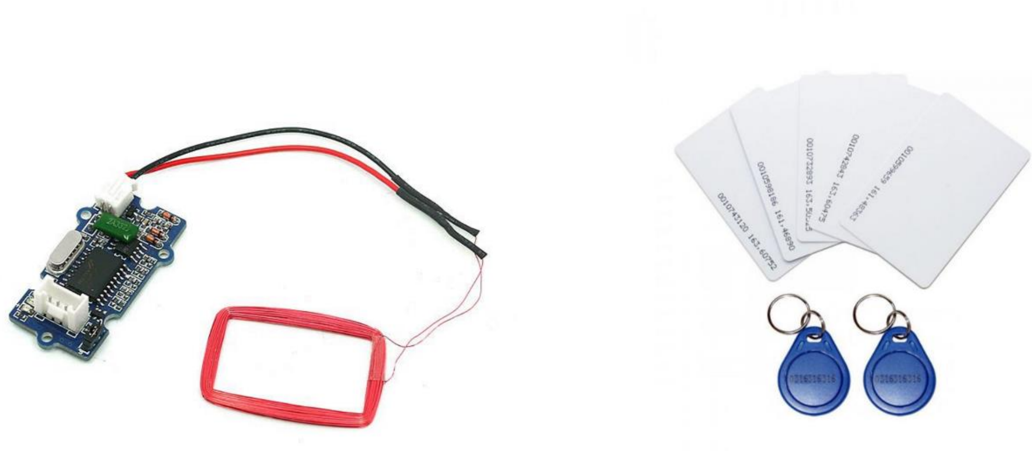


Figure 5: Grove 125Khz RFID Reader and Tag

The RFID reader primarily has three components. The first component is the RF signal generator, responsible for producing the radio waves that are subsequently sent via the antenna. The RFID reader is equipped with a receiver or signal detector to receive the feedback signal from the tag. In order to handle the data sent by the RFID tag, it is equipped with a microcontroller and is directly linked to a computer.

Regarding tags, the majority of regularly used tags are passive, meaning they lack their own power source. This is because passive tags are more cost-effective compared to active tags, which possess their own power supply. We use passive tags that are not only cost-effective but also small in size. These devices are available in several formats such as cards, key fobs, labels, and chips. However, we specifically used the card and key fob variations.

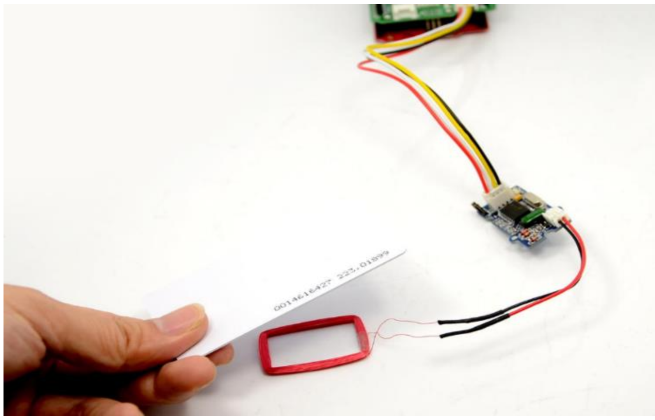


Figure 6: RFID Reader operation

The primary constituent of the RFID tag is the transponder, which receives radio waves emitted by the reader and transmits a response signal back to the reader. Passive tags lack an independent power source and instead depend on the radio waves emitted by the reader. A rectifier circuit is used to store the energy from the reader in a capacitor. This stored energy is then utilized as a power source for the controller and memory element in the RFID tag.

**The working principle of RFID:**

Every RFID tag is encoded with a unique identification at the time of production. When in close proximity to an RFID reader, the RFID tag detects and captures the radio-frequency signal emitted by the reader's antenna. The signal's energy triggers the activation of the RFID tag. The RFID tag utilises the energy obtained from the received signal to fuel its internal circuitry. Passive RFID tags lack an internal power source and depend only on energy obtained from the reader's signal. Active RFID tags are equipped with a battery, which enables them to have a greater range. In our project, we are using passive tags. The operational RFID tag transfers the data contained in its microchip back to the RFID reader. This dataset comprises the distinct identification, artwork particulars, and any other pertinent data. Here we used a distinctive identification. The RFID reader analyses the incoming data and may execute diverse operations depending on the specific application. Data is used in art preservation for the purpose of verifying inventory authenticity. RFID systems function in several frequency ranges, such as low-frequency (LF), high-frequency (HF), and ultra-high-frequency (UHF). We used the Grove 125Khz RFID system, which functions in the low frequency (LF) range.

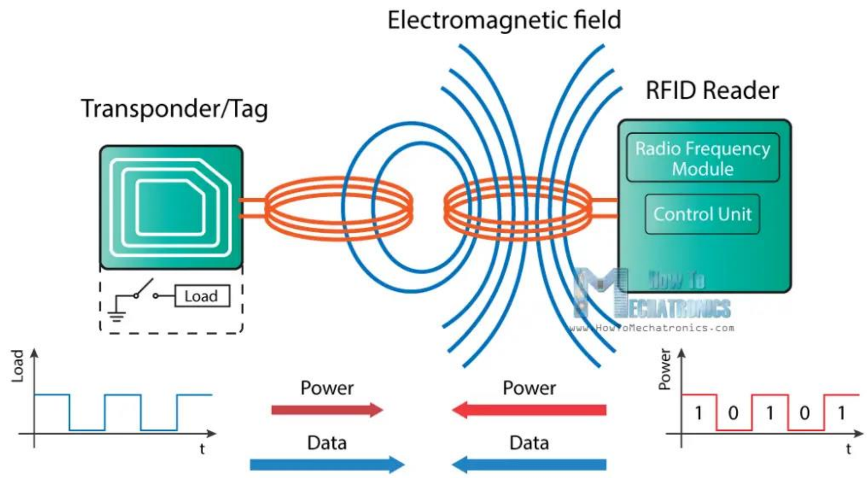



Figure 7: Working principle of RFID reader and tag

### 3.1.4. Dust Sensor

Artworks may experience progressive degradation due to the accumulation of dust on their surfaces. It is necessary to identify and monitor the amounts of dust in order for art conservators to create cleaning routines and protection measures that will minimize the effect on fragile surfaces. Dust sensors may serve as a vital asset in the preservation of art by monitoring and regulating environmental conditions. Particulate particles, such as dust, may have detrimental impacts on artworks and artefacts as time passes. We are using the grove dust sensor to safeguard the artwork.

Some of the key features and specifications are:

- **Particulate Matter Detection:** Determines the level of particulate matter (PM) in the atmosphere.
- **Optical Detection Method:** Usually use an optical technique to detect airborne dust particles.
- **Compact Design:** Intended to have a small size and be readily included in our developments.
- **Analog or Digital Output:** Offers either analogue or digital output, dependent upon the requirement.
- **Compatibility:** Intended to be interoperable with diverse microcontrollers and development platforms such as ESP32 and others.

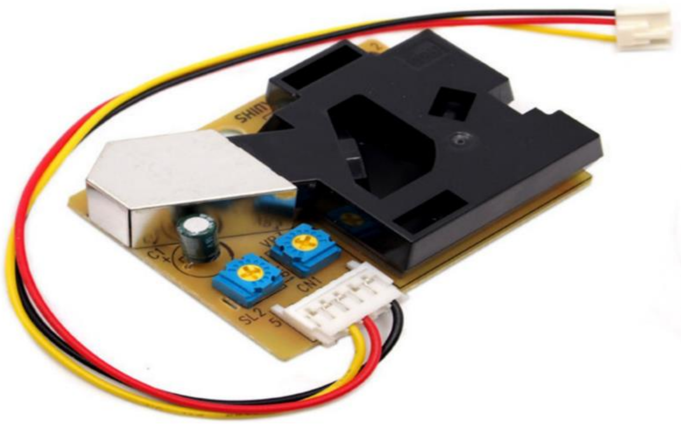


Figure 8: Grove Dust Sensor

- **Operating Voltage:** Operates within a certain voltage range, such as 5 volts.
- **Detection Range:** Defines the spectrum of particle sizes that the sensor is capable of detecting, such as PM1.0, PM2.5, and PM10.
- **Measurement Accuracy:** Quantifies the degree of precision in determining the concentration of particle matter.
- **Communication Interface:** The interface used for communication with microcontrollers, such as analogue, UART, or I2C.

**The working principle of the dust sensor:**

Dust sensors often use an infrared light-emitting diode (LED) to emit light into the surrounding air. When the light that is released contacts tiny particles in the air, such dust particles, it scatters. The degree of dispersion is affected by the density of particles. The sensor incorporates a photodiode or another light-sensitive component that detects the dispersed light. The quantity of diffused light that reaches the photodiode is directly proportional to the density of particulate matter in the atmosphere. The photodiode transforms the incoming light into an electrical signal. The magnitude of this signal correlates to the luminosity of light diffused by the dust particles. The sensor has the capability to provide either analogue or digital output. We are using analogue output in the project since it allows for a continuous range of values. The data obtained from the sensor may be analyzed to approximate the level of particulate matter present in the atmosphere.

The ESP32 can get the sensor's output and then analyze or send the data for analysis. Alerts or messages will be activated if the dust concentration is above pre-established criteria, encouraging the implementation of suitable measures.

### 3.1.5 Individual Sensor Board

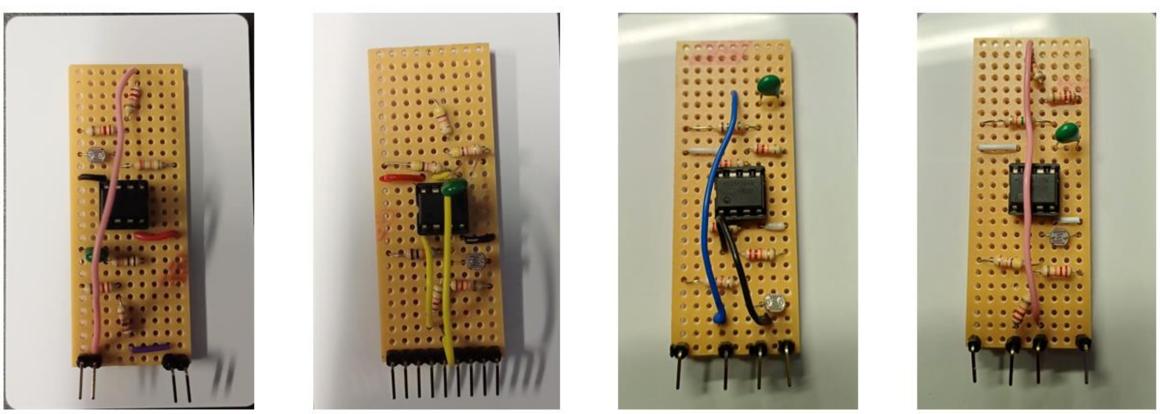
We crafted and soldered individual sensor boards for the temperature and light sensors, aiming to create a signal conditioning circuit. Employing a 20k thermistor, TO-18 photocell, and LM358-D Amplifier, we referenced the datasheets for the thermistor and photocell to compute the requisite resistance values for the circuit design. This calculation included determining the appropriate gain. To validate the circuit design, we conducted simulations using LT-Spice, enabling us to fine-tune resistor values (R1, R2, R3) based on the parameters of the sensors.


Figure 9: 20k Thermistor & TO-18 Photocell

An operational amplifier is not used alone but is designed to be connected to other circuits to perform a vast variety of operations. This article provides some typical examples of usage of circuits with operational amplifiers. Leveraging circuit designs optimized for Qua d Operational Amplifiers, these dual operational amplifiers possess notable characteristics such as low power consumption, a broad common-mode input voltage range extending to ground/VEE, and flexibility for single or split supply operation. The LM358 series offers similar functionality to one half of an LM324.


Figure 10: LM358 IC Pinout

In single supply applications, these amplifiers present distinct advantages over standard operational amplifier types. They operate within a wide range of supply voltages, from as low as 3.0 V to as high as 32 V, exhibiting quiescent currents approximately one-fifth of those associated with the MC1741 per amplifier. Notably, the common mode input range includes the negative supply, eliminating the need for external biasing components in numerous applications. Additionally, the output voltage range covers the negative power supply voltage, enhancing their versatility in various operational scenarios.



Sadhana Siddharth Shashank Spoorthy

Figure 11: Individual Sensor Boards

We incorporated the individual sensors into our project by utilizing the mentioned components. Our sensor boards were instrumental in integrating the temperature and light sensors, enabling us to measure the room temperature and light intensity. This monitoring allows us to assess the conditions where the artwork is situated. These sensor boards served as valuable tools, providing insights into sensor fabrication, calibration, and practical implementation in real-life applications.

We employed industry-standard instruments to calibrate both the temperature and light sensors, exposing them to varying environmental conditions like temperature fluctuations and changes in light density. This exposure allowed us to determine the relationship between the measured voltage and the respective physical quantities.

Subsequently, we calculated the necessary calibration coefficients based on these established relationships. These coefficients were integrated into the code, enabling the display of practical physical quantities—for instance, presenting a temperature reading of 20 degrees Celsius and a light intensity of 100 lux.

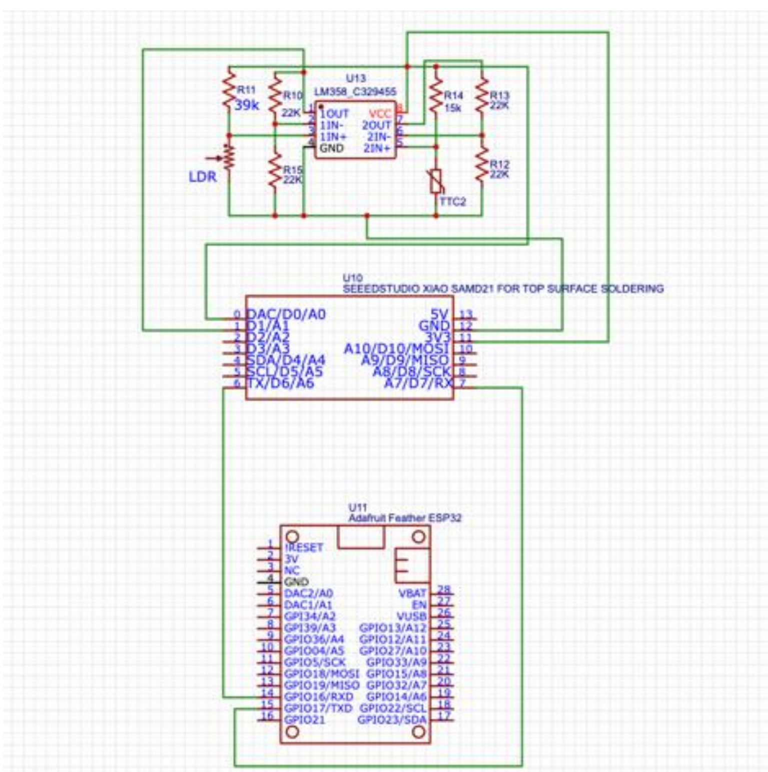



Figure 12: Sensor Board Integration - Schematic Diagram

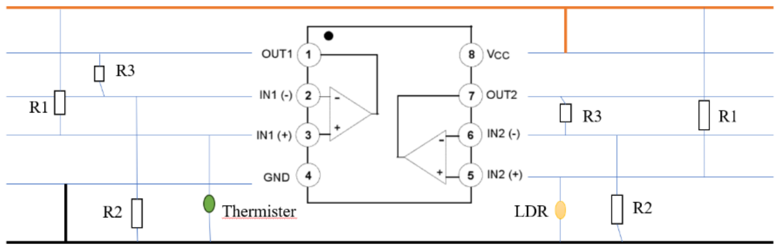



Figure 13: Individual sensor board pinout

In our project's storyline, we highlighted the significance of this calibration process and the meticulous establishment of connections between measured voltages and actual physical parameters. This emphasis highlighted the precision and reliability achieved through this calibration procedure.

### 3.1.6 Individual contribution for Sensor Calibration

To streamline the sensor calibration and interfacing process, we allocated tasks efficiently among team members to manage the project effectively. By distributing responsibilities, we aimed to alleviate the workload and ensure a more organized approach to sensor calibration and interfacing. This division of tasks enabled each team member to focus on specific aspects, contributing to a more efficient and coordinated project execution.

1. **Sadhana Ramu (Group Leader)** – Dust Sensor & Individual sensor board (Hardware and Software)

**2. Shashank Ramachandra Maidur -** Humidity Sensor & Individual sensor board (Hardware and Software)

**3. Siddharth Balu Pagare** – Humidity/ Individual sensor Board (Hardware)

**4. Spoorthy Matada Siddalingaswamy** – RFID Reader & Individual sensor board (Hardware and Software)

## 3.2 CRC, TDMA Communication, I2C, UART, IOT Connection

### 3.2.1 Cyclic Redundancy Check (CRC)

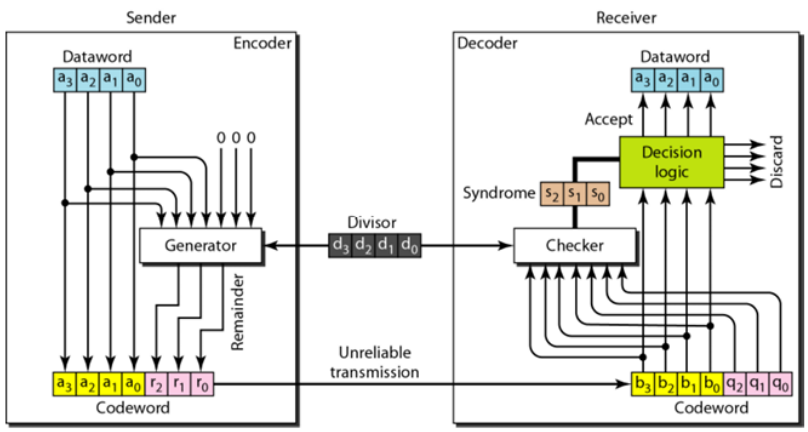



Figure 14: CRC flow

The cyclic redundancy check (CRC) method is a technique that may be used to identify errors in digital data. In order to perform a checksum-like function, the CRC creates a data set of a specified length, which is dependent on the creation of a file or a larger data set. CRC is a hash function that is used in digital telecommunications networks and storage systems such as hard disc drives in order to detect unintended alterations that have been made to raw computer data. As a component of the cyclic redundancy check, the message that has to be broadcast is augmented with a preset quantity of check bits, which are usually referred to as a checksum. Following the receipt of the data, the data receivers examine the check bits to confirm the absence of any mistakes.

A mathematical evaluation of the utilization that is attached is performed by data receivers by the calculation of the remaining polynomials division of the contents that have been sent. In the event that it looks that such an error has taken place, a negative acknowledgment is being provided, which requests that the data be resented. The length of the block that is anticipated to be protected has an impact on the designs of CRC polynomials. Error prevention procedures are another factor that may be used to select the CRC design. It is possible that the resources that are available for CRC implementation will have an effect on performance. After performing an XOR operation between each bit of the message and the code table, the CRC is computed and then returned.

This process is repeated until the CRC is calculated. In the event that the computed CRC is equal to zero, the CRC right is equal to one; otherwise, it corresponds to zero. The value of CRC right is equal to zero, which indicates that the message that was transferred reached the receiver end in an accurate manner.

```
};
// Function to calculate CRC-8
uint8_t calculateCRC8(const void* data, size_t length) {
    uint8_t crc = 0;
    uint8_t* buffer = (uint8_t*)data;

    for (size_t i = 0; i < length; i++) {
        crc = crc8_table[crc ^ buffer[i]];
    }

    return crc;
}
```

Figure 15: CRC calculation code

The code snippet demonstrates the CRC check within our sensors. In the provided image, the CRC starts as zero, and within the for-loop, the variable 'i' begins at zero and continues until it reaches the length of the transmitted message. The CRC calculation involves using the XOR operation between each bit of the message and the code table. After computation, the function returns the CRC value. If the calculated CRC equals zero, the variable CRC right is set to one; otherwise, it is set to zero. A CRC right value of one signifies that the transmitted message has been accurately received at the receiver's end.

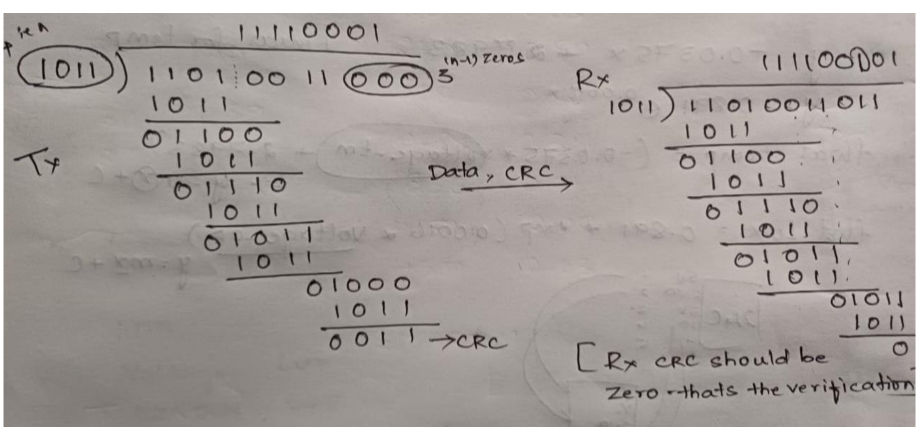



Figure 16: CRC calculation and Verification

### 3.2.2 I2C Protocol

One of the most important technologies in the field of electronics is the Inter-Integrated Circuit (I2C) communication protocol, which enables devices that are linked to one another to exchange data without any interruptions. Using a two-wire bus, the I2C protocol adheres to a master-slave architecture. This means that every device has its own distinct address, which allows for more precise communication. By using byte-oriented data transport and clock stretching for synchronization, the protocol is able to establish communication via the use of start and stop conditions. Reliability is improved via acknowledgment bits, which certify that data has been successfully received.

The effectiveness of I2C resides in its simplicity, which makes it suited for communication across small distances inside a common circuit board. Multi-master setups are supported by the protocol, which makes it possible for several controllers to cohabit in a peaceful manner. The integrity of the signal is preserved via pull-up resistors, which ensure that the lines are pushed high even when they are not being actively driven down.

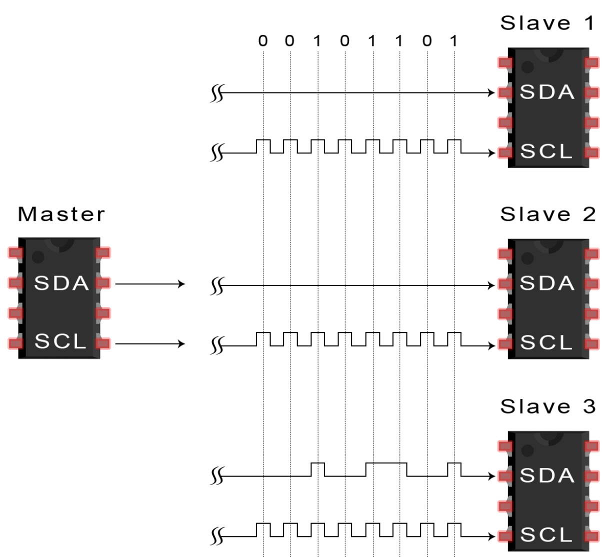



Figure 17: Representation of I2C

The data which is generated by the sensor module is transferred to ESP32 3 via I2C communication. I2C is a two-wire interface comprised of the signals serial data (SAD) and serial cloak (SCL).

In general, the lines are open-drain and bi-directional. In a generalized I2C interface implementation, attached devices can be a master or a slave. The master device puts the slave address on the bus, and the slave device with the matching address acknowledges the master.

### 3.2.3 UART Protocol

An essential serial communication protocol, known as UART (Universal Asynchronous Receiver/Transmitter), is used extensively in the field of electronics for the purpose of transferring and receiving data between various devices. In order to frame each data byte, the Universal Asynchronous Receiver and Transmitter (UART) uses start and stop bits. Additionally, in order to ensure synchronization, both communication devices agree on a same baud rate. The standard data frame is made up of eight bits, although it is also feasible to have configurations that include seven or nine bits. Devices are able to interact in either a one-way or two-way fashion because to the fact that UART enables both simplex and duplex communication. Because of its ease of use and adaptability, it is often selected for use in microcontroller-based systems, embedded applications, and devices that need point-to-point communication.

The universal asynchronous receiver (UART) is an essential component in the process of creating dependable serial communication across short or intermediate distances. It is widely used in applications such as RS-232 communication and wireless modules.

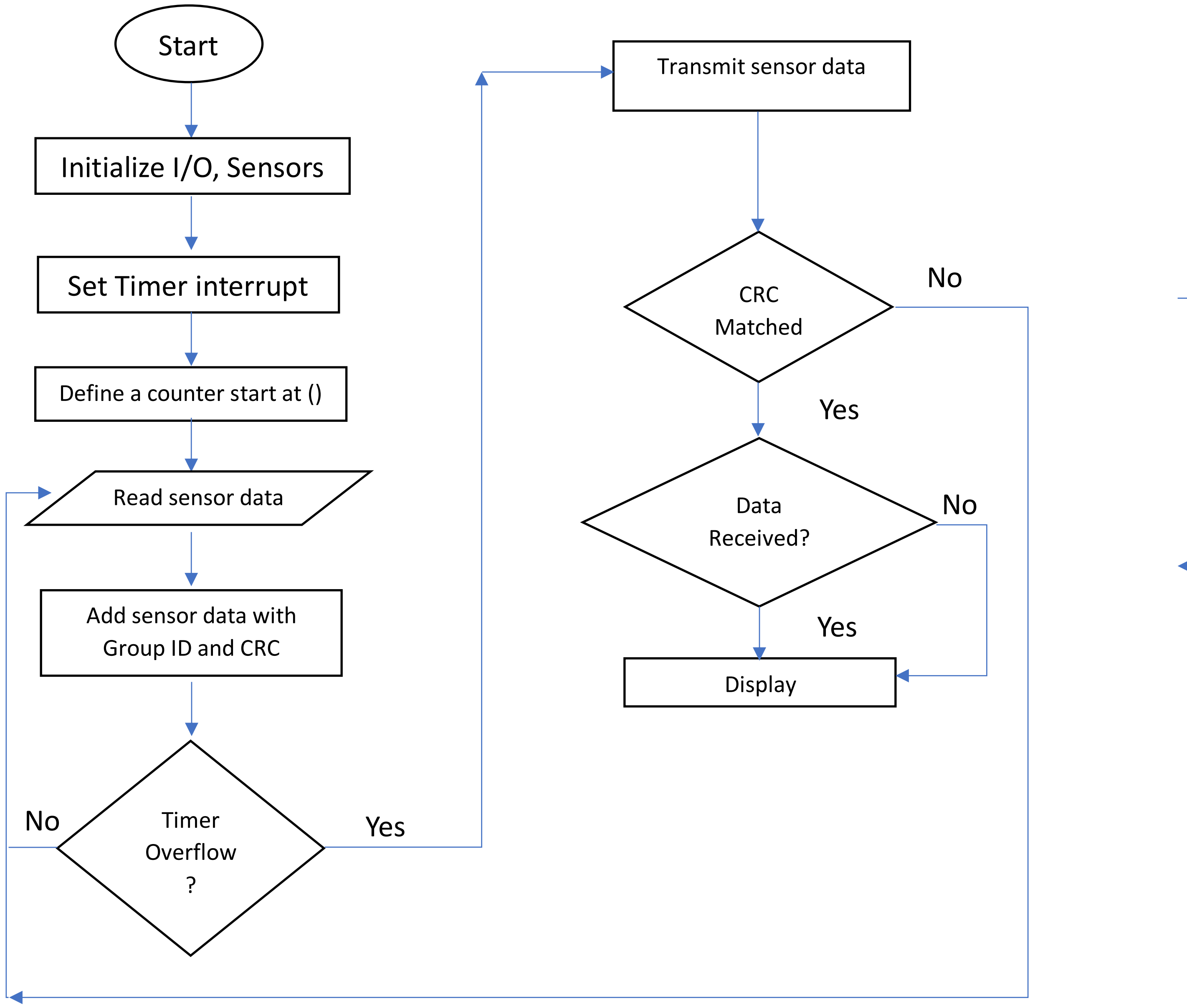
We have implemented UART communication protocol to both RFID and Dust Sensor.

### 3.2.4 Time Division Multiple Access (TDMA)

An example of a communication protocol that is frequently used in wireless networks is called Time Division several Access (TDMA). This protocol allows several users to share a shared frequency channel in an efficient manner. In time division multiple access (TDMA), the available time is segmented into slots, and each user is assigned a particular time slot for the purpose of data transmission. Users are able to share the same frequency without interfering with one another because to the sequential arrangement of time slots that has been provided. The time-division multiple access (TDMA) protocol is well-known for its time-efficiency. It enables several users to send and receive data inside the same channel, which contributes to the most efficient utilization of the available bandwidth.

The use of this protocol is widespread in cellular networks, satellite communication, and a variety of wireless technologies. Its utilization contributes to the enhancement of communication capacity and the reduction of interference in shared frequency spectrums.

### 3.2.5 Flow Chart



### 3.3 ESP32 as an IoT Gateway of WSNs

The ESP32 module can be configured to connect to Wi-Fi in either AP mode, where it acts like a router, or STA mode, which enables it to connect as a client to an existing Wi-Fi network. In AP mode, it functions as a standalone network device, while in STA mode, it connects to an established Wi-Fi network.

Wi-Fi networks are typically identified by human-readable names called SSIDs, reflecting the network owner's preferences. The ESP32 can check its connection status using the `WIFI. Status()` function. If the status consistently indicates an unconnected state, the code can be programmed to enter a loop, waiting until a successful Wi-Fi connection is established before proceeding.

```
void setup() {
  Serial.begin(BAUD_RATE);
  SerialPort.begin(BAUD_RATE, SERIAL_8N1, UART_RX_PIN, UART_TX_PIN);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  if(WiFi.status() != WL_CONNECTED){
      Serial.print("Attempting to connect Wi-Fi");
      while(WiFi.status() != WL_CONNECTED){
        WiFi.begin(ssid, password);
        delay(5000);
      }
```

Figure 18: ESP32 as an IOT Gateway

```
unsigned long myChannelNumber = 1; //The channel number in ThingSpeak (e.g.Channel 1)
const char* myWriteAPIKey = "9SIOLGEEJAB2EYP9"; //Write API Key, can be obtained from ThingSpeak.
```

To send data to ThingSpeak using an ESP32, you'll utilize `ThingSpeak.writeFields(ChannelNumber, APIKey)` to specify the channel and API key. This function returns a status value upon execution. A '200' status indicates a successful upload, while '404' signals an incorrect API key.

It's important to verify the correctness of the API key generated by ThingSpeak and differentiate between the writeAPI and readAPI, as they serve distinct purposes.

Furthermore, `ThingSpeak.setField(field, data)` is employed to assign data to specific fields within the channel.

## 3.4 Data Management and Visualization

In IoT applications, effective data management and visualization are crucial, providing users with insights into sensor data, alerts, and statistical analyses. In secure environments, like rooms protecting artifacts, it's essential to store RFID tag access data and monitor environmental sensor readings for optimal artifact protection.

### 3.4.1 Thingspeak

ThingSpeak is an IoT platform that enables users to collect, store, analyze, and visualize data from various IoT devices. It offers a range of functionalities:

1. Data Collection: ThingSpeak allows the collection of data from sensors, devices, and applications via RESTful API calls or IoT protocols like MQTT.

2. Data Storage: It provides a platform for storing and managing collected data in the form of channels. Each channel can store multiple fields of numeric data along with timestamps.

3. Visualization: ThingSpeak offers built-in tools for creating visualizations such as graphs, charts, and gauges. These visualizations help users understand and interpret data trends.

4. Processing and Analysis: Users can perform basic analysis on the collected data using MATLAB Analysis and MATLAB Visualizations. This allows for data preprocessing, filtering, and basic analytics.

5. Integration and Alerts: It integrates with various IoT devices and services, enabling real-time data updates and triggering alerts based on predefined conditions or thresholds.

6. Open API: ThingSpeak provides an open API, allowing users to access their data and interact with the platform programmatically.

Overall, ThingSpeak serves as a versatile IoT platform that simplifies data collection, storage, analysis, and visualization, making it popular among developers and hobbyists for various IoT projects.

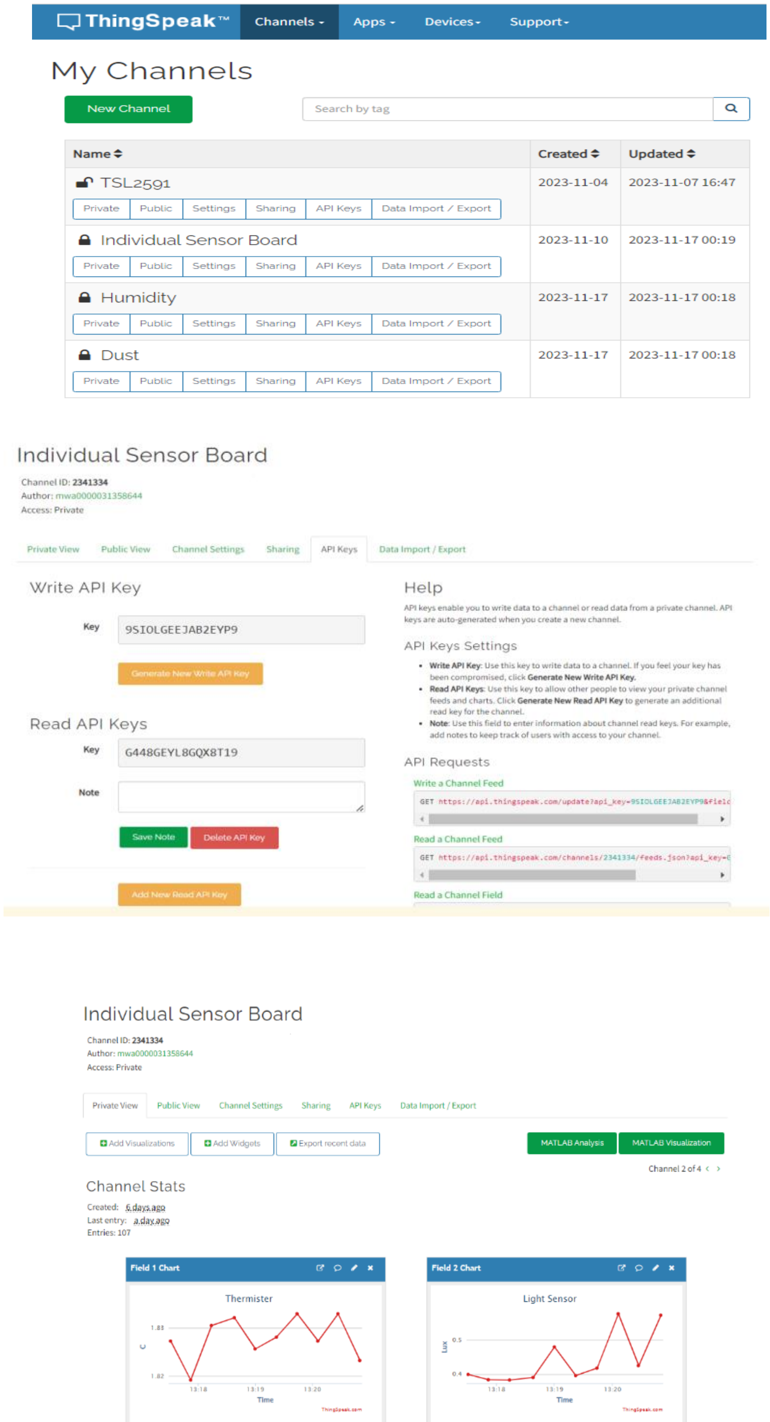
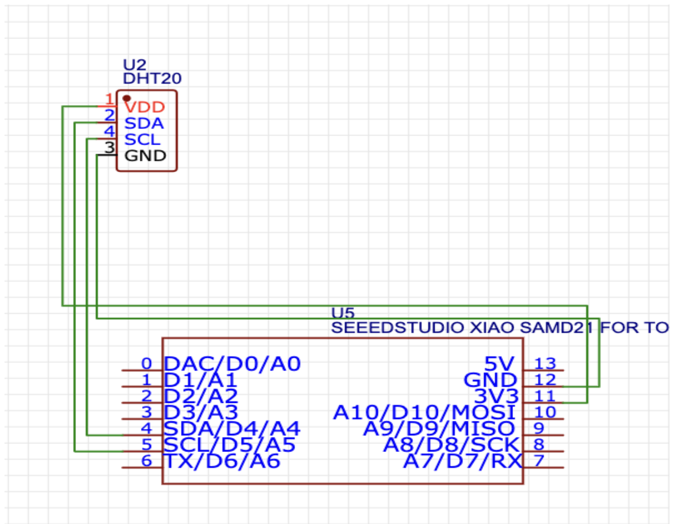


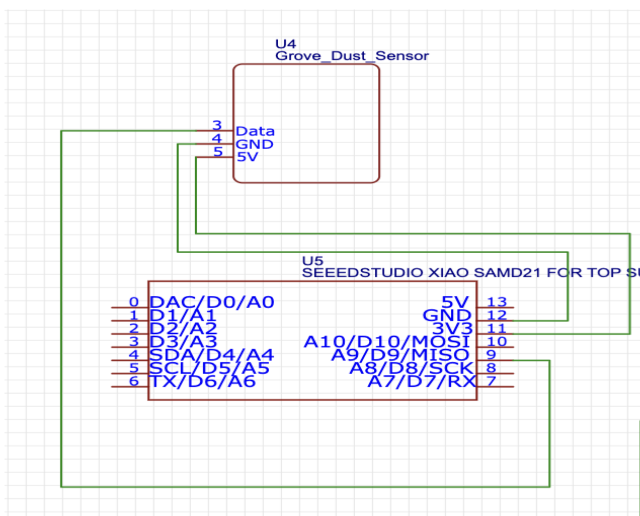

Figure 19: Thingspeak output

# Chapter 4 - Results and Discussion/explanation

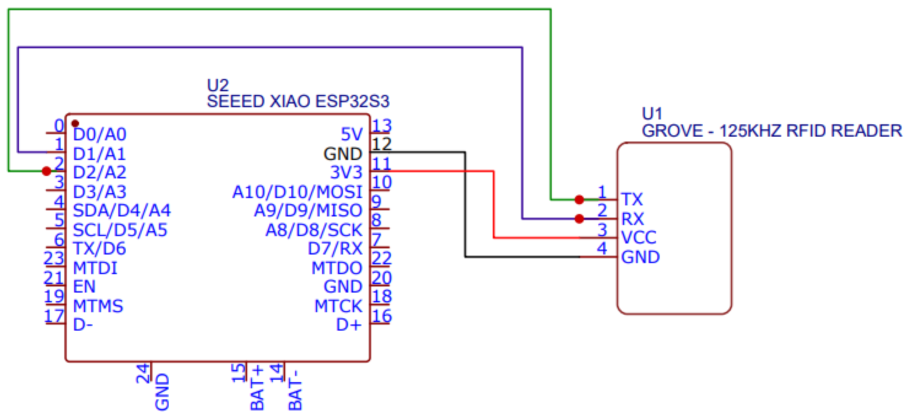
## 4.1.1 Schematic Diagrams



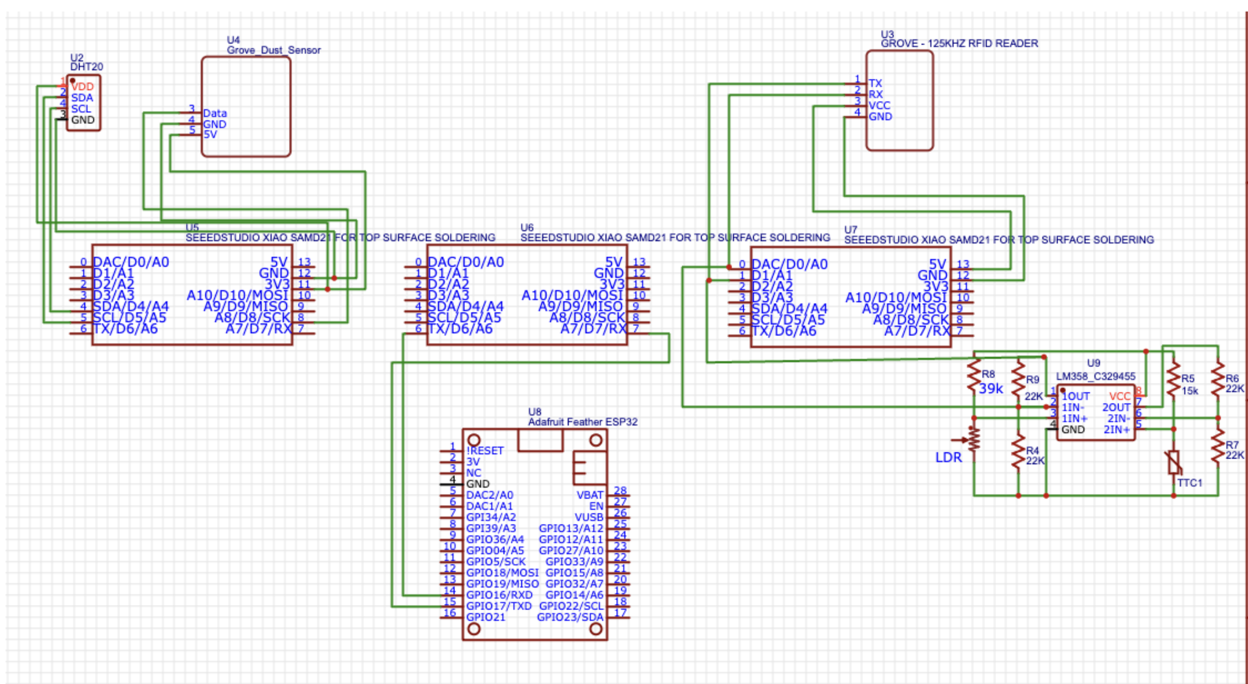

DHT20-Humidity Sensor interfaced with Xiao-ESP32S3



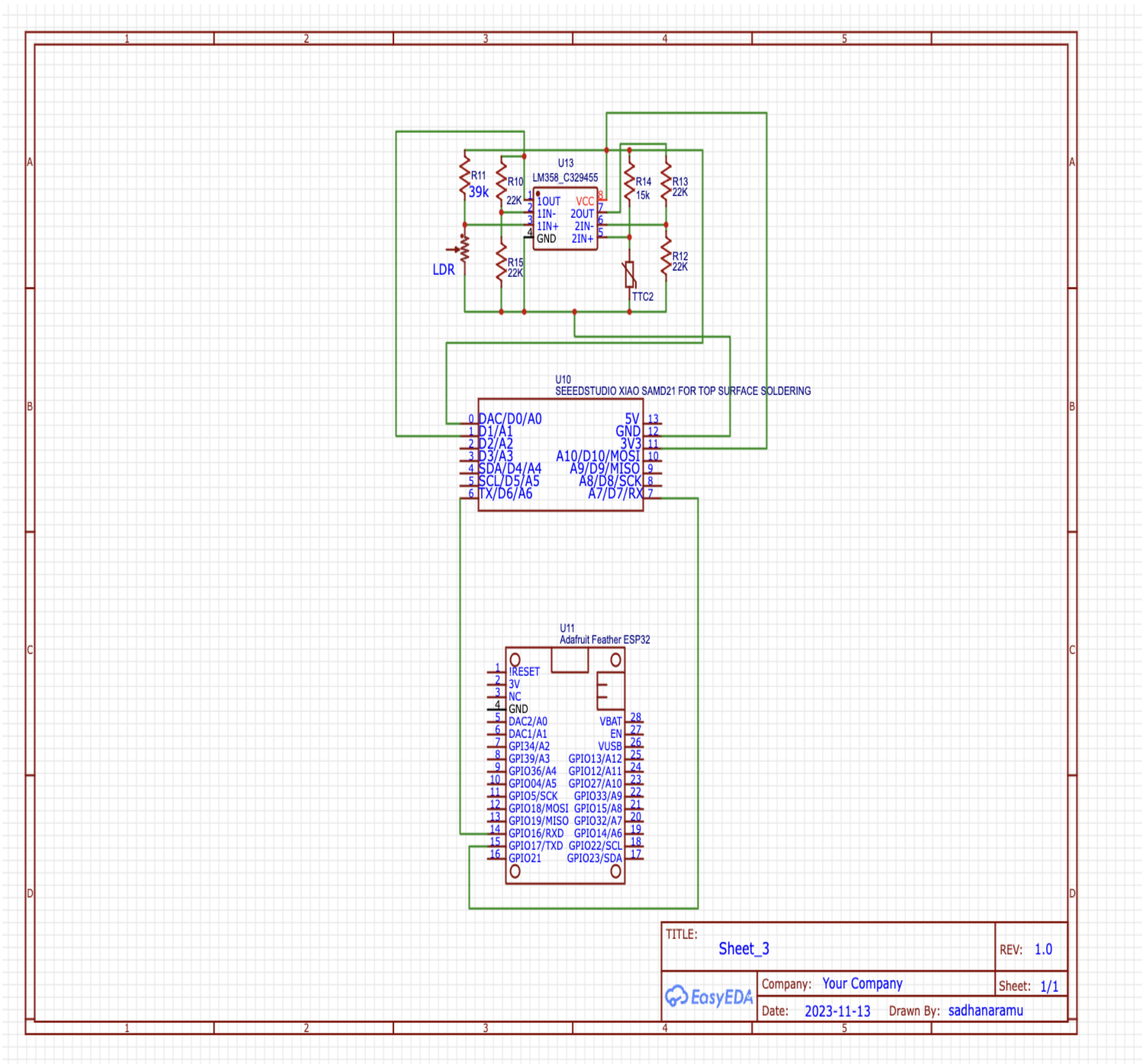

DHT20-Humidity Sensor interfaced with Xiao-ESP32S3




Grove 125KHz RFID Reader



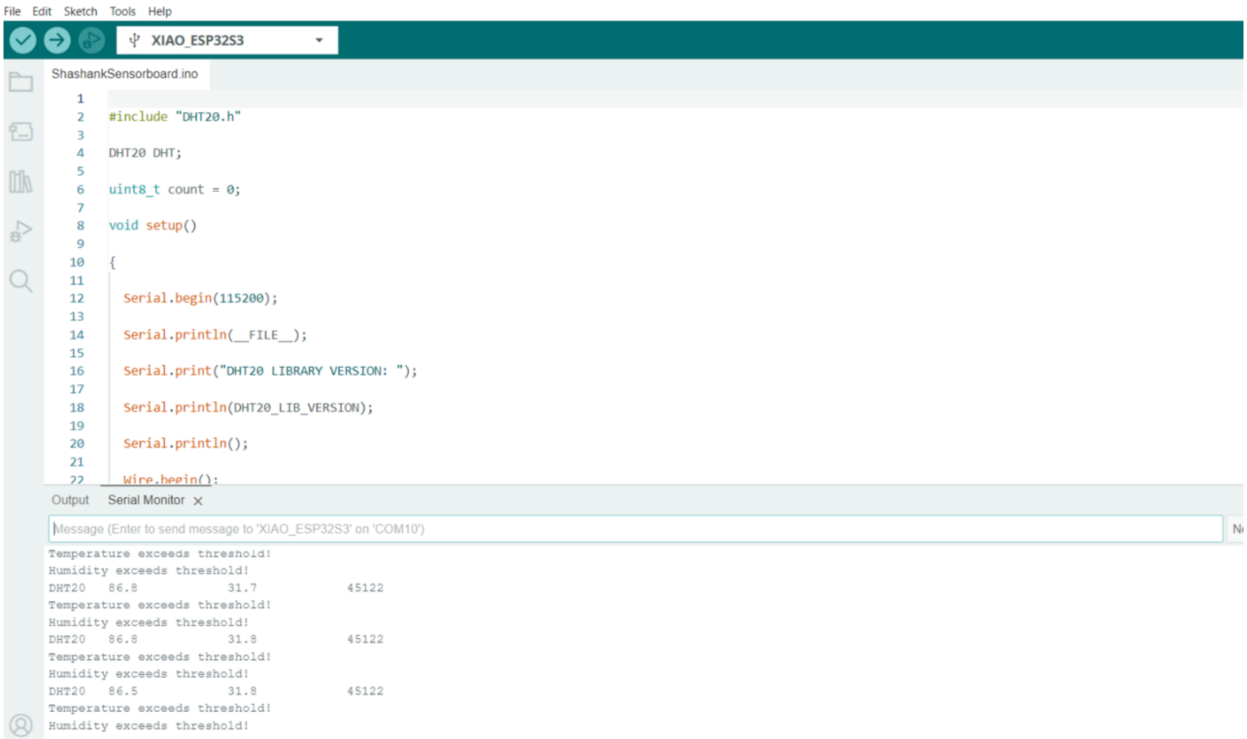

Full Schematic diagram of our project

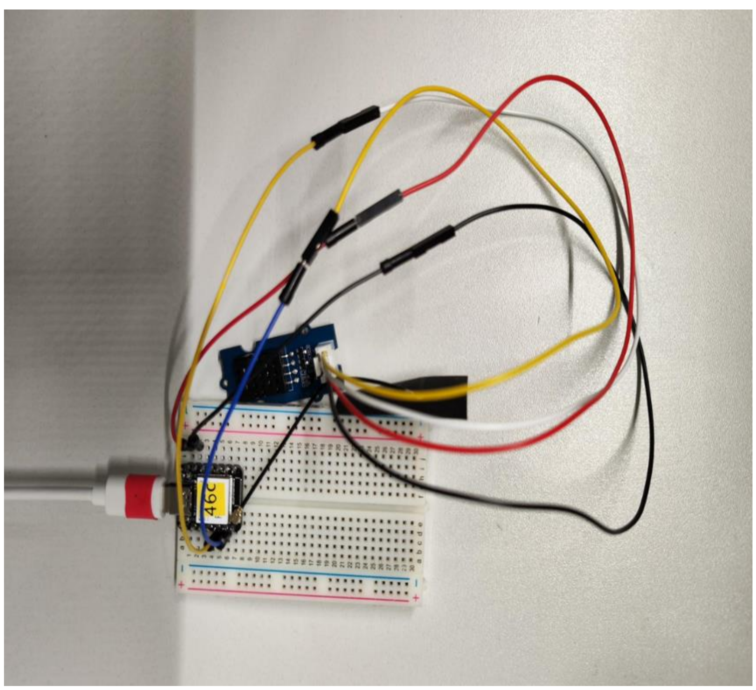



Individual Sensor Board Schematic

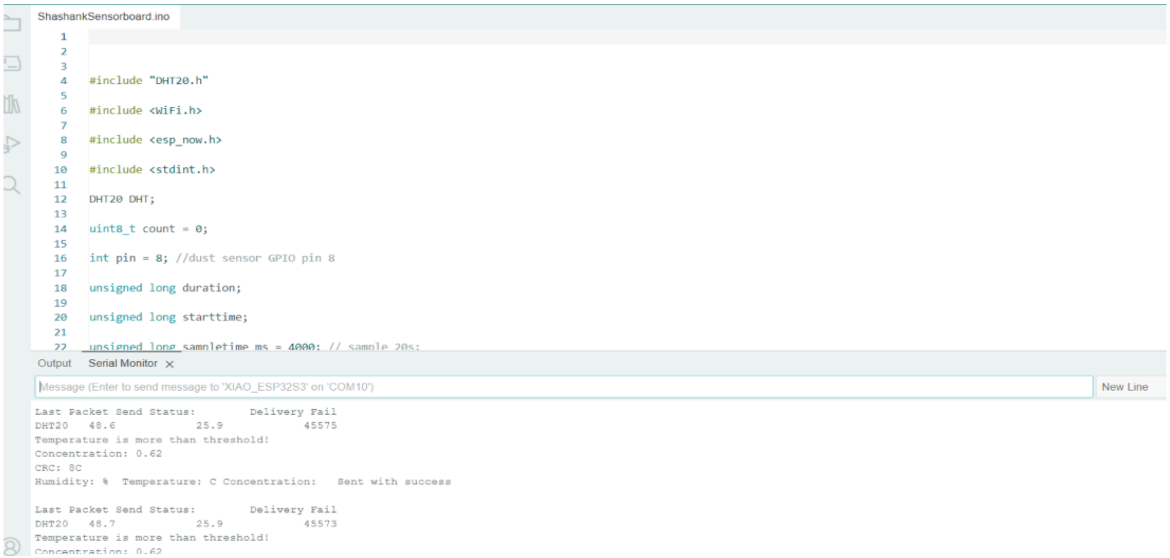
# 4.1.1 Sensor Board Output

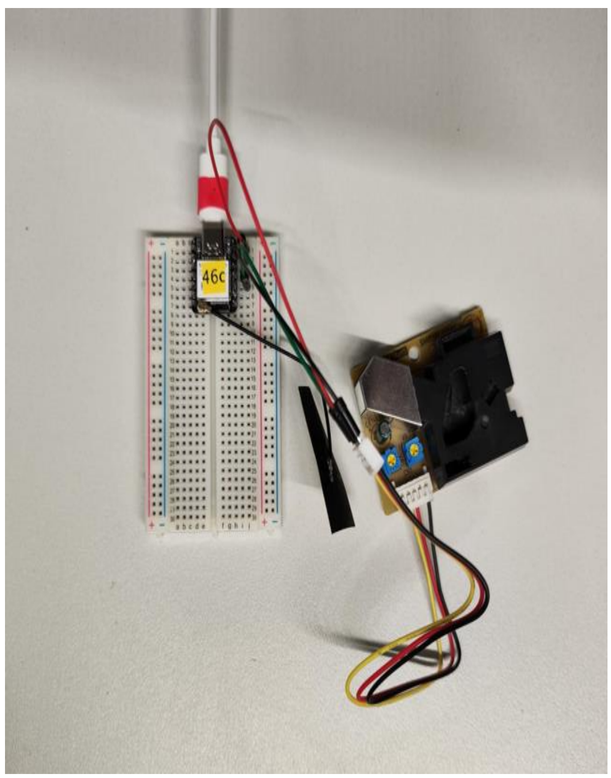
## DHT20-Humidity Sensor & its Hardware connection



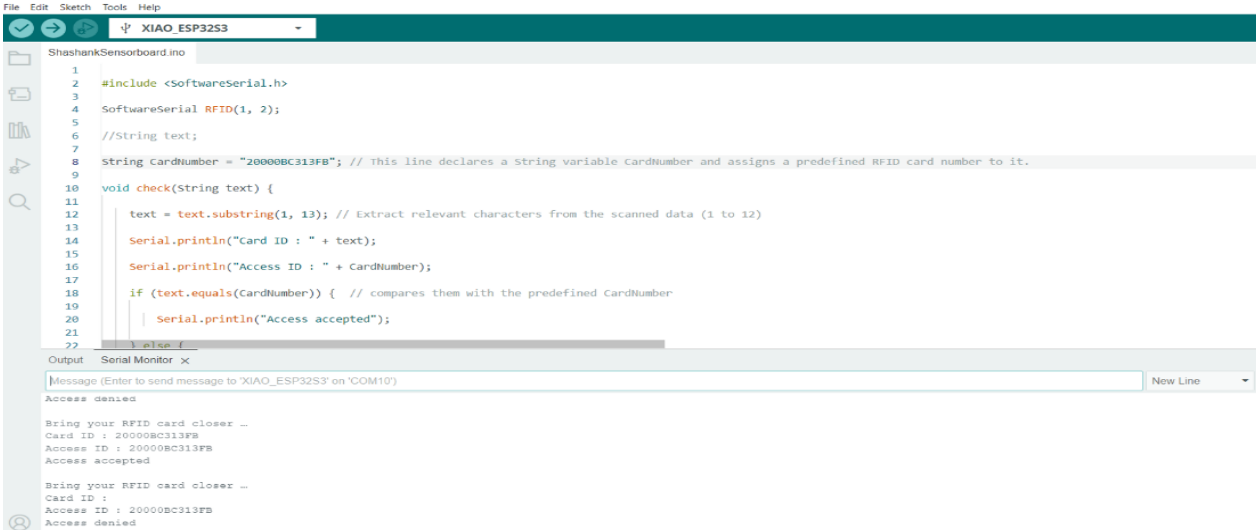



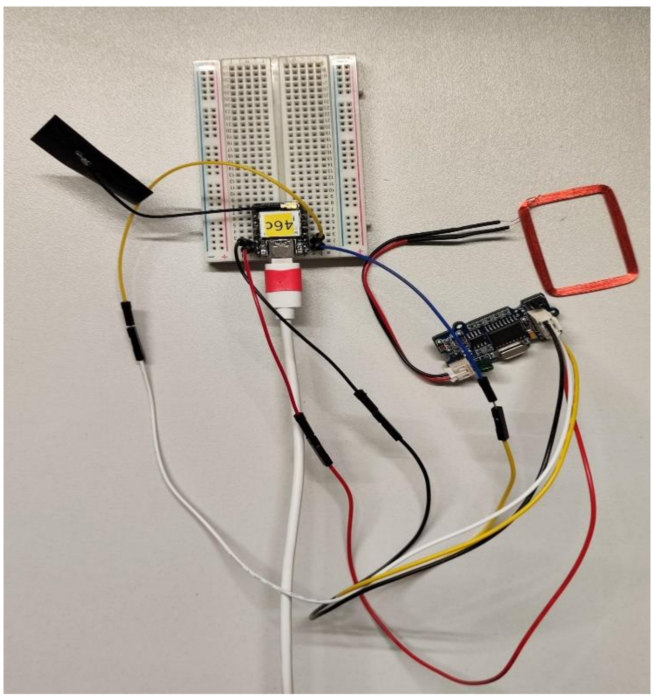
## Dust Sensor & its Hardware connection



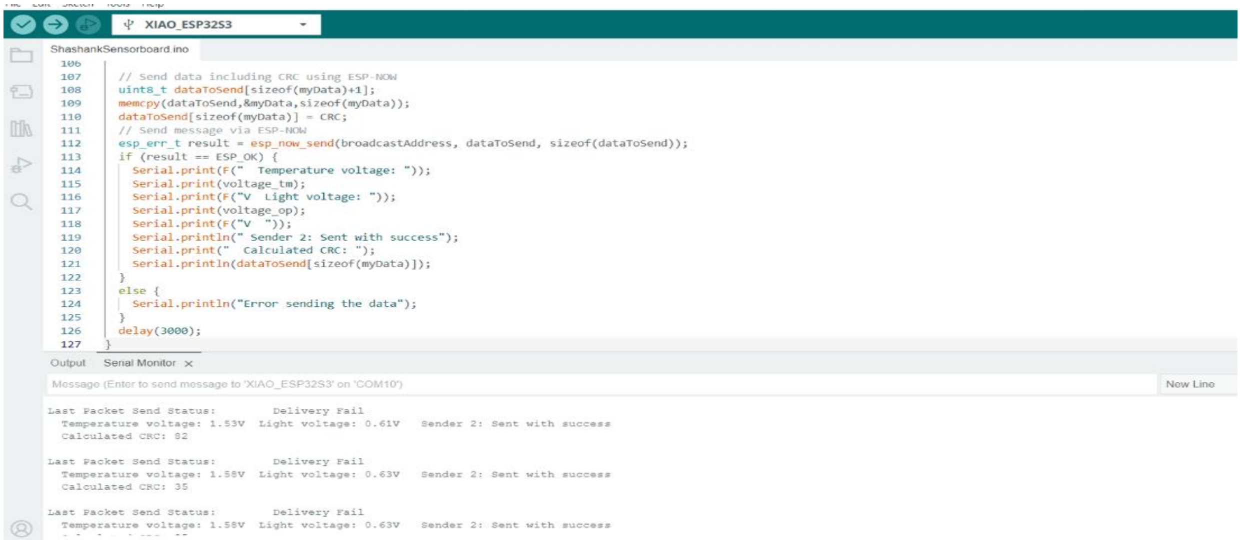



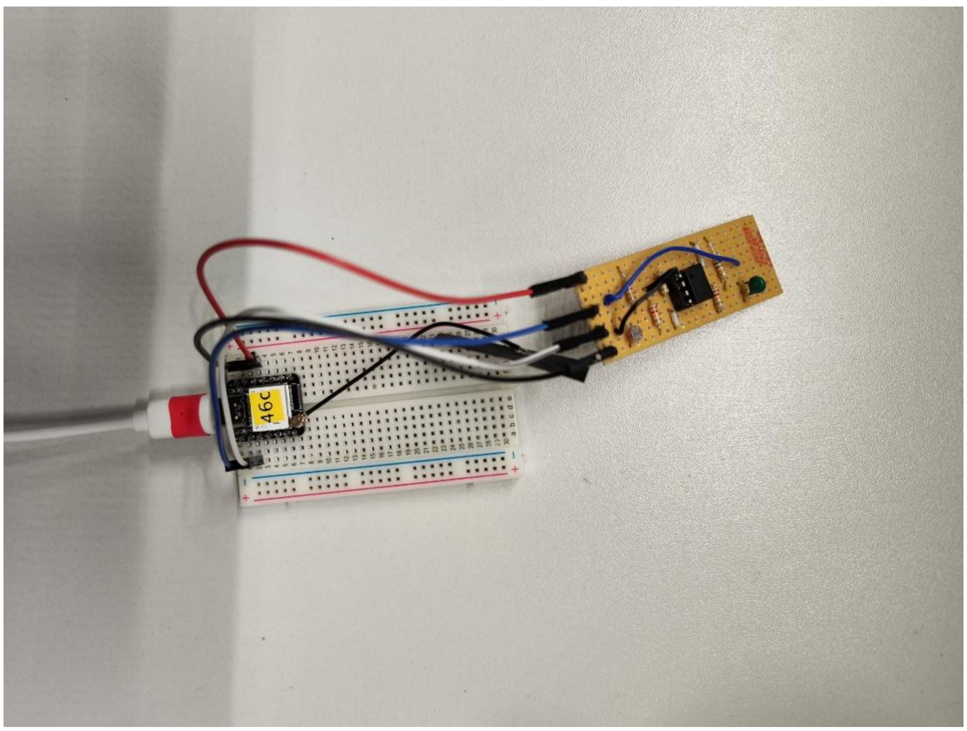
# RFID Reader &Tag & its Hardware connection



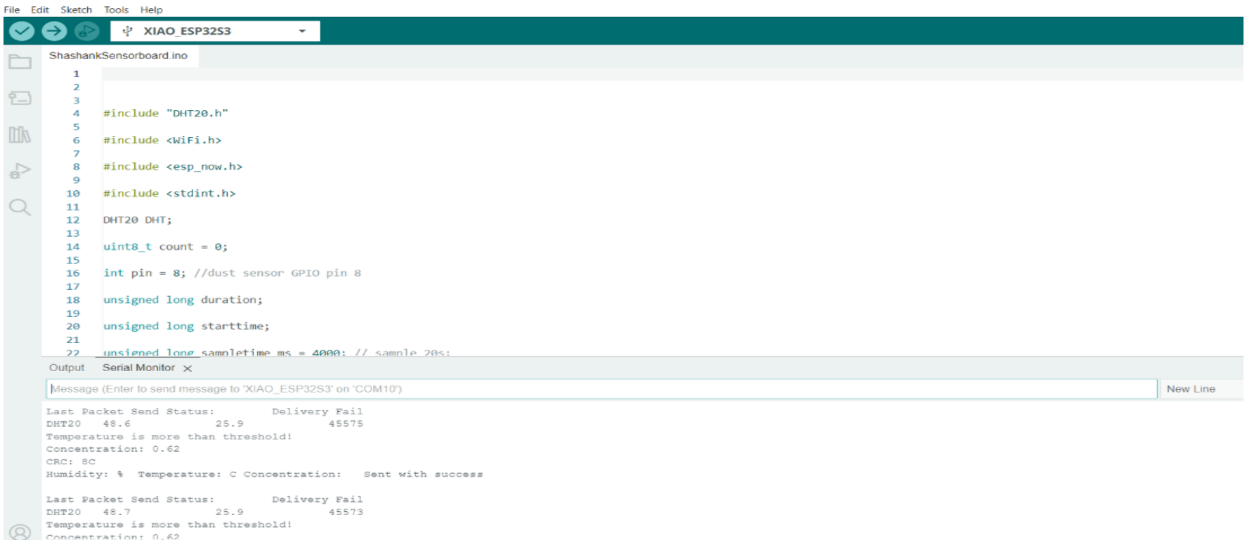



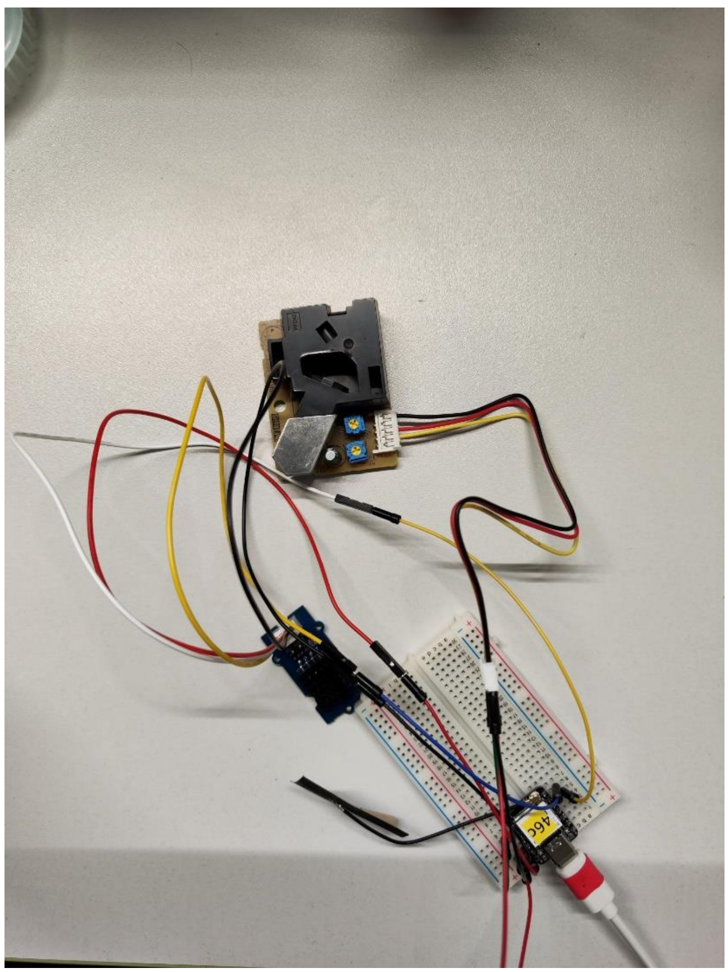
## Individual Sensor Board & its Hardware connection



## Humidity and Dust Sensor interface (Slave- A) & it’s Hardware connection



## RFID and Individual Sensor board interface(Slave- B) & it’s Hardware connection

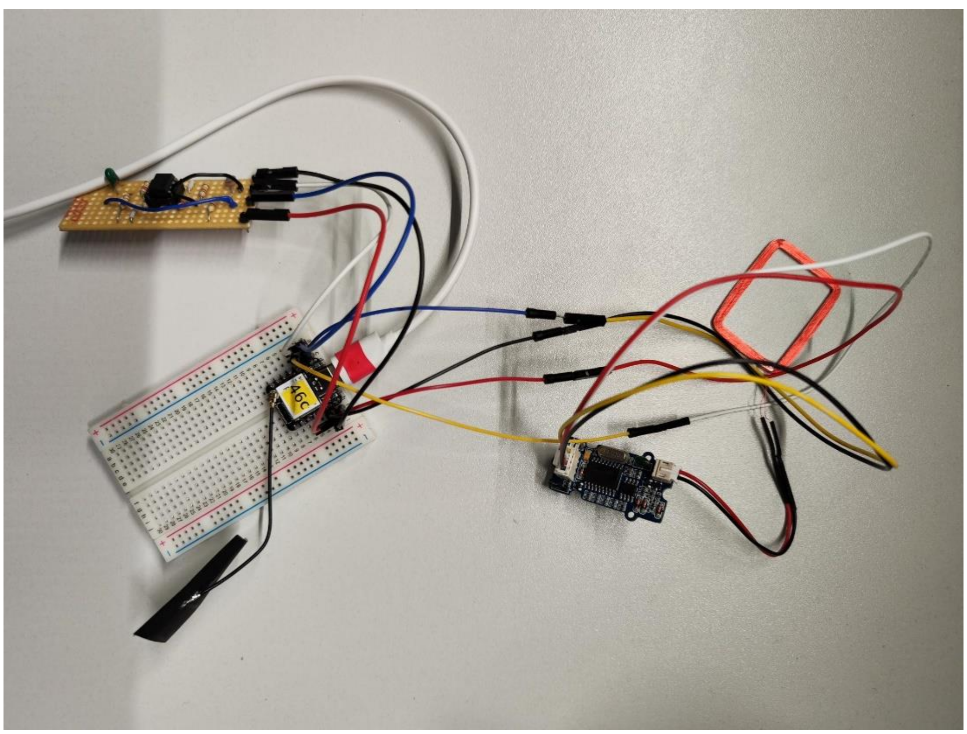
ShashankSensorboard.ino

```
#include <WiFi.h>
#include <esp_now.h>
#include <SoftwareSerial.h>
SoftwareSerial RFID(1, 2);
String CardNumber = "20000BC313FB"; // Predefined RFID card number
#define TempPin 3  // GPIO pin for temperature sensor
#define LightPin 4 // GPIO pin for light sensor
uint8_t broadcastAddress[] = {0x35, 0x85, 0x18, 0xAC, 0xC7, 0xB4}; // Replace with the MAC address of the receiver ESP32
typedef struct struct_message {
  int id;
  float temperature;
  float light;
  String rfid; // Change from char to String
} struct_message;
struct_message myData;
// CRC-8 lookup table for polynomial 0x8C (reverse of 0x31)
const uint8_t crc8_table[256] = {
  // ... (unchanged)
  0x00, 0x8C, 0x31, 0xBD, 0x45, 0xC9, 0x74, 0xF8, 0x8A, 0x06, 0xBB, 0x37, 0xCF, 0x43, 0xFE, 0x72,
    0x29, 0xA5, 0x18, 0x94, 0x6C, 0xE0, 0x5D, 0xD1, 0xA3, 0x2F, 0x92, 0x1E, 0xE6, 0x6A, 0xD7, 0x5B,
    0x52, 0xDE, 0x63, 0xEF, 0x17, 0x9B, 0x26, 0xAA, 0xD8, 0x54, 0xE9, 0x65, 0x9D, 0x11, 0xAC, 0x20,
    0x7B, 0xE7, 0x4A, 0xC6, 0x3F, 0xB2, 0x0F, 0x83, 0xF1, 0x7D, 0xC0, 0x4C, 0xB4, 0x38, 0x85, 0x09,
```

Output    Serial Monitor ×

Message (Enter to send message to 'XIAO_ESP32S3' on 'COM10')

```
Last Packet Send Status:        Delivery Success
Card ID : 20000BC313FB
Access ID : 20000BC313FB
RFID: Accepted
  Temperature voltage: 0.69V  Light voltage: 0.04V   Sender 2: Sent with success
RFID:   Calculated CRC: 233
  Sender 2: Sent with success  Calculated CRC: 233
```



# Chapter 5 - Conclusion including group & individual contribution, reflection and further work.

## 5.1 Group Work and Individual Contribution

Using the Internet of Things, the crew carried out the object conservation. The outcomes that were discussed before may be used for further and continued study of this project in the future.
The contribution that each individual student made to the project is detailed below for each student.

**1. Sadhana Ramu (Group Leader) - 230611159**

In the culmination of her collaborative efforts, she takes pride in leading the team through the intricate development phases of the project. The design phase required meticulous attention to detail, and her role involved steering the team through brainstorming sessions, encouraging innovative ideas, and ensuring that the proposed design met the project objectives. Collaborating with talented individuals brought a diversity of perspectives to the table, enriching the ideation process and resulting in a sensor design that is both robust and adaptable to various environments.

**2. Shashank Ramachandra Maidur - 230319598**

In the collaborative endeavour focused on humidity sensor hardware and software development, his individual contribution was instrumental in ensuring a seamless integration between the hardware and software components. Responsible for the calibration process, he meticulously fine-tuned the sensor's performance, contributing to the project's precision and reliability. Additionally, his involvement in rigorous testing procedures further validated the functionality of the humidity sensor, affirming its accuracy and robustness within the overall system. He joined the team effort in constructing and refining the report. He extends his gratitude to his team members for their support and collaboration throughout this project and looks forward in applying the knowledge gained in future endeavours.

**3. Siddharth Balu Pagare - 230340248**

He took charge of the hardware section of the project, overseeing both the hardware and software aspects while also managing report creation and preparing presentations for the group. His meticulous approach ensured the proper calibration and functionality of the sensor, striving for optimal output. Additionally, he led the programming efforts to enable seamless data transmission to ThingSpeak and skilfully interfaced the sensors with the IoT platform. His comprehensive involvement spanned from hardware handling to software intricacies, ensuring a cohesive integration between components and a robust system performance. His dedication to detail and holistic approach significantly contributed to the project's success, garnering appreciation from the team for his comprehensive contributions.

**4. Spoorthy Matada Siddalingaswamy - 230649244**

She took charge of both the hardware and software components in the development of an RFID reader and tag system, assumed a leadership role. She led the creation of firmware for the RFID reader, ensuring seamless communication with tags within the software domain. She is responsible for system architecture, the integration of RFID components, and overseeing prototype development and testing. Additionally, she actively contributed to the composition of the final report, showcasing a commitment to deadlines, transparent communication, and adept incorporation of feedback, ultimately playing a pivotal role in the production of a cohesive and high-quality report. She expresses her appreciation to her team members for their assistance and cooperation during this effort.

## 5.2 Further work

In the future, there are several opportunities for further development and improvements to the project focused on preserving artefacts. Conduct research and apply enhancements to wireless communication, with the goal of achieving increased efficiency and coverage. This may include investigating nascent technologies or protocols that augment communication between the ESP32 and other constituents. Implement real-time monitoring capabilities to provide instant alerts in case of adverse environmental conditions, this enables prompt intervention and minimizes potential damage to artifacts. Integrate automated control systems that can adjust environmental conditions based on sensor data. For example, if humidity levels rise beyond a certain threshold, the system could activate dehumidifiers or adjust ventilation to maintain optimal conditions. Design the system to be adaptable to different types of artifacts with varying preservation requirements. This may involve customizable settings and profiles to accommodate a diverse range of collections.

## 5.3 Conclusion

In conclusion, the artefact restoration effort symbolises a noteworthy achievement in our dedication to safeguarding cultural heritage for forthcoming generations. By using state-of-the-art technology, meticulous planning, and cooperative endeavours, we have successfully protected rare artefacts and spearheaded groundbreaking methods in the realm of conservation.

The implementation of sophisticated sensor networks, which consist of humidity, dust, and RFID sensors, has granted us a thorough comprehension of the environmental circumstances around these objects. The implementation of real-time monitoring has provided us with the ability to actively and pre-emptively mitigate any threats, therefore guaranteeing the preservation of these valuable cultural artefacts. The data obtained acts as evidence of our commitment and also offers a great resource for further study and conservation methods.

The use of the RFID technology has significantly transformed the way we oversee and monitor artefacts in our collection. The flawless incorporation of modern technology has not only improved security measures but also optimised inventory operations, equipping curators and scholars with effective tools to track artefacts along their complex paths.

As we contemplate the achievements of this project, it is essential to recognise the cooperative mindset that propelled our efforts. The integration of professionals from many fields like as conservation, technology, and project management has played a crucial role in our accomplishments. The insights gained from each team member have enhanced our comprehensive understanding of the complexities and potentialities in artefact preservation.

In the future, the insights acquired from this research will undeniably shape our next undertakings. Our goal is to continuously improve our methods by integrating new technology and best practises. This will guarantee that our cultural legacy is not only protected, but also accessible and valued by a wide range of people.

In expressing gratitude, acknowledgment is extended to all team members, collaborators, and stakeholders whose integral roles contributed significantly to the success of this artifact conservation project. The endeavour unfolds as a narrative marked by dedication, innovation, and a shared commitment to preserving our rich cultural heritage. As future conservation initiatives take shape, may this project stand as a testament to the potential achievable through the fusion of passion, expertise, and cutting-edge technology in safeguarding our historical treasures.

## References

[1] Research Paper: Cristina BALACEANU, Roxana ROȘCĂNEANU, Robert-Alexandru STRECHE, Filip-Emanuel OSIAC & George SUCIU, *“Artefacts conservation using an IoT system”*. Beia Consult International, Bucharest, Romania.

[2] H. Rushmeier, F. Samsel and J. Zhang, "Art and Cultural Heritage," in IEEE Computer Graphics and Applications, vol. 40, no. 3, pp. 17-18, 1 May-June 2020, doi: 10.1109/MCG.2020.2984927.

[3] EspressifSystems “ESP32 Series Datasheet” 2.4GHzWi-Fi+Bluetooth®+BluetoothLESoC, v4.3, 2023

[4] P. Gajendran, G. S. Setty, S. Vasanth, C. Ajay Kumar and R. Manoj Kumar, "IoT based Circuit Breaker with Access Control," 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS), Coimbatore, India, 2023, pp. 928-933, doi: 10.1109/ICSCSS57650.2023.10169319.

[5] H. Casas Sanchez, J. Loayza Apeña, J. Palomino and E. Paiva-Peredo, "Design of a people access control system using the ESP32 module and Internet of Things for a sanitory facility in a shopping mall," 2022 IEEE Engineering International Research Conference (EIRCON), Lima, Peru, 2022, pp. 1-4, doi: 10.1109/EIRCON56026.2022.9934094.

[6] D. Torre, A. Chennamaneni and A. Rodriguez, "Privacy-Preservation Techniques for IoT Devices: A Systematic Mapping Study," in IEEE Access, vol. 11, pp. 16323-16345, 2023, doi: 10.1109/ACCESS.2023.3245524.

[9] T. H. Nasution, A. Hizriadi, K. Tanjung, and F. Nurmayadi, "Design of Indoor Air Quality Monitoring Systems," 2020 4th International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM), Medan, Indonesia, 2020, pp. 238-241, doi: 10.1109/ELTICOM50775.2020.9230511.

[10] J. Shah and B. Mishra, "Customized IoT enabled Wireless Sensing and Monitoring Platform for Preservation of Artwork in Heritage Buildings," presented at the IEEE WiSPNET 2016 conference, DOI: 10.1109/WiSPNET.2016.

[11] E. Elhariri and S. A. Taie, "An Energy Efficient System for Artifacts Preservation and Occupant Comfort," in 2018 International Conference on Computer and Applications (ICCA), Fayoum, Egypt, 2018, 219. 987-1-5386-4371-6.

[12] G. Alsuhly and A. Khattab, "An IoT Monitoring and Control Platform for Museum Content Conservation," in 2018 International Conference on Computer and Applications (ICCA), Cairo University, Giza, Egypt, 2018, 197. 978-1-5386-4371-6/18/$31.00 c2018 IEEE.

[13] C. B. V. Prasanth, G. Sreekar, and J. Shah, "IoT Based Environment Monitoring System To Protect Heritage Artefacts," in 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT), Bengaluru, India, December 10-11, 2021, 145. 978-1-6654-3272-6/21/$31.00, IEEE.

[14] F. D'Amato, P. Gamba, and E. Goldoni, "Monitoring Heritage Buildings and Artworks with Wireless Sensor Networks," in Proceedings of the IEEE International Conference on [Conference Name], [Location], [Year], pp. [Page Numbers].

[15] C.-W. Kuo, J. K. Chiang, and Q.-P. Lin, "The Research on Applying RFID Information System Architecture for Museum Service," in 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology, Taipei, Taiwan, 2009, 218. 978-0-7695-3896-9/09, DOI 10.1109/ICCIT.2009.248.

[16] Website: https://wiki.seeedstudio.com/Grove_Sensor_Intro/ [ Date of access- 10 November 2023]

[17] ASAIR "Data Sheet DHT20", Humidity and Temperature Module, V1.0, May 2021.

[18] Website: Grove - 125KHz RFID Reader - Seeed Studio [ Date of access- 12 November 2023]

[19] Website: https://wiki.seeedstudio.com/Grove_Sensor_Intro/ [ Date of access- 11 November 2023]

[20] Website: Introducing DHT20 Temperature and Humidity Sensor and Comparing it with DHT11! - Latest Open Tech From Seeed (seeedstudio.com) [ Date of access- 15 November 2023]

# Appendix - Codes

## Dust Sensor Code

```
int pin = 8;
unsigned long duration;
unsigned long starttime;
unsigned long sampletime_ms = 30000;//sampe 30s ;
unsigned long lowpulseoccupancy = 0;

float ratio = 0;
float concentration = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(pin,INPUT);
  starttime = millis();//get the current time;
}

void loop()
{
  duration = pulseIn(pin, LOW);
  lowpulseoccupancy = lowpulseoccupancy+duration;
  if ((millis()-starttime) > sampletime_ms)//if the sampel time == 30s
  {
    ratio = lowpulseoccupancy/(sampletime_ms*10.0);  // Integer percentage 0=>100
    concentration = 1.1*pow(ratio,3)-3.8*pow(ratio,2)+520*ratio+0.62; // using spec sheet curve
    Serial.print(lowpulseoccupancy);
    Serial.print(",");
    Serial.print(ratio);
```

```
        Serial.print(",");
        Serial.println(concentration);

        lowpulseoccupancy = 0;
        starttime = millis();
    }
}
```

## RFID Reader Code

#include <SoftwareSerial.h> //This line includes the SoftwareSerial library, which allows you to create a virtual serial port on digital pins other than the hardware serial pins (usually 0 and 1).

// Define the RX and TX pins - These lines define the digital pins used for RX (Receive) and TX (Transmit) connections to the RFID reader.

int RXPin = 1;  // Replace with the actual RX pin (D2 - Yellow)

int TXPin = 2;  // Replace with the actual TX pin (D3 - White)

SoftwareSerial RFID(RXPin, TXPin);  // RX, TX - This line initializes a SoftwareSerial object named RFID with the specified RX and TX pins. This allows communication with the RFID module using these virtual serial pins.

void setup() { // The setup() function is called once when the Arduino starts. It initializes the hardware and software serial communication.

Serial.begin(9600); // Serial.begin(9600) initiates the hardware serial port for communication with the computer.

RFID.begin(9600); //  initializes the software serial port for communication with the RFID module.

Serial.println("RFID Reader Test"); // prints a message to the serial monitor, indicating that the RFID reader is being tested.

}

void loop() { //The loop() function is called repeatedly. Inside this loop, it checks if there is data available on the RFID module using RFID.available()

  if (RFID.available() > 0) { //If data is available

    char input = RFID.read(); // it reads a character from the RFID module using RFID.read()

    Serial.print(input); // prints it to the serial monitor using Serial.print(input)

  }

}

## RFID Tag validation Code

```
#include <SoftwareSerial.h>

SoftwareSerial RFID(1, 2);

//String text;

String CardNumber = "20000BC313FC"; // This line declares a String variable CardNumber and assigns a predefined RFID card number to it.


void check(String text) {

  text = text.substring(1, 13); // Extract relevant characters from the scanned data (1 to 12)

  Serial.println("Card ID : " + text);

  Serial.println("Access ID : " + CardNumber);

  if (text.equals(CardNumber)) {  // compares them with the predefined CardNumber

    Serial.println("Access accepted");

  } else {

    Serial.println("Access denied");

  }

  delay(2000);

  Serial.println(" ");

  Serial.println("Bring your RFID card closer …");

}

void setup() {     // The setup() function is called once when the Arduino starts. It initializes the serial communication with the computer and with the RFID module. It also prints an initial message to the serial monitor.

  Serial.begin(9600);

  RFID.begin(9600);

  Serial.println("Bring your RFID Card Closer...");

}

char data;

void loop() {

  String text = " "; // Clear the text variable at the beginning of each loop iteration // The loop() function is called repeatedly. It clears the text variable and then enters a loop where it reads
```

characters from the RFID module and appends them to the text variable until no more characters are available. After exiting the loop, it calls the check function with the collected RFID data for processing.

```
  while (RFID.available() > 0) {
    data = RFID.read();
    text += data;
  }
  check(text);
}
```

## DHT20 Humidity Sensor

```
#include "DHT20.h"

DHT20 DHT;

uint8_t count = 0;

void setup()
{
  Serial.begin(115200);
  Serial.println(__FILE__);
  Serial.print("DHT20 LIBRARY VERSION: ");
  Serial.println(DHT20_LIB_VERSION);
  Serial.println();

  Wire.begin();
  DHT.begin();    //  ESP32 default pins 21 22


  delay(1000);
}
void loop()
{
  if (millis() - DHT.lastRead() >= 1000)
  {
    // Read data from DHT20 sensor
    uint32_t start = micros();
    int status = DHT.read();
    uint32_t stop = micros();
```

```
// Print sensor readings and status
if ((count % 10) == 0)
{
  count = 0;
  Serial.println();
  Serial.println("Type\tHumidity (%)\tTemp (°C)\tTime (µs)\tStatus");
}
count++;

float humidity = DHT.getHumidity();
float temperature = DHT.getTemperature();

Serial.print("DHT20 \t");
Serial.print(humidity, 1);
Serial.print("\t\t");
Serial.print(temperature, 1);
Serial.print("\t\t");
Serial.print(stop - start);
Serial.print("\t\t");

// Print status based on the value of the 'status' variable
switch (status)
{
  // Cases for different status values
}
Serial.print("\n");

// Check temperature and humidity thresholds
float temperatureThreshold1 = 20.0; // Set your temperature threshold
```

```
    float humidityThreshold1 = 40.0;    // Set your humidity threshold

    if (temperature < temperatureThreshold1)
    {
      Serial.println("Temperature is less than threshold!");
      // Your action for exceeding temperature threshold goes here
    }

    if (humidity < humidityThreshold1)
    {
      Serial.println("Humidity is less than threshold!");
      // Your action for exceeding humidity threshold goes here
    }
    // Check temperature and humidity thresholds
    float temperatureThreshold2 = 22.0; // Set your temperature threshold
    float humidityThreshold2 = 60.0;    // Set your humidity threshold

      if (temperature > temperatureThreshold2)

    {
      Serial.println("Temperature exceeds threshold!");
      // Your action for exceeding temperature threshold goes here
    }
    if (humidity > humidityThreshold2)
    {
      Serial.println("Humidity exceeds threshold!");
      // Your action for exceeding humidity threshold goes here
  }
  }
  }
```

## Individual Sensor Board Code

```
#include <WiFi.h>
#include <esp_now.h>

/* Read Temperature and Light Sensor*/
#include <stdint.h>
#define TempPin 1  //GPIO pin, check the pin number an dfunction of your board
#define LightPin 2

// Replace with the MAC address of the receiver ESP32S3 board
uint8_t broadcastAddress[] = {0x34, 0x85, 0x18, 0xAC, 0xC7, 0x60}; //34:85:18:AC:B4:2C Mac Address for Board 24a

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
  // char a[32];
  int id;
  float b;
  float c;
  // uint8_t crc; //CRC for error checking
  // bool d;
} struct_message;

/*------------------ CRC-8 Calculation-----------------------*/
// CRC-8 lookup table for polynomial 0x8C (reverse of 0x31)
const uint8_t crc8_table[256] = {
    0x00, 0x8C, 0x31, 0xBD, 0x45, 0xC9, 0x74, 0xF8, 0x8A, 0x06, 0xBB, 0x37, 0xCF, 0x43, 0xFE, 0x72,
    0x29, 0xA5, 0x18, 0x94, 0x6C, 0xE0, 0x5D, 0xD1, 0xA3, 0x2F, 0x92, 0x1E, 0xE6, 0x6A, 0xD7, 0x5B,
```

```
    0x52, 0xDE, 0x63, 0xEF, 0x17, 0x9B, 0x26, 0xAA, 0xD8, 0x54, 0xE9, 0x65, 0x9D, 0x11, 0xAC, 0x20,
    0x7B, 0xF7, 0x4A, 0xC6, 0x3E, 0xB2, 0x0F, 0x83, 0xF1, 0x7D, 0xC0, 0x4C, 0xB4, 0x38, 0x85, 0x09,
    0xA4, 0x28, 0x95, 0x19, 0xE1, 0x6D, 0xD0, 0x5C, 0x2E, 0xA2, 0x1F, 0x93, 0x6B, 0xE7, 0x5A, 0xD6,
    0x8D, 0x01, 0xBC, 0x30, 0xC8, 0x44, 0xF9, 0x75, 0x07, 0x8B, 0x36, 0xBA, 0x42, 0xCE, 0x73, 0xFF,
    0xF6, 0x7A, 0xC7, 0x4B, 0xB3, 0x3F, 0x82, 0x0E, 0x7C, 0xF0, 0x4D, 0xC1, 0x39, 0xB5, 0x08, 0x84,
    0xDF, 0x53, 0xEE, 0x62, 0x9A, 0x16, 0xAB, 0x27, 0x55, 0xD9, 0x64, 0xE8, 0x10, 0x9C, 0x21, 0xAD,
    0x91, 0x1D, 0xA0, 0x2C, 0xD4, 0x58, 0xE5, 0x69, 0x1B, 0x97, 0x2A, 0xA6, 0x5E, 0xD2, 0x6F, 0xE3,
    0xB8, 0x34, 0x89, 0x05, 0xFD, 0x71, 0xCC, 0x40, 0x32, 0xBE, 0x03, 0x8F, 0x77, 0xFB, 0x46, 0xCA,
    0xC3, 0x4F, 0xF2, 0x7E, 0x86, 0x0A, 0xB7, 0x3B, 0x49, 0xC5, 0x78, 0xF4, 0x0C, 0x80, 0x3D, 0xB1,
    0xEA, 0x66, 0xDB, 0x57, 0xAF, 0x23, 0x9E, 0x12, 0x60, 0xEC, 0x51, 0xDD, 0x25, 0xA9, 0x14, 0x98,
    0x21, 0xAD, 0x10, 0x9C, 0x64, 0xE8, 0x55, 0xD9, 0xAB, 0x27, 0x9A, 0x16, 0xEE, 0x62, 0xDF, 0x53,
    0x08, 0x84, 0x39, 0xB5, 0x4D, 0xC1, 0x7C, 0xF0, 0x82, 0x0E, 0xB3, 0x3F, 0xC7, 0x4B, 0xF6, 0x7A,
    0x73, 0xFF, 0x42, 0xCE, 0x36, 0xBA, 0x07, 0x8B, 0xF9, 0x75, 0xC8, 0x44, 0xBC, 0x30, 0x8D, 0x01,
    0x5A, 0xD6, 0x6B, 0xE7, 0x1F, 0x93, 0x2E, 0xA2, 0xD0, 0x5C, 0xE1, 0x6D, 0x95, 0x19, 0xA4, 0x28
};

// Function to calculate CRC-8
uint8_t calculateCRC8(const void* data, size_t length) {
    uint8_t crc = 0;
    uint8_t* buffer = (uint8_t*)data;

    for (size_t i = 0; i < length; i++) {
```

```
        crc = crc8_table[crc ^ buffer[i]];
    }

    return crc;
}
// Create a struct_message called myData
struct_message myData;
esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {
  // Init Serial Monitor
  Serial.begin(115200);
  pinMode(TempPin,INPUT);
  pinMode(LightPin,INPUT);
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
```

```
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  // Set values to send
  // Reading temperature or humidity takes about 250 milliseconds!
  int sensorValue_tm = analogRead(TempPin);
  int sensorValue_op = analogRead(LightPin);
  // Convert the analog reading ADC:12bit (which goes from 0 - 4095) to a voltage (0 - 5V):
  float voltage_tm = sensorValue_tm * (3.3 / 4096.0);
  float voltage_op = sensorValue_op * (3.3 / 4096.0);
  float temp = (-26.312* voltage_tm ) + 72.839;
  float lux = 1.1705 * exp(0.0008 * voltage_op);

  // s1;trcpy(myData.a, "This is Sender1-59A ");
  myData.id=2;
  myData.b = voltage_tm;
```

```
  myData.c = voltage_op;

  // Calculate CRC
  uint8_t CRC =calculateCRC8(&myData, sizeof(myData));

  // Send data including CRC using ESP-NOW
  uint8_t dataToSend[sizeof(myData)+1];
  memcpy(dataToSend,&myData,sizeof(myData));
  dataToSend[sizeof(myData)] = CRC;
  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, dataToSend, sizeof(dataToSend));

  if (result == ESP_OK) {
    Serial.print(F("  Temperature voltage: "));
    Serial.print(voltage_tm);
    Serial.print(F("V  Light voltage: "));
    Serial.print(voltage_op);
    Serial.print(F("V  "));
    Serial.println(" Sender 2: Sent with success");
    Serial.print("  Calculated CRC: ");
    Serial.println(dataToSend[sizeof(myData)]);
  }
  else {
    Serial.println("Error sending the data");
  }
  delay(3000);
}
```

## Dust and Humidity Sensors integrated together with CRC - Slave A

```
#include "DHT20.h"
#include <WiFi.h>
#include <esp_now.h>
#include <stdint.h>

DHT20 DHT;

uint8_t count = 0;

int pin = 8; //dust sensor GPIO pin 8
unsigned long duration;
unsigned long starttime;
unsigned long sampletime_ms = 4000; // sample 20s;
unsigned long lowpulseoccupancy = 0;
float ratio = 0;
float concentration = 0;
int threshold = 500; // Set your threshold value

// Replace with the MAC address of the receiver ESP32S3 board
uint8_t broadcastAddress[] = {0x34, 0x85, 0x18, 0x91, 0x30, 0xA4};

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
  int id;
  float b;
  float c;
  float a;
```

```
} struct_message;

// CRC-8 lookup table for polynomial 0x8C (reverse of 0x31)
const uint8_t crc8_table[256] = {
    0x00, 0x8C, 0x94, 0x18, 0xA4, 0x28, 0x30, 0xBC, 0xC4, 0x48, 0x50, 0xDC, 0x60, 0xEC, 0xF4, 0x78,
    0x04, 0x88, 0x90, 0x1C, 0xA0, 0x2C, 0x34, 0xB8, 0xC0, 0x4C, 0x54, 0xD8, 0x64, 0xE8, 0xF0, 0x7C,
    0x08, 0x84, 0x9C, 0x10, 0xAC, 0x20, 0x38, 0xB4, 0xCC, 0x40, 0x58, 0xD4, 0x68, 0xE4, 0xFC, 0x70,
    0x0C, 0x80, 0x98, 0x14, 0xA8, 0x24, 0x3C, 0xB0, 0xC8, 0x44, 0x5C, 0xD0, 0x6C, 0xE0, 0xF8, 0x74,
    0x10, 0x9C, 0x84, 0x08, 0xB4, 0x38, 0x20, 0xAC, 0xD4, 0x58, 0x40, 0xCC, 0x70, 0xFC, 0xE4, 0x68,
    0x14, 0x98, 0x80, 0x0C, 0xB0, 0x3C, 0x24, 0xA8, 0xD0, 0x5C, 0x44, 0xC8, 0x74, 0xF8, 0xE0, 0x6C,
    0x18, 0x94, 0x8C, 0x00, 0xBC, 0x30, 0x28, 0xA4, 0xDC, 0x50, 0x48, 0xC4, 0x78, 0xF4, 0xEC, 0x60,
    0x1C, 0x90, 0x88, 0x04, 0xB8, 0x34, 0x2C, 0xA0, 0xD8, 0x54, 0x4C, 0xC0, 0x7C, 0xF0, 0xE8, 0x64,
    0x20, 0xAC, 0xB4, 0x38, 0x84, 0x08, 0x10, 0x9C, 0xE4, 0x68, 0x70, 0xFC, 0x40, 0xCC, 0xD4, 0x58,
    0x24, 0xA8, 0xB0, 0x3C, 0x80, 0x0C, 0x14, 0x98, 0xE0, 0x6C, 0x74, 0xF8, 0x44, 0xC8, 0xD0, 0x5C,
    0x28, 0xA4, 0xBC, 0x30, 0x8C, 0x00, 0x18, 0x94, 0xEC, 0x60, 0x78, 0xF4, 0x48, 0xC4, 0xDC, 0x50,
    0x2C, 0xA0, 0xB8, 0x34, 0x88, 0x04, 0x1C, 0x90, 0xE8, 0x64, 0x7C, 0xF0, 0x4C, 0xC0, 0xD8, 0x54,
    0x30, 0xBC, 0xA4, 0x28, 0x94, 0x18, 0x00, 0x8C, 0xF4, 0x78, 0x60, 0xEC, 0x50, 0xDC, 0xC4, 0x48,
    0x34, 0xB8, 0xA0, 0x2C, 0x90, 0x1C, 0x04, 0x88, 0xF0, 0x7C, 0x64, 0xE8, 0x54, 0xD8, 0xC0, 0x4C,
    0x38, 0xB4, 0xAC, 0x20, 0x9C, 0x10, 0x08, 0x84, 0xFC, 0x70, 0x68, 0xE4, 0x58, 0xD4, 0xCC, 0x40,
```

```
    0x3C, 0xB0, 0xA8, 0x24, 0x98, 0x14, 0x0C, 0x80, 0xF8, 0x74, 0x6C, 0xE0, 0x5C, 0xD0, 0xC8,
0x44};



// Create a struct_message called myData
struct_message myData;

esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup()
{
  // Init Serial Monitor
  Serial.begin(115200);
  Serial.print(F("DHT22 Test!"));
  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  pinMode(pin, INPUT);
  Wire.begin();
  DHT.begin();
  starttime = millis(); // get the current time;

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK)
```

```
  {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Transmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }
}

void getTemp()
{
  uint32_t start = micros();
  int status = DHT.read();
  uint32_t stop = micros();

  // Print sensor readings and status
  if ((count % 10) == 0)
```

```
{
  count = 0;
  Serial.println();
  Serial.println("Type\tHumidity (%)\tTemp (°C)\tTime (µs)\tStatus");
}
count++;

float humidity = DHT.getHumidity();
float temperature = DHT.getTemperature();

Serial.print("DHT20 \t");
Serial.print(humidity, 1);
Serial.print("\t\t");
Serial.print(temperature, 1);
Serial.print("\t\t");
Serial.print(stop - start);
Serial.print("\t\t");

// Print status based on the value of the 'status' variable
switch (status)
{
// Cases for different status values
}
Serial.print("\n");

// Check temperature and humidity thresholds
float temperatureThreshold1 = 20.0;
float humidityThreshold1 = 40.0;

if (temperature < temperatureThreshold1)
```

```
  {
    Serial.println("Temperature is less than threshold!");
    // Your action for exceeding temperature threshold goes here
  }

  if (humidity < humidityThreshold1)
  {
    Serial.println("Humidity is less than threshold!");
    // Your action for exceeding humidity threshold goes here
  }

  // Check temperature and humidity thresholds
  float temperatureThreshold2 = 22.0;
  float humidityThreshold2 = 60.0;

  if (temperature > temperatureThreshold2)
  {
    Serial.println("Temperature is more than threshold!");
    // Your action for exceeding temperature threshold goes here
  }

  if (humidity > humidityThreshold2)
  {
    Serial.println("Humidity is more than threshold!");
    // Your action for exceeding humidity threshold goes here
  }
}

void loop()
{
```

```
// Set values to send
delay(2500);
myData.id = 1; // Board ID
myData.a = DHT.getTemperature();
myData.b = DHT.getHumidity();
myData.c = 0.0;

// Calculate CRC directly in the loop
uint8_t CRC = 0;
uint8_t *dataPtr = (uint8_t *)&myData;

for (size_t i = 0; i < sizeof(myData); i++)
{
  CRC = crc8_table[CRC ^ dataPtr[i]];
}

// Send data including CRC using ESP-NOW
uint8_t dataToSend[sizeof(myData) + 1];
memcpy(dataToSend, &myData, sizeof(myData));
dataToSend[sizeof(myData)] = CRC;

// Print CRC value
Serial.print("CRC: ");
Serial.println(CRC, HEX); // Print CRC value in hexadecimal

// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress, dataToSend, sizeof(dataToSend));

if (result == ESP_OK)
{
```

```
  Serial.print(F("Humidity: "));
  Serial.print(F("%  Temperature: "));
  Serial.print(F("C "));
  Serial.print(F("Concentration: "));
  Serial.println("  Sent with success");
}
else
{
  Serial.println("Error sending the data");
}

duration = pulseIn(pin, LOW);
lowpulseoccupancy = lowpulseoccupancy + duration;

getTemp();
if ((millis() - starttime) > sampletime_ms)
{
  ratio = lowpulseoccupancy / (sampletime_ms * 10.0);
  concentration = 1.1 * pow(ratio, 3) - 3.8 * pow(ratio, 2) + 520 * ratio + 0.62;
  Serial.print("Concentration: ");
  Serial.println(concentration);

  if (concentration > threshold)
  {
    Serial.println("Alert! Dust concentration exceeds threshold.");
  }

  lowpulseoccupancy = 0;
  starttime = millis();
}
```

```
  delay(2000);
}
```

## RFID and Individual sensor Board Integration - Slave- B Code

```
#include <WiFi.h>
#include <esp_now.h>
#include <SoftwareSerial.h>

SoftwareSerial RFID(1, 2);

String CardNumber = "20000BC313FB"; // Predefined RFID card number

#define TempPin 3  // GPIO pin for temperature sensor
#define LightPin 4 // GPIO pin for light sensor

uint8_t broadcastAddress[] = {0x35, 0x85, 0x18, 0xAC, 0xC7, 0xB4}; // Replace with the MAC address of the receiver ESP32

typedef struct struct_message {
  int id;
  float temperature;
  float light;
  String rfid; // Change from char to String
} struct_message;

struct_message myData;

// CRC-8 lookup table for polynomial 0x8C (reverse of 0x31)
const uint8_t crc8_table[256] = {
  // ... (unchanged)
  0x00, 0x8C, 0x31, 0xBD, 0x45, 0xC9, 0x74, 0xF8, 0x8A, 0x06, 0xBB, 0x37, 0xCF, 0x43, 0xFE, 0x72,
   0x29, 0xA5, 0x18, 0x94, 0x6C, 0xE0, 0x5D, 0xD1, 0xA3, 0x2F, 0x92, 0x1E, 0xE6, 0x6A, 0xD7, 0x5B,
```

```
  0x52, 0xDE, 0x63, 0xEF, 0x17, 0x9B, 0x26, 0xAA, 0xD8, 0x54, 0xE9, 0x65, 0x9D, 0x11, 0xAC, 0x20,
  0x7B, 0xF7, 0x4A, 0xC6, 0x3E, 0xB2, 0x0F, 0x83, 0xF1, 0x7D, 0xC0, 0x4C, 0xB4, 0x38, 0x85, 0x09,
  0xA4, 0x28, 0x95, 0x19, 0xE1, 0x6D, 0xD0, 0x5C, 0x2E, 0xA2, 0x1F, 0x93, 0x6B, 0xE7, 0x5A, 0xD6,
  0x8D, 0x01, 0xBC, 0x30, 0xC8, 0x44, 0xF9, 0x75, 0x07, 0x8B, 0x36, 0xBA, 0x42, 0xCE, 0x73, 0xFF,
  0xF6, 0x7A, 0xC7, 0x4B, 0xB3, 0x3F, 0x82, 0x0E, 0x7C, 0xF0, 0x4D, 0xC1, 0x39, 0xB5, 0x08, 0x84,
  0xDF, 0x53, 0xEE, 0x62, 0x9A, 0x16, 0xAB, 0x27, 0x55, 0xD9, 0x64, 0xE8, 0x10, 0x9C, 0x21, 0xAD,
  0x91, 0x1D, 0xA0, 0x2C, 0xD4, 0x58, 0xE5, 0x69, 0x1B, 0x97, 0x2A, 0xA6, 0x5E, 0xD2, 0x6F, 0xE3,
  0xB8, 0x34, 0x89, 0x05, 0xFD, 0x71, 0xCC, 0x40, 0x32, 0xBE, 0x03, 0x8F, 0x77, 0xFB, 0x46, 0xCA,
  0xC3, 0x4F, 0xF2, 0x7E, 0x86, 0x0A, 0xB7, 0x3B, 0x49, 0xC5, 0x78, 0xF4, 0x0C, 0x80, 0x3D, 0xB1,
  0xEA, 0x66, 0xDB, 0x57, 0xAF, 0x23, 0x9E, 0x12, 0x60, 0xEC, 0x51, 0xDD, 0x25, 0xA9, 0x14, 0x98,
  0x21, 0xAD, 0x10, 0x9C, 0x64, 0xE8, 0x55, 0xD9, 0xAB, 0x27, 0x9A, 0x16, 0xEE, 0x62, 0xDF, 0x53,
  0x08, 0x84, 0x39, 0xB5, 0x4D, 0xC1, 0x7C, 0xF0, 0x82, 0x0E, 0xB3, 0x3F, 0xC7, 0x4B, 0xF6, 0x7A,
  0x73, 0xFF, 0x42, 0xCE, 0x36, 0xBA, 0x07, 0x8B, 0xF9, 0x75, 0xC8, 0x44, 0xBC, 0x30, 0x8D, 0x01,
  0x5A, 0xD6, 0x6B, 0xE7, 0x1F, 0x93, 0x2E, 0xA2, 0xD0, 0x5C, 0xE1, 0x6D, 0x95, 0x19, 0xA4, 0x28
};

// Function to calculate CRC-8
uint8_t calculateCRC8(const void* data, size_t length) {
  uint8_t crc = 0;
  uint8_t* buffer = (uint8_t*)data;
```

```
  for (size_t i = 0; i < length; i++) {
    crc = crc8_table[crc ^ buffer[i]];
  }

  return crc;
}

esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void check(String text) {
  text = text.substring(1, 13); // Extract relevant characters from the scanned data (1 to 12)
  Serial.println("Card ID : " + text);
  Serial.println("Access ID : " + CardNumber);

  String rfid; // Declare rfid as a String

  if (text.equals(CardNumber)) {
    rfid = "Accepted";
  } else {
    rfid = "Denied";
  }

  delay(2000);

  Serial.println("RFID: " + rfid);
```

```
}

void setup() {
  Serial.begin(115200);
  RFID.begin(9600);

  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_send_cb(OnDataSent);

  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  pinMode(TempPin, INPUT);
  pinMode(LightPin, INPUT);
}

void loop() {
  String text = "";
```

```
while (RFID.available() > 0) {
  char data = RFID.read();
  text += data;
}
check(text);

int sensorValue_tm = analogRead(TempPin);
int sensorValue_op = analogRead(LightPin);

float voltage_tm = sensorValue_tm * (3.3 / 4096.0);
float voltage_op = sensorValue_op * (3.3 / 4096.0);

myData.id = 2;
myData.temperature = voltage_tm;
myData.light = voltage_op;
myData.rfid = ""; // Initialize rfid as an empty string

// Calculate CRC
uint8_t CRC = calculateCRC8(&myData, sizeof(myData));

Serial.print(F("  Temperature voltage: "));
Serial.print(voltage_tm);
Serial.print(F("V  Light voltage: "));
Serial.print(voltage_op);
Serial.print(F("V  "));
Serial.println(" Sender 2: Sent with success");
Serial.print("RFID: " + myData.rfid);
Serial.print("  Calculated CRC: ");
Serial.println(CRC);
```

```
  uint8_t dataToSend[sizeof(myData) + 1];
  memcpy(dataToSend, &myData, sizeof(myData));
  dataToSend[sizeof(myData)] = CRC;

  esp_err_t result = esp_now_send(broadcastAddress, dataToSend, sizeof(dataToSend));

  if (result == ESP_OK) {
    Serial.print("  Sender 2: Sent with success");
    Serial.print("  Calculated CRC: ");
    Serial.println(dataToSend[sizeof(myData)]);
  } else {
    Serial.println("Error sending the data");
  }

  delay(3000);
}
//Sender has context menu
```

**ThingSpeak Code**

```
#include <HardwareSerial.h>
#include <WiFi.h>
#include "ThingSpeak.h"

const char* ssid="SENGEEE8092GW"; //Type wifi name
const char* password="pin-9201.double";//Type wifi password

HardwareSerial SerialPort(2); //use UART2
#define UART_TX_PIN 43
#define UART_RX_PIN 44
#define BAUD_RATE 115200
int led = 13;

typedef struct UART_message {
   int id;
   float b;
   float c;
   uint8_t CRC_checksum; //CRC for error checking
} UART_message;

WiFiClient client;

unsigned long myChannelNumber = 1; //The channel number in ThingSpeak (e.g.Channel 1)
const char* myWriteAPIKey = "9SIOLGEEJAB2EYP9"; //Write API Key, can be obtained from ThingSpeak,differenct channel has different API keyTimer variables

unsigned long lastTime = 0;
unsigned long timerDelay = 20000;
```

```
void setup() {
  Serial.begin(BAUD_RATE);
  SerialPort.begin(BAUD_RATE, SERIAL_8N1, UART_RX_PIN, UART_TX_PIN);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  if(WiFi.status() != WL_CONNECTED){
      Serial.print("Attempting to connect Wi-Fi");
      while(WiFi.status() != WL_CONNECTED){
        WiFi.begin(ssid, password);
        delay(5000);
      }
      Serial.println("\nWiFi Connected.");
  }
  Serial.print("RSSI: ");
  Serial.println(WiFi.RSSI());
  pinMode(led,OUTPUT);
  ThingSpeak.begin(client);  // Initialize ThingSpeak
  delay(1000);
}

void loop() {

// Variable definition for uploading data to ThingSpeak
    float humidity;
    float temperature;
    float TempVoltage;
    float LightVoltage;
    float LightDensity;
```

```
// Serial Port Scanning
  if (SerialPort.available()) {
    // String message = SerialPort.readString();
    // Serial.println("Received message: " + message);
    size_t byteSize = sizeof(UART_message);
    byte* byteArray = new byte[byteSize];
    SerialPort.read(byteArray, byteSize);

    // Deserialize the byte array back into the structure
    UART_message UARTReceivedData;
    memcpy(&UARTReceivedData, byteArray, byteSize);

/*--------Process different types of data----------*/
    switch(UARTReceivedData.id){
      case 1:
            // Print the received structure data and check the data valid or not
            if (UARTReceivedData.b > 0 && (UARTReceivedData.c > -50 && UARTReceivedData.c < 50)){
              Serial.print("Received from Board ID: ");
              Serial.println(UARTReceivedData.id);
              Serial.print("    Humidity: ");
              Serial.print(UARTReceivedData.b);   // b is linked to humidity
              Serial.println(" % ");
              Serial.print("    Temperature: ");    // c to temperature
              Serial.print(UARTReceivedData.c);
              Serial.println(" C ");
              Serial.print("    CRC = ");  // CRC=1 means data transmission right
              Serial.println(UARTReceivedData.CRC_checksum);
              humidity=UARTReceivedData.b;
              temperature=UARTReceivedData.c;
```

```
        ThingSpeak.setField(1, humidity);
        ThingSpeak.setField(2, temperature);
      }
      break;
  case 2:
      if (UARTReceivedData.b > 0 && UARTReceivedData.c > 0 ){
        Serial.print("Received from Board ID: ");
        Serial.println(UARTReceivedData.id);
        Serial.print("    Temperature voltage: ");
        Serial.print(UARTReceivedData.b);   // b is linked to humidity
        Serial.println(" V ");
        Serial.print("    Light voltage: ");    // c to temperature
        Serial.print(UARTReceivedData.c);
        Serial.println(" V ");
        Serial.print("    CRC = ");  // CRC=1 means data transmission right
        Serial.println(UARTReceivedData.CRC_checksum);
        TempVoltage=UARTReceivedData.b;
        LightVoltage=UARTReceivedData.c;
        ThingSpeak.setField(1, TempVoltage);
        ThingSpeak.setField(2, LightVoltage);
      }
      break;
 default:
      if (UARTReceivedData.b > 0 ){
        Serial.print("Received from Board ID: ");
        Serial.println(UARTReceivedData.id);
        Serial.print("    TSL2591 Light Density: ");
        Serial.print(UARTReceivedData.b);   // b is linked to light desnity
        Serial.println(" lux ");
        Serial.print("    CRC = ");  // CRC=1 means data transmission right
```

```
            Serial.println(UARTReceivedData.CRC_checksum);
            LightDensity=UARTReceivedData.b;
            ThingSpeak.setField(5, LightDensity);
          }
          break;
  }
  digitalWrite(led, HIGH);
  delay(100);

//ThingSpeak Data UIploading
if ((millis() - lastTime) > timerDelay) {

  // Connect or reconnect to WiFi

  if(WiFi.status() != WL_CONNECTED){
    Serial.print("Attempting to connect");
    while(WiFi.status() != WL_CONNECTED){
      WiFi.begin(ssid, password);
      delay(5000);
    }
    Serial.println("\nConnected.");
  }
  // Get a new temperature reading
    // ThingSpeak.setField(1, humidity);
    // ThingSpeak.setField(2, temperature);
    // ThingSpeak.setField(3, TempVoltage);
    // ThingSpeak.setField(4, LightVoltage);
    // ThingSpeak.setField(5, LightDensity);
```

```
    // Write to ThingSpeak. There are up to 8 fields in a channel, allowing you to store up to 8 different
    // pieces of information in a channel.  Here, we write to field 1.
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);


    if(x == 200){
      Serial.println("Channel update successful.");
    }
    else{
      Serial.println("Problem updating channel. HTTP error code " + String(x));
    }
    lastTime = millis();
  }
  }
  digitalWrite(led, LOW);//digitalWrite(LED, HIGH);


}
```