Microcontroller Programming Documentation

Plant Monitoring System - test.ino

Overview:

The `test.ino` file contains the firmware code for a Plant Monitoring System based on an ESP32 microcontroller. This system is designed to monitor various environmental parameters of plants, including soil moisture, light intensity, temperature, humidity, and volatile organic compounds (VOCs). The data collected by the system is published to an AWS (Amazon Web Services) cloud server for storage and analysis. Additionally, the system provides a web-based interface for user interaction and configuration.

Libraries and Dependencies:

- `Wire.h`: Arduino library for I2C communication.
- `Adafruit_SGP40.h`: Library for interfacing with the Adafruit SGP40 VOC sensor.
- `WiFiManager.h`: Library for easy configuration of WiFi networks on ESP32 devices.
- `AsyncTCP.h` and `ESPAsyncWebServer.h`: Libraries for asynchronous TCP communication and web server implementation on ESP32.
- `BH1750.h`: Library for the BH1750 light sensor.
- `Adafruit_SHT31.h`: Library for the Adafruit SHT31 temperature and humidity sensor.
- `NTPClient.h` and `WiFiUdp.h`: Libraries for NTP (Network Time Protocol) client implementation and WiFi UDP communication.
- `utils.h`: Header file containing functions for AWS IoT communication.
- `Arduino.h`: Arduino core library.

Pin Definitions:

- `SOIL_MOISTURE_PIN`: Pin connected to the soil moisture sensor.
- `LIGHT_SENSOR_SDA` and `LIGHT_SENSOR_SCL`: Pins for I2C communication with the I2C sensors.

Device Information:

- `__typename`: Type name of the data.
- `username`: Username associated with the data.
- `description`: Description of the data.
- `name`: Name associated with the data.
- `updatedAt`: Timestamp indicating when the data was last updated.
- `smell`: Initial value set for VOC measurement.

Setup Function (`void setup()`):

- Initializes serial communication for debugging purposes.
- Initializes random seed for generating random IDs.
- Initializes I2C communication.
- Initializes sensors (light sensor and temperature/humidity sensor).

- Configures WiFi connection using WiFiManager library.
- Initializes NTP Client for time synchronization.
- Connects to AWS IoT Core using `connectAWS()` function.
- Sets up web server endpoints for handling HTTP requests.
- Initializes VOC sensor and checks for availability.
- Prints a message indicating successful WiFi connection.

Loop Function (`void loop()`):

- Generates a random ID for each data reading.
- Retrieves current date and time using NTP Client.
- Reads sensor data including soil moisture, light intensity, temperature, humidity, and VOC levels.
- Prints sensor data and metadata to the serial monitor for debugging.
- Publishes sensor data along with metadata to AWS IoT Core using `publishMessageA()` function.
- Handles incoming MQTT messages from AWS IoT Core.
- Delays execution for stability.

Integration Steps:

1. Set up the hardware components (ESP32 board, sensors).
2. Configure AWS IoT Core and obtain the necessary credentials.
3. Include the required libraries and dependencies.
4. Upload the `test.ino` firmware code to the ESP32 board.
5. Monitor the serial output for debugging information.
6. Access the web-based interface hosted by the device for data visualization and configuration.

Conclusion:

The Plant Monitoring System provides a comprehensive solution for monitoring and managing plant environments. By leveraging IoT technologies and cloud integration, the system offers real-time insights into environmental conditions, facilitating informed decision-making for plant care and optimization. The provided firmware code enables seamless integration with AWS IoT Core, allowing for secure and efficient data transmission and management.

AWS IoT Communication Module Documentation

Overview:

The AWS IoT Communication Module facilitates communication between an ESP32-based device and the AWS IoT Core service. It allows the device to publish sensor data to AWS and subscribe to topics for receiving commands or updates from the cloud. This module is essential for integrating IoT devices into AWS infrastructure, enabling seamless data transmission and management.

Header File: utils.h

Functions:

1. void messageHandler(char* topic, byte* payload, unsigned int length):
    - Description: Handles incoming MQTT messages from the AWS IoT Core.
    - Parameters:
        - char* topic: Topic of the received message.
        - byte* payload: Payload of the received message.
        - unsigned int length: Length of the payload.
    - Usage: This function is called automatically whenever a message is received from AWS IoT Core. It parses the message payload, processes the data, and performs necessary actions based on the received information.
2. void connectAWS():
    - Description: Establishes a secure connection to AWS IoT Core.
    - Usage: Call this function to connect the ESP32 device to the AWS IoT service. It handles the establishment of a secure Wi-Fi connection and sets up the necessary certificates for secure communication. Once connected, the device can publish data and subscribe to topics on AWS IoT Core.
3. void publishMessageA(String id, char __typename[], char createdAt[], char username[], char description[], float humidity, float light, float moisture, float smell, float temperature, char name[]):
    - Description: Publishes a message containing sensor data to AWS IoT Core.
    - Parameters:
        - String id: Unique identifier for the message.
        - char __typename[]: Type name of the data.
        - char createdAt[]: Timestamp of data creation.
        - char username[]: User associated with the data.
        - char description[]: Description of the data.
        - float humidity: Humidity sensor reading.
        - float light: Light sensor reading.
        - float moisture: Soil moisture sensor reading.
        - float smell: VOC sensor reading.
        - float temperature: Temperature sensor reading.
        - char name[]: Name associated with the data.
    - Usage: Call this function to publish sensor data to AWS IoT Core. It constructs a JSON message containing the provided data and publishes it to the specified MQTT topic on AWS. The message can then be processed or stored by AWS services for further analysis or action.

Integration Steps:

1. Include Header File:
    - Include the utils.h header file in your ESP32 project.
2. Configure AWS IoT Core:

- Set up an AWS IoT Core instance and obtain the necessary credentials and endpoint information.
- Update the aws_credentials.h file with your AWS IoT endpoint, device certificates, and thing name.

3. Initialize WiFi:
    - Ensure that the ESP32 device is connected to a Wi-Fi network.
4. Connect to AWS IoT Core:
    - Call the connectAWS() function to establish a connection to AWS IoT Core. This should be done once during the setup phase of your program.
5. Publish Sensor Data:
    - Use the publishMessageA() function to publish sensor data to AWS IoT Core whenever new readings are available. Provide the relevant sensor data as parameters to the function.
6. Handle Incoming Messages:
    - Implement message handling logic in the messageHandler() function to process incoming messages from AWS IoT Core, if required.

Conclusion:

The AWS IoT Communication Module simplifies the integration of ESP32-based devices with AWS IoT Core, enabling seamless communication and data exchange between IoT devices and cloud services. By leveraging this module, developers can build robust IoT applications that harness the power of AWS for data processing, storage, and analysis.

AWS Credentials Header File Documentation

Overview:

The aws_credentials.h header file contains the necessary credentials and configurations for establishing a secure connection between an ESP32 device and the AWS IoT Core service. These credentials include WiFi network information and AWS IoT endpoint details, along with the device's Thing name and associated certificates for secure communication.

Header File: aws_credentials.h

Definitions and Configurations:

1. THINGNAME:
    - Description: The unique identifier (Thing name) assigned to the ESP32 device in the AWS IoT Core registry.
    - Usage: Replace "esp32" with the desired Thing name for the device.
2. WiFi Credentials:
    - WIFI_SSID: SSID of the WiFi network to which the ESP32 device will connect.
    - WIFI_PASSWORD: Password for the WiFi network.
    - Usage: Replace "ssid" and "password" with the respective WiFi SSID and password.
3. AWS IoT Endpoint:

- AWS_IOT_ENDPOINT:
    - Description: The endpoint URL of the AWS IoT Core service.
    - Usage: Replace "example@amazonaws.com" with the actual endpoint URL provided by AWS.

4. Certificates:
    - AWS_CERT_PRIVATE:
        - Description: Private key certificate for the device stored in program memory (PROGMEM).
        - Usage: Replace the placeholder text with the actual private key certificate obtained from AWS.
    - AWS_CERT_CRT:
        - Description: Certificate for the device stored in program memory (PROGMEM).
        - Usage: Replace the placeholder text with the actual certificate obtained from AWS.
    - AWS_CERT_CA:
        - Description: Root CA certificate for AWS IoT Core stored in program memory (PROGMEM).
        - Usage: Replace the placeholder text with the actual Root CA certificate obtained from AWS.

Integration Steps:

1. AWS IoT Core Setup:
    - Set up an AWS IoT Core instance and create a Thing for the ESP32 device.
    - Obtain the Thing name and endpoint URL from AWS IoT Core.
2. WiFi Configuration:
    - Ensure that the ESP32 device has access to a WiFi network.
    - Update the WIFI_SSID and WIFI_PASSWORD constants with the appropriate WiFi credentials.
        i. This is for debugging purposes to override the WiFiManager protocol in the main file.
3. AWS IoT Endpoint Configuration:
    - Update the AWS_IOT_ENDPOINT constant with the endpoint URL provided by AWS IoT Core.
4. Certificates Configuration:
    - Obtain the necessary certificates (private key, device certificate, Root CA certificate) from AWS IoT Core.
    - Update the corresponding constants (AWS_CERT_PRIVATE, AWS_CERT_CRT, AWS_CERT_CA) with the actual certificate contents.
5. Include Header File:
    - Include the aws_credentials.h header file in your ESP32 project.

Conclusion:

The aws_credentials.h header file serves as a centralized location for storing AWS IoT Core credentials and configurations required for secure communication between ESP32 devices and AWS services. By following the integration steps and updating the necessary parameters, developers can seamlessly connect their devices to AWS IoT Core and leverage cloud-based functionalities for their IoT applications.