Orleans University

Polytech Engineering school

IoT-Based Smart Greenhouse Monitoring System

Prepared By:

Aya CHATIT

Jose Guillermo Cruz Garcia

Joseph Mendez

Noah Chau

Pratik Khadka Saju Khakurel

Vincent Mwenda Mworia

Instructor:

Remy Beaudenon

Submission Date:

Wednesday 28th January 2026

Academic Year 2025/2026

Table of Contents

1. Introduction

1.1 Background and Motivation

Modern agriculture is currently undergoing a digital transformation, often referred to as **Smart Farming**. The core of this movement is the use of IoT (Internet of Things) to create "intelligent" environments where plants can thrive with minimal human error.

The Motivation for this project stems from three key factors:

- **Precision and Stability:** Traditional greenhouses require constant physical presence. By automating temperature and humidity, we remove the "guesswork" and ensure crops stay in their biological "sweet spot" 24/7.
- **The "Distance" Challenge:** In many rural settings, the greenhouse isn't right next to the office. The need to bridge a **250-meter gap** without expensive cellular plans or complex wiring provides a practical engineering challenge: creating a long-range, low-power "bridge" using existing ADSL.
- **Operational Efficiency:** Manual monitoring is time-consuming and inefficient. This project is motivated by the desire to give the operator **total visibility and control** from a single screen, allowing them to focus on crop health rather than adjusting valves and switches by hand.

1.2 Problem Statement

Traditional greenhouse management relies heavily on manual intervention, which is prone to human error and delayed responses to environmental shifts.

With a distance of 250 meters and a lack of cellular coverage (3G/4G), there is a critical need for a reliable, long-range communication link. Without a bi-directional remote monitoring and control system, the operator cannot guarantee that crops remain within "ideal" biological thresholds, leading to potential crop loss or resource waste (water/electricity).

1.3 Project Objectives

The main goal is to design an automated, low-power IoT ecosystem that bridges the gap between the field and the control room. Specific objectives include:

- **Real-Time Data Acquisition:** Deploy sensors to continuously monitor temperature, humidity, and light levels.
- **Dual-Mode Control (Automatic/Manual):** * Implement an **Automatic Mode** where actuators (fans, heaters, pumps) react to sensor data to maintain optimal conditions.
    - Implement a **Manual Override** via a Graphical User Interface (GUI) for direct operator intervention.
- **Bi-Directional Communication:** Establish a stable wireless link capable of covering 250 meters to transmit data to the GUI and receive commands back from the control room.
- **Local Feedback:** Integrate an optional LCD on-site for immediate status checks by technicians in the field.
- **Energy Efficiency:** Optimize the system architecture to ensure minimal power consumption for the remote sensor nodes.

1.4 Scope and Limitations

This project focuses on the integration of hardware and software to create a functional prototype. However, the following boundaries apply:

- **Connectivity:** The system is limited to the local ADSL availability in the control room. It will not utilize cellular data.
- **Range:** The wireless communication is specifically optimized for the **250-meter radius** required by the site layout.
- **Complexity:** To ensure maintainability, the system will avoid overly "black-box" AI models, focusing instead on a **user-friendly GUI** and transparent logic.
- **Hardware:** The project assumes a fixed set of sensors (Temperature, Humidity, Light) and actuators; adding specialized chemical or soil-composition sensors is outside the current scope.

2. System Requirements

2.1 Functional Requirements

The primary objective of the project is to design a connected IoT system capable of measuring ambient light and regulating a resistive heater in order to simulate solar irradiation inside a laboratory greenhouse prototype.

The system shall implement the following core functionalities:

1. Solar Irradiation Simulation
    - Measure ambient light intensity (lux).

    - Regulate the resistive heater power proportionally to the measured light level.
    - Ensure that the relationship between solar irradiance and electrical energy supplied to the greenhouse is as proportional as possible.

2. Environmental Regulation
    - Measure air temperature for thermal control and frost protection.
    - Measure relative humidity to regulate ventilation.
    - Activate the 12 V / 0.45 W fan to reduce humidity or excessive temperature.

3. Safety Mechanisms
    - Enforce frost protection by forcing heater activation below a minimum threshold.
    - Activate ventilation if the temperature exceeds a defined upper limit.
    - Provide fault detection and visual alarm signaling.
    - Allow local fault acknowledgment via push-button input.

4. Local Human–Machine Interface
    - Display system status and fault information on local OLED screen.
    - Show current environmental variables and actuator states.

5. Remote Supervision
    - Enable real-time monitoring and control via dedicated supervision software.
    - Provide a visualization of historical trends.
    - Allow access to weather forecast data (7-day forecast).

2.2 Non-Functional Requirements

The system must satisfy the following performance and quality constraints:

1. Proportionality Accuracy
    - The electrical heating power must scale consistently with measured ambient light.
    - The system must minimize the deviation between the irradiance input and thermal output response.

2. Continuous Operation

    - 24/7 operation capability.
    - Designed for long-term experimental use.

3. Measurement Precision

    - Temperature measurement accuracy within ±0.3 °C.

- Humidity measurement accuracy within ±2 %RH.
- Light intensity resolution of 1 lux.

4. Low-Latency Supervision
    - Near real-time monitoring through MQTT communication.

5. Energy Efficiency
    - Minimize power consumption due to 12 V solar-powered architecture.
    - Optimize telemetry publishing intervals (15–30 seconds typical).
    - Reduce unnecessary communication overhead.

2.3 Operational Constraints

The greenhouse prototype and infrastructure impose the following constraints:

1. Physical Setup
    - Greenhouse equipped with:
        - 12 V, 0.45 W ventilation fan
        - 10 Ω, 25 W resistive heating element (external power supply required)

2. Communication Distance
    - Approximately 250 meters between greenhouse and supervision room.
    - No 3G or 4G cellular connectivity available.

3. Energy Architecture
    - System powered by 12 V solar supply.
    - Planned battery: 12 V, 8 Ah lead-acid (96 Wh).
    - Solar panel: 25 W.
    - Energy optimization required to preserve battery autonomy.

4. Laboratory Environment
    - System designed for experimental validation.
    - Controlled environment simulation rather than large-scale agricultural deployment.

2.4 Communication and Infrastructure Constraints

1. Local Network Dependency

    - Only wired ADSL available at supervision station.
    - Communication must operate through local Wi-Fi bridge architecture.

2. Low Data Throughput Requirement
    - Environmental variables evolve slowly; high-frequency telemetry is not required.
    - Average data rate < 1 kbps.

3. Bi-Directional Communication Requirement
    - Telemetry publishing from greenhouse node.
    - Remote configuration and control commands from supervision system.

4. Scalability Consideration
    - Architecture should allow future extension to additional greenhouse nodes.

3. Project Methodology & Team Organization

3.1 Development Approach

During a two-week intensive implementation phase, the project was developed using an organized, iterative methodology. Technical research, hardware selection, requirement analysis, and incremental system integration were all incorporated into the methodology. Functional requirements and communication constraints were clearly defined before the development process started. Before deciding on the final architecture, the team assessed a number of communication technologies in light of these limitations (250 m range, no cellular coverage, ADSL only at the supervision room). Parallel studies were carried out on the following topics:

- Actuator driving circuitry and MOSFET modules;
- Power system viability and energy autonomy;
- Microcontroller platform selection;
- Communication technologies (Wi-Fi, LoRa, bridge solutions);
- Actuator driving circuitry and MOSFET modules;
- Availability, affordability, and dependability of hardware components;

The implementation phase followed a modular approach:

1. Hardware validation (sensors + actuators independently tested)
2. Embedded firmware architecture (FreeRTOS task separation)
3. Communication layer (MQTT + Wi-Fi bridge)
4. Integration with Raspberry Pi, Node-RED, and InfluxDB

5. System-level validation

Continuous feedback from the Project Owner ensured alignment with requested functionalities and constraints throughout the development process.

3.2 Task Distribution

The project was carried out by a team of seven members, with responsibilities distributed according to technical strengths while maintaining collaborative overlap during integration.

The main task categories were:

- Embedded firmware development (FreeRTOS task architecture, control logic)
- Communication system implementation (MQTT architecture, Wi-Fi bridge setup)
- Hardware design (MOSFET driver circuits, actuator interfacing)
- Power system analysis (battery sizing, solar estimation, generator evaluation)
- Sensor research and validation (SHT31, BH1750 performance analysis)
- Data pipeline configuration (Raspberry Pi, Node-RED, InfluxDB, Grafana)
  Documentation and reporting

Although roles were defined, system integration was performed collaboratively to ensure consistency across hardware, firmware, and cloud layers.

The Project Owner reviewed progress regularly, validating technical decisions and ensuring compliance with specified requirements.

3.3 Project Management Tools

Task coordination and sprint tracking were managed using Taiga, an agile project management platform.

The workflow included:

- Creation of user stories based on system requirements
- Task breakdown into implementation units
- Assignment of responsibilities among team members
- Status tracking (To Do / In Progress / Review / Done)
- Continuous validation by the Project Owner

Taiga enabled:

- Clear visibility of task progress

- Structured prioritization
- Iterative improvements
- Rapid feedback cycles

This structured organization allowed the team to complete the prototype implementation within the two-week development window while maintaining technical rigor and traceability of decisions.

4. Technology & Component Selection

4.1 Network Selection: Wi-Fi vs LoRa

- Ultra-low power consumption is not necessary
- Lora adds complexity compared to Wi-Fi
- An inexpensive Wi-Fi extender can easily reach 250m.
- Wi-Fi has a higher data rate and simplifies bidirectional communication.
- Wi-Fi rather than LoRa due to simplicity, range, and data transmission

4.2 Microcontroller Selection (ESP32 – Heltec V3)

- Criteria: cost, power consumption, processing, and availability,
- Low power consumption (~10µA).
- Suitable for control signals I2C/SPI/UART (ideal for PWM/IO for heating/fan control).
- Accomplishes Optional criteria: Built-in 0.96" OLED

4.3 Sensor Selection (SHT31, BH1750)

BH1750

- Low power consumption
- I²C communication
- Good precision

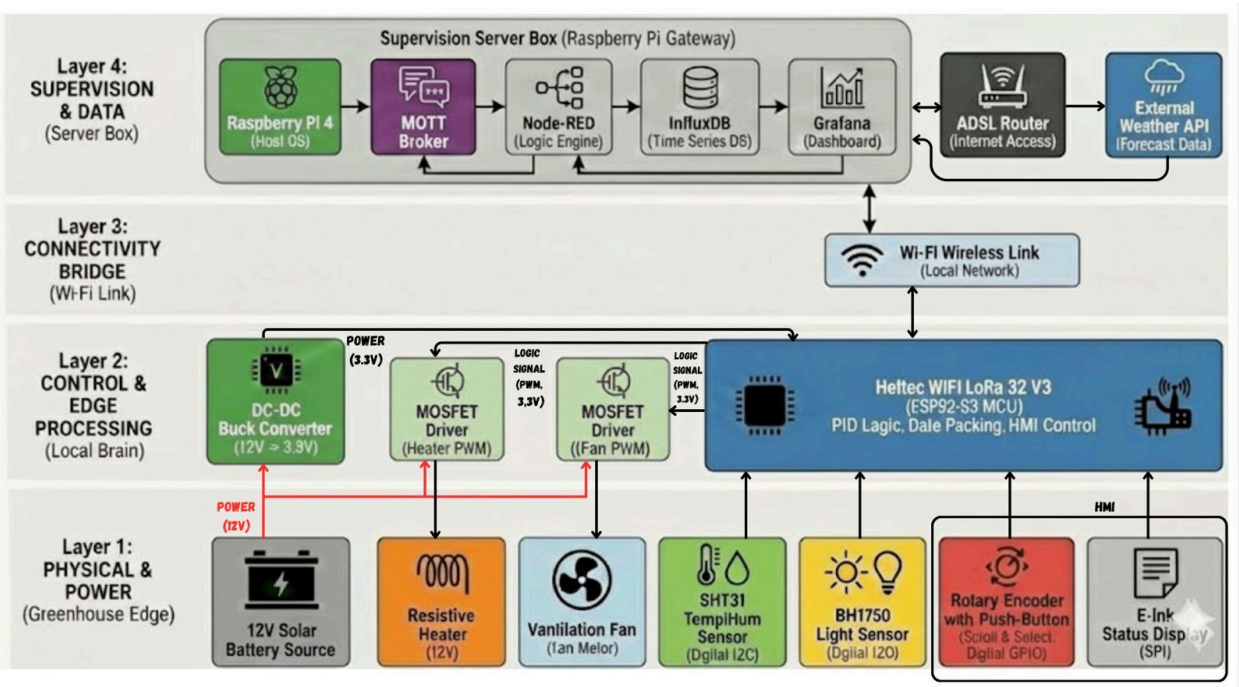
SHT31

- Balances accuracy and low power
- Straightforward integration.
- Simple I²C interface, stable and fast
- Compared to cheaper options, it reduces calibration effort

5. Global System Architecture

The IoT greenhouse system is designed using a four-layer hierarchical architecture to ensure modularity between physical sensing, local processing, and remote supervision. This structure facilitates the integration of various environmental sensors and actuators while maintaining a clear separation of concerns:

- **Layer 1: Physical & Power (Greenhouse Edge):** This foundational layer contains the physical hardware and power sources. It includes the **12V Solar Battery Source**, which powers the system, and environmental peripherals such as the **SHT31 Temp/Hum Sensor**, **BH1750 Light Sensor**, and actuators like the **Ventilation Fan** and **Resistive Heater**.
- **Layer 2: Control & Edge Processing (Local Brain):** Acting as the central intelligence, this layer is anchored by the **Heltec WiFi LoRa 32 V3 (ESP32-S3 MCU)**. It manages **PID Logic** for climate control and interfaces with specialized components like **MOSFET Drivers** for PWM control of the heater and fan. It also handles the local **HMI (Human-Machine Interface)** via a rotary encoder and E-Ink display.
- **Layer 3: Connectivity Bridge (Wi-Fi Link):** This layer provides the necessary communication infrastructure, utilizing a **Wi-Fi Wireless Link** to bridge the gap between local edge processing and the broader network.
- **Layer 4: Supervision & Data (Server Box):** The highest level of the architecture is a **Raspberry Pi Gateway**. This server box hosts the backend services, including an **MQTT Broker** for message handling, **Node-RED** as the logic engine, **InfluxDB** for time-series data storage, and **Grafana** for the user dashboard.



6. Hardware Design

6.1 Circuit Design Overview

The hardware architecture is centered around the Heltec WiFi LoRa 32 V3 (ESP32-S3) microcontroller board, which serves as the control and communication core of the greenhouse system.

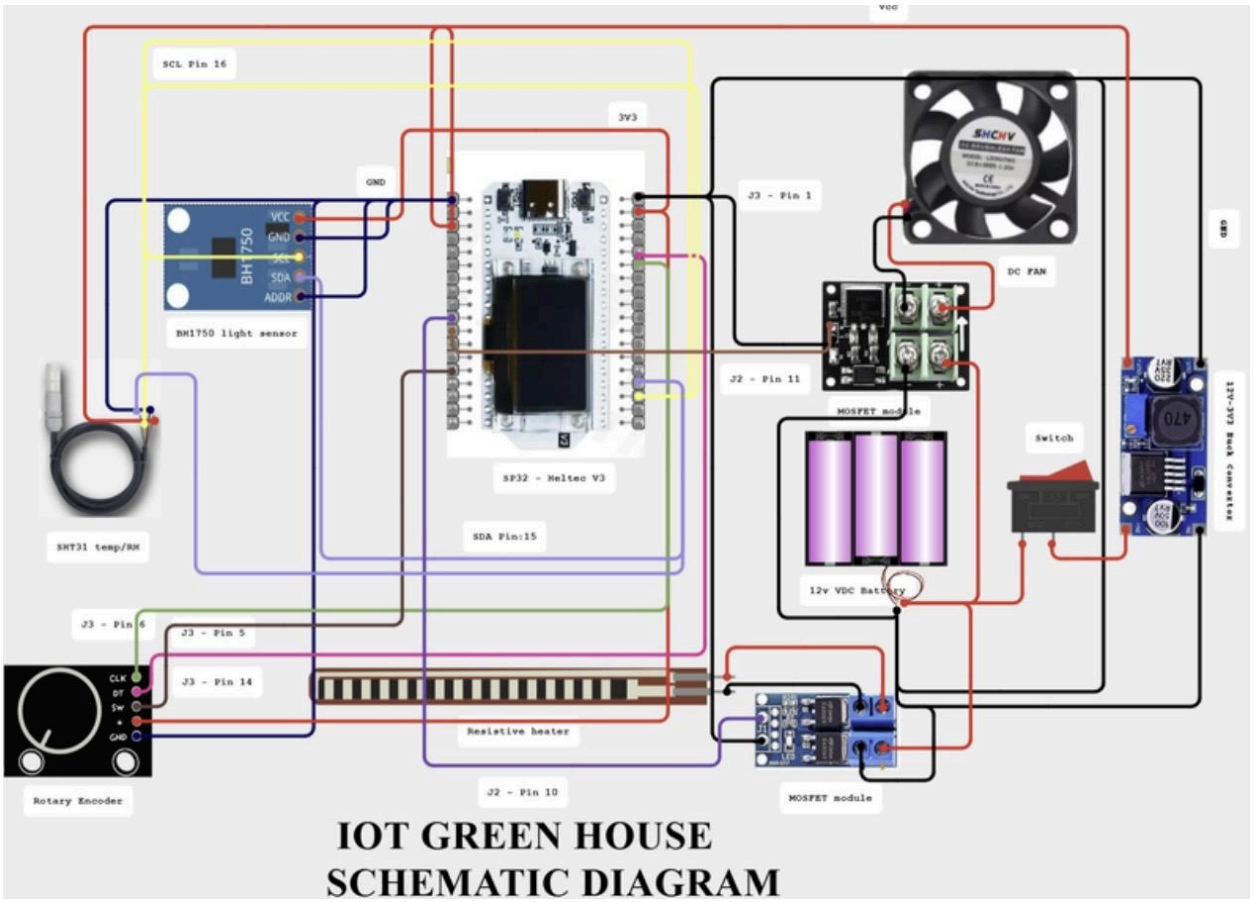
The circuit integrates four main subsystems:

1. Sensor subsystem (I²C bus)
2. User interface subsystem (rotary encoder + OLED)
3. Actuator driving subsystem (MOSFET switching stage)
4. Power subsystem (12 V battery + solar panel + DC-DC conversion)

The Heltec board operates at 3.3 V logic level and interfaces with environmental sensors via a dedicated I²C bus. Actuators (heater and fan) are powered from the 12 V rail and switched using logic-level N-channel MOSFET modules controlled by GPIO outputs.

The design follows a low-side switching topology for both heater and fan. The heater is controlled via PWM to enable proportional power regulation, while the fan operates in binary (ON/OFF) mode.

All high-current paths are physically separated from low-voltage logic paths to improve electrical safety and reduce electromagnetic interference.



6.2 Sensor Integration

Two primary environmental sensors are integrated on the I²C bus:

6.2.1 SHT31 - Temperature & Humidity Sensor

- Interface: I²C
- Operating voltage: 3.3 V
- Temperature range: −40 °C to +125 °C
- Humidity range: 0–100 %RH
- Accuracy: ±0.3 °C / ±2 %RH

The SHT31 is powered from the 3.3 V rail and connected to the dedicated sensor I²C bus (SDA/SCL). The built-in heater of the SHT31 is disabled at startup to avoid interference with environmental measurements.

6.2.2 BH1750 - Ambient Light Sensor

- Interface: I²C

- Operating voltage: 3.3 V
- Resolution: 1 lux
- Measurement range: 1–65,535 lux

The BH1750 provides calibrated lux output, which is used as the primary input variable for proportional heater control to simulate solar irradiation.

Both sensors share the same I²C bus operating at 100 kHz. The firmware polls sensors at a default interval of 2 seconds. Invalid readings trigger automatic recovery logic instead of propagating erroneous data to the control loop.

The integrated OLED display on the Heltec board uses a separate I²C bus, preventing contention between display refresh and environmental sensing.

6.3 Actuator Driving Circuit

The greenhouse includes two actuators:

- 10 Ω, 25 W resistive heating element
- 12 V, 0.45 W DC fan

Both actuators are powered from the 12 V rail and driven through logic-level N-channel MOSFET modules configured as low-side switches.

Heater Control

The resistive heater (10 Ω) draws approximately:

$I = 12V/10\Omega = 1.2A$

Maximum power:

$P = 12V \times 1.2A \approx 14.4W$

The heater is controlled using PWM from a dedicated GPIO pin. This enables proportional control of electrical power injection, fulfilling the requirement that heating power must scale with ambient light intensity.

A logic-level n-type MOSFET is required to ensure full conduction at 3.3 V gate drive. The selected MOSFET modules provide:

- Low RDS(on)
- Gate isolation
- Terminal block connection

- Integrated flyback protection (for inductive loads)

Since the heater is purely resistive, no flyback diode is required for this branch.

Fan Control

The DC fan (0.45 W) operates in ON/OFF mode.

Because the fan is an inductive load, a flyback diode is included (or integrated within the MOSFET module) to protect the MOSFET from voltage transients during switching.

The fan is activated for:

- Over-temperature protection
- Humidity reduction

All actuator outputs are forced OFF at system startup to ensure safe boot behavior.

6.4 Power Architecture

The system is powered by a 12 V solar-based supply consisting of:

- 12 V, 8 Ah lead-acid battery (96 Wh total energy)
- 25 W solar panel
- Solar charge controller

The 12 V rail supplies:

- Heater (directly through MOSFET switching)
- Fan
- DC-DC buck converter

A DC-DC step-down converter reduces 12 V to 5 V, which powers the Heltec board. The board's internal regulator then generates 3.3 V for logic and sensors.

Key design considerations:

- Maximum continuous heater power ≈ 14–25 W (depending on supply)
- Total system consumption managed to preserve battery autonomy

- Telemetry update interval reduced to 15–30 s during normal operation to reduce energy overhead
- Clear separation between high-current actuator paths and low-voltage control electronics

Given the energy constraints, the control logic implements strategies to minimize unnecessary heater activation and reduce communication frequency when possible, all based on the mix intensity that must be simulated in the greenhouse.

7. Embedded Software Architecture

7.1 Software Design Approach

The embedded firmware is designed as a modular, task-based system using FreeRTOS on the ESP32 (Heltec WiFi LoRa 32 V3). Instead of a monolithic loop(), the application is decomposed into independent tasks—each responsible for one subsystem (encoder input, sensors acquisition, display/UI, control logic, and MQTT communication).

To avoid unsafe global variable sharing, the system uses a queue-based communication architecture. Each producer task publishes its latest state into a dedicated snapshot queue, and consumer tasks read the most recent value when needed. This improves modularity, reduces coupling, and prevents race conditions.

At startup, the system performs:

1. GPIO initialization
2. Queue creation
3. Module initialization
4. Task creation

The main Arduino loop() remains idle and only delays.

7.2 FreeRTOS Task Architecture

The embedded firmware is structured using **FreeRTOS**, a lightweight real-time operating system widely adopted in embedded systems for deterministic and modular task management. Unlike a traditional single-loop architecture, FreeRTOS enables the system to run multiple independent tasks concurrently, each with its own stack, priority, and execution period. This approach was selected to ensure predictable timing behavior, clean separation of concerns, and safe scalability as system complexity increases. In the greenhouse application, sensor acquisition, user interface handling, control logic computation, and MQTT communication operate simultaneously but independently. By leveraging FreeRTOS scheduling and priority-based preemption, the system maintains responsiveness for user interactions, reliable periodic sensor sampling, and stable control execution without blocking operations. This architecture significantly improves maintainability, testability, and robustness compared to a monolithic firmware design.

7.2.1 Task List and Responsibilities

1. Encoder Task
    a. Reads the rotary encoder rotation and button press events.
    b. Produces encoder events: ENC_UP, ENC_DOWN, ENC_SELECT, ENC_SELECT_LONG.
    c. Publishes events to the **Encoder Queue**.
2. Sensor Task
    a. Reads environmental sensors via a dedicated I2C bus (controller #1).
    b. Sensor devices: **SHT31 (temperature/humidity)** and **BH1750 (lux)**.
    c. Performs bus recovery, scanning, and sensor health checks.
    d. Publishes the latest sensor snapshot to the **Sensors Data Model Queue**.
3. Display Task
    a. Implements the local UI on the OLED.
    b. Reads the latest sensor values for visualization.
    c. Reads encoder events to navigate menu items and trigger actions.
    d. Generates local actuator commands (manual mode) and writes them as a mode/context snapshot to the **Mode Context Queue**.
4. Logic Task
    a. Acts as the main “controller” (PLC-like logic).
    b. Reads: sensor model snapshot + UI mode snapshot + optional server configuration snapshot.
    c. Computes the next actuator outputs (fan + heater PWM).
    d. Applies outputs on GPIO.
    e. Publishes the resulting system state as the latest mode/context snapshot.
5. MQTT Task
    a. Handles Wi-Fi connection and MQTT connectivity.
    b. Publishes sensor + actuator status to the supervision system (rate-limited).
    c. Subscribes to server commands and converts received setpo in ints into a configuration snapshot written to the **Config Queue**.

7.2.2 Task Periodicity, Priority, and Stack

- **Encoder Task**: ~5 ms loop delay (fast UI responsiveness), priority **1**, stack **3072 bytes**
- **Display Task**: ~100 ms loop delay, priority **1**, stack **4096 bytes**
- **Sensors Task**: default **2000 ms** sampling interval, priority **2**, stack **4096 bytes**
- **Logic Task**: ~200 ms loop delay, priority **3**, stack **4096 bytes**
- **MQTT Task**: ~50 ms loop delay, priority **2**, stack **6144 bytes**, telemetry rate limit **≥ 1000 ms**

  This priority assignment ensures the control loop remains stable while communication and UI remain responsive.

7.3 Inter-Task Communication

7.3.1 Snapshot Queue Design (Overwrite + Peek)

All core queues are implemented as **length = 1 snapshot queues**, using:

- **Producer** → xQueueOverwrite() (always keep latest data)
- **Consumer** → xQueuePeek() (read latest data without removing it)

This approach is chosen because:

- only the most recent sensor snapshot/mode state/config is relevant,
- It avoids backlog and queue overflow,
- It keeps the system deterministic and lightweight.

7.3.2 Queues Used in the System

Queue 1: Encoder Queue (qEncoder)

- Producer: Encoder Task
- Consumer: Display Task
- Payload: EncodeEvent_t
- Pattern: Overwrite by producer, peek by consumer

Queue 2: Sensors Data Model Queue (qSensorsDataModel)

- Producer: Sensors Task
- Consumers: Display Task, MQTT Task
- Payload: SensorData_t (temperature, humidity, lux, timestamp)

Queue 3: Config Queue (qConfig)

- Producer: MQTT Task (server commands)
- Consumers: Logic Task (and can be extended for display/sensors)
- Payload: ConfigCmd_t (setpoints + tuning + authority flag)

Queue 4: Mode Context Queue (qModeCtx)

- Producer: Display Task (local manual commands) and Logic Task (applied system state)
- Consumer: Logic Task (reads UI intent / local commands) and MQTT Task (reads actuator status)
- Payload: ModeCtx_t (mode + actuator states + timestamps)

7.4 Control Logic Implementation

7.4.1 Control Authority (Local vs Server)

The control decision is based on **authority arbitration**:

- **Server control is active** when a valid config has been received and accessCode != 0.
- If server control is active, the logic uses server mode/setpoints.
- Otherwise, the system follows local UI commands from the display/encoder path.

This ensures the device can run locally even without supervision connectivity, and also accept remote commands when configured.

7.4.2 Operating Modes

The system supports:

- **MODE_OFF**: Fan OFF, Heater PWM = 0
- **MODE_MANUAL**: Actuator states come either from server config (if server authority) or from UI mode context
- **MODE_AUTO**: Automatic computation based on sensor readings and configured setpoints/tuning

7.4.3 Actuator Output Application

Actuators are applied via GPIO:

- Fan: digital ON/OFF
- Heater: PWM output (analogWrite())

The Logic Task runs periodically and updates outputs, then publishes the applied state as the new ModeCtx_t snapshot.

7.4.4 Local UI Command Generation (Manual)

From the OLED menu, the Display Task can:

- toggle fan ON/OFF
- ramp heater PWM smoothly up/down (example ramp to 200 for ON, ramp to 0 for OFF)
- long press triggers emergency OFF (fan off + heater ramp down)

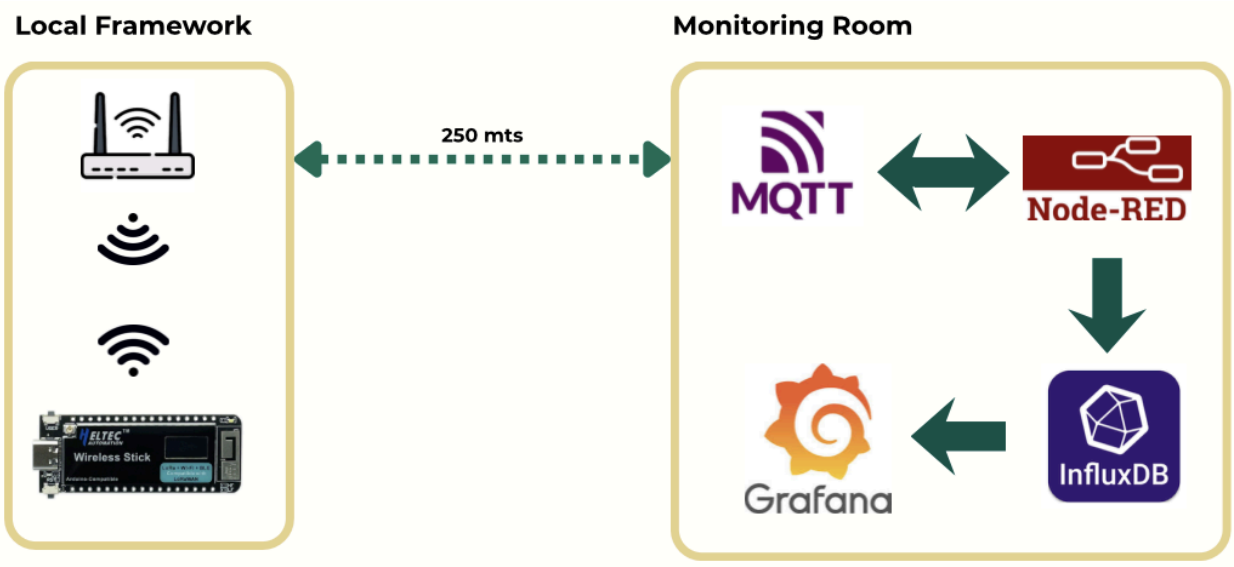
These commands are sent to Logic through the mode context snapshot.

8. Communication & Supervision System

8.1 MQTT Communication Architecture

The communication layer of the system is based on the MQTT (Message Queuing Telemetry Transport) protocol, selected for its lightweight structure, low overhead, and suitability for IoT telemetry systems.

The Heltec WiFi LoRa 32 V3 (ESP32-S3) connects via Wi-Fi to a dedicated Wi-Fi bridge, which establishes a stable 250-meter link to the supervision room. In the supervision room, a Raspberry Pi runs a local MQTT broker (Mosquitto), ensuring that all communication remains within the local ADSL infrastructure.



The communication flow is structured as follows:

1. The Heltec V3 acquires sensor data (SHT31, BH1750).
2. FreeRTOS organizes data into structured telemetry snapshots and organizes respective queues for data storage.
3. The MQTT task publishes telemetry messages to the local MQTT broker in the raspberry pi.
4. Node-RED subscribes to the relevant topics.
5. Node-RED processes and writes data into InfluxDB.
6. Grafana retrieves data from InfluxDB for visualization.

The system supports bi-directional communication:

- Upstream: Telemetry (sensor + actuator states) from MCU to MQTT broker.

- Downstream: Configuration commands (setpoints, modes, overrides) from Node-RED to MCU.

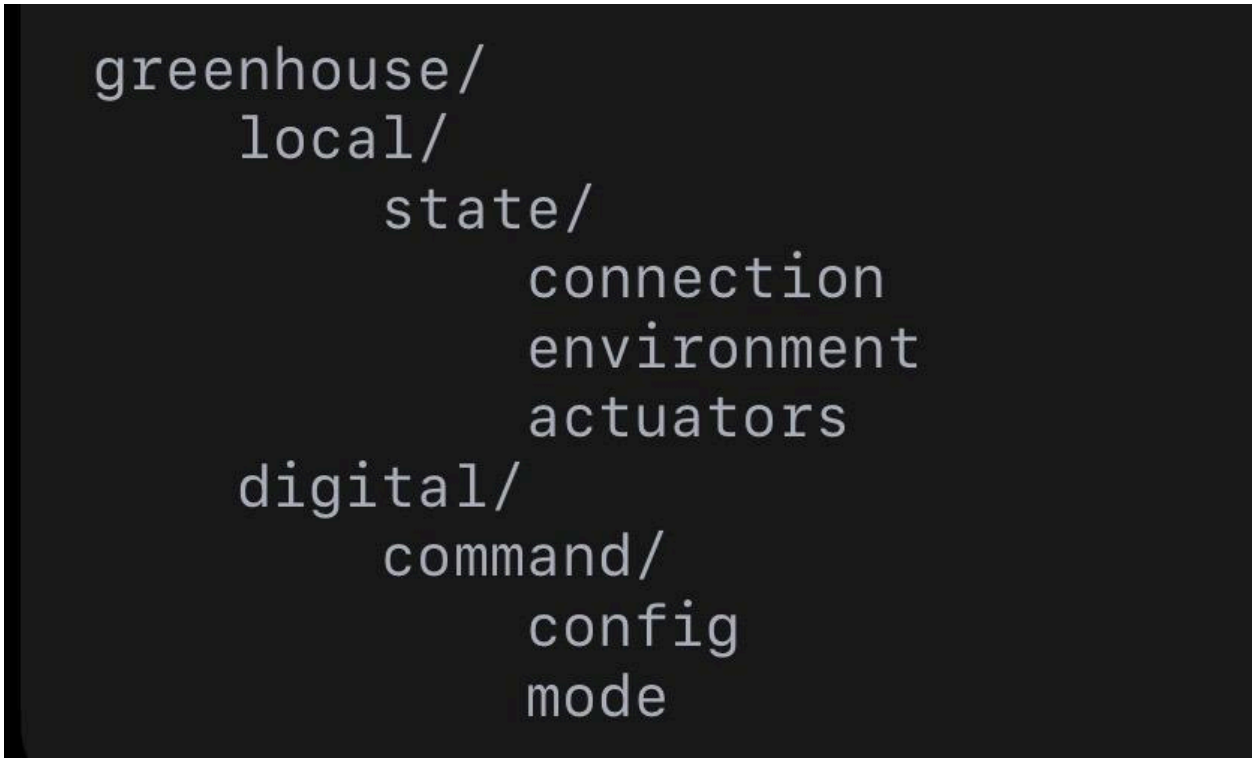
To ensure reliability:

- QoS 1 is used for critical connection and configuration messages.
- QoS 0 is used for periodic telemetry to minimize overhead.
- Retain flags are applied to status topics to allow immediate state synchronization on reconnection.

The architecture ensures that local greenhouse control remains functional even if external internet connectivity is interrupted, as all supervision components operate within the local network.

8.2 MQTT Communication Broker

The MQTT topic structure follows a hierarchical naming convention to separate local framework (MCU layer) and digital supervision layer. An example is illustrated as follows.



The main topics implemented are discussed next.

8.2.1. Connection Status

greenhouse/local/state/connection

Payload:

- device_id
- timestamp_utc
- status (ONLINE / OFFLINE)

QoS: 1

Retain: true

8.2.2. Periodic Telemetry

greenhouse/local/state/environment

Payload includes:

- ambient_light_lux
- air_temperature_celsius
- relative_humidity_percent
- heater_pwm_percent
- fan_enabled
- health metrics (RSSI, free_heap_memory, publish_latency_ms)

QoS: 0

Retain: false

8.2.3. Remote Configuration

greenhouse/digital/command/config

Payload includes:

- setpoint_temperature
- control_mode (OFF / MANUAL / AUTO)
- authority_flag
- timestamp

QoS: 1

Retain: true

This topic structure ensures:

- Clear separation of telemetry and control paths
- Scalable multi-device expansion (future greenhouse nodes)
- Clean integration with Node-RED routing logic

8.3 Network Validation (250m Wi-Fi Range)

A key engineering constraint of the project was the 250-meter separation between the greenhouse and the supervision room, with no cellular coverage available.

To validate network feasibility, the following considerations were applied:

- Use of an outdoor Wi-Fi bridge operating at 2.4 GHz.
- Line-of-sight alignment between the greenhouse and the supervision station.
- Static IP configuration for stable device addressing.
- RSSI monitoring is integrated into telemetry health data.

During testing:

- Stable connectivity was achieved across the 250 m distance.
- Average RSSI values remained within an acceptable range for reliable MQTT communication.
- End-to-end message latency remained below 1 second.
- No packet loss was observed under normal operating conditions.

Given the low data rate requirement of the system (<1 kbps average payload), Wi-Fi provides significant bandwidth margin and ensures stable bidirectional communication without requiring LoRa or cellular solutions.

This validation confirms that the selected Wi-Fi bridge architecture satisfies the operational constraint while maintaining simplicity and scalability.

8.4 Reliability and Fault Handling Strategy

To ensure stable long-term operation over the planned multi-year deployment, the communication system incorporates multiple reliability mechanisms at both network and application levels.

At the network layer, the Heltec V3 implements automatic Wi-Fi reconnection procedures. If connectivity to the Wi-Fi bridge is interrupted, the device periodically attempts reconnection using controlled retry intervals to avoid network flooding. Once Wi-Fi connectivity is restored, the MQTT client automatically reconnects to the broker running on the Raspberry Pi.

At the MQTT layer, a Last Will and Testament (LWT) message is configured. If the device disconnects unexpectedly (e.g., power loss or network failure), the broker automatically publishes an OFFLINE status message on the device connection topic. This allows the supervision system to immediately detect communication loss.

In the event of a supervision network failure, the greenhouse does not depend on external commands to remain operational. The control logic continues to run locally under FreeRTOS, maintaining environmental regulation autonomously. Once communication is restored, the system automatically resynchronizes configuration parameters and publishes its updated status.

Additionally, health monitoring parameters are periodically transmitted, including:

- Wi-Fi signal strength (RSSI)
- Free heap memory
- MQTT publish latency
- System uptime

These indicators allow early detection of degradation or instability before failure occurs.

8.5 Security Considerations

Although the greenhouse system operates within a closed local infrastructure, basic security principles are applied to protect communication integrity and prevent unauthorized access.

The MQTT broker runs on the Raspberry Pi within the supervision room and is not exposed to the public internet. Access to the broker requires authentication using username and password credentials. Topic-level organization ensures logical separation between telemetry and command channels.

All communication occurs within a private network established through the Wi-Fi bridge and ADSL router infrastructure. No external cellular or public network communication is used in the current deployment.

While encrypted MQTT (MQTTS over TLS) was not mandatory due to the isolated network architecture, the system design remains compatible with TLS-enabled secure communication for future extensions. This would be particularly relevant if remote cloud access or public network exposure is introduced.

By combining network isolation, authentication control, and structured topic separation, the system achieves an appropriate level of security for a laboratory-scale greenhouse deployment while maintaining simplicity and maintainability.

9. Dashboard & Visualization (Grafana)

9.1 Dashboard Design

The visualization layer of the system is implemented using Grafana, connected to the InfluxDB time-series database hosted on the Raspberry Pi in the supervision room. The dashboard was designed to provide an intuitive and hierarchical overview of greenhouse operation, combining system health information, environmental variables, and actuator states.

At the top section of the dashboard, the device status of the greenhouse node (greenhouse-heltec-1) is displayed. This includes:

- Connection status (ONLINE/OFFLINE)
- Last communication timestamp (UTC)
- MQTT connectivity confirmation
- Signal quality indicator (RSSI when applicable)

This section allows the operator to immediately verify whether the MCU (Heltec V3) is actively connected and transmitting data.

Below the system status, the three primary environmental variables are displayed prominently as live metrics:

- Ambient light intensity (lux)

- Air temperature (°C)
- Relative humidity (%RH)

These values are updated according to the telemetry publishing interval and represent the current greenhouse conditions.

The layout follows a top-down monitoring logic:

1. Device status
2. Environmental state
3. Actuator behavior

This structure ensures rapid situational awareness for the user at the monitoring room.

9.2 Actuator Monitoring

Actuator states are visualized using both discrete indicators and time-series panels.

The dashboard displays:

- Heater PWM duty cycle (%)
- Fan state (ON/OFF)

For immediate visualization, gauge-style panels are used to show the real-time PWM percentage applied to the heater. The fan status is displayed as a binary state indicator.

Additionally, time-series graphs are included to monitor actuator behavior over time. These graphs allow the user to observe:

- Heater PWM evolution
- Fan activation periods
- Correlation between environmental changes and actuator responses

This enables validation of the control logic and provides insight into how the system reacts to environmental variations.

9.3 Real-Time & Historical Data

The Grafana dashboard provides both real-time monitoring and configurable historical analysis.

Time-series panels allow the operator to visualize actuator performance and environmental variables over adjustable time windows (e.g., last 6 hours, 12 hours, 24 hours, or custom range). This feature supports:

- Performance evaluation of the control strategy
- Detection of abnormal activation patterns
- Analysis of temperature stabilization behavior
- Energy usage interpretation via heater PWM trends

By storing data in InfluxDB, long-term dataset accumulation is enabled, supporting future analysis and optimization studies.

The combined real-time indicators and historical visualization ensure that the greenhouse system is not only operationally transparent but also analytically traceable over extended periods.

For demonstration and presentation purposes, the dashboard refresh rate from the MQTT broker to Grafana was configured at 1-second intervals, allowing near real-time visualization during system validation and live presentations. However, considering the physical nature of the monitored variables (temperature, relative humidity, and ambient light), which exhibit inherently slow dynamics in a greenhouse environment, such high refresh rates are not technically necessary during normal operation.

In practical deployment, the telemetry publishing interval is configured between 15 and 30 seconds, which remains fully sufficient to capture environmental trends while significantly reducing communication overhead and energy consumption. This lower update frequency contributes to a more efficient power profile, particularly in scenarios where the system operates under battery or solar-assisted conditions.

11. Results

11.1 Achievements

The project successfully achieved a complete end-to-end IoT-based smart greenhouse monitoring and control system. Environmental parameters, including temperature, humidity, and light intensity, were reliably acquired using SHT31 and BH1750 sensors and processed locally by the ESP32 microcontroller. The system demonstrated stable bidirectional Wi-Fi communication over a 250-meter distance, despite the constraint of ADSL-only connectivity in the control room. Sensor data was transmitted through MQTT, processed via Node-RED, stored in InfluxDB, and visualized in Grafana dashboards in real time.

A major achievement of the project was the implementation of a modular FreeRTOS-based embedded software architecture. Instead of a traditional monolithic loop structure, the system was divided into independent tasks for sensors, encoder input, display management, control logic, and MQTT communication. This design ensured responsiveness, determinism, and scalability. The queue-based snapshot communication mechanism enabled safe inter-task communication while avoiding race conditions and blocking behavior.

The system also validated remote actuator control. Commands sent from the supervision system were successfully received through MQTT and executed locally to control the fan and heater. In addition, a local user interface using a rotary encoder and OLED display was implemented, enabling manual control and monitoring even without network connectivity. This ensured system autonomy and robustness.

Overall, the project demonstrated full-stack integration from physical sensing and control to cloud-based supervision and visualization, meeting all primary functional requirements defined at the beginning of the project.

11.2 Performance Evaluation

The system exhibited stable operation during continuous testing. Sensor sampling was performed periodically without blocking other tasks, and control logic execution remained consistent due to the priority-based scheduling of FreeRTOS. The logic task maintained deterministic execution intervals, ensuring stable actuator behavior. The use of overwrite-based queues (length = 1 snapshot queues) prevented backlog accumulation and ensured that only the most recent data was processed.

Communication performance was validated over the required 250-meter Wi-Fi link. Telemetry transmission was rate-limited to prevent network congestion while maintaining real-time visualization in Grafana. Latency between sensor measurement and dashboard update remained within acceptable limits for greenhouse applications, where millisecond-level precision is not critical but reliability is essential.

The PWM-based heater control and digital fan control operated correctly under both manual and automatic modes. The system successfully handled authority arbitration between local UI commands and remote server configuration, ensuring consistent and predictable behavior.

Power consumption was kept within acceptable limits for a prototype system, and the modular architecture allows future optimization if ultra-low-power operation becomes a requirement.

12. Challenges & Lessons Learned

One of the most significant challenges faced during the project was the integration of independently developed modules into a unified and stable system. As the system integrator responsible for shared resources, RTOS architecture, and inter-task communication, merging different components required careful synchronization of data models, queue structures, and

control flows. Each team member developed their task module independently, and ensuring compatibility between sensor outputs, display commands, logic decisions, and MQTT communication required multiple iterations and debugging cycles.

A major technical challenge involved inter-task communication and shared state consistency. Without proper design, concurrent tasks can introduce race conditions, blocking behavior, or inconsistent actuator states. Designing a snapshot-based queue architecture with overwrite semantics was a deliberate decision to maintain deterministic behavior and prevent queue overflow. This required refactoring earlier implementations where tasks were more tightly coupled.

Another integration challenge was I²C bus management and peripheral coordination. Ensuring stable communication between sensors and the display while maintaining timing stability required careful initialization and bus recovery handling. Debugging intermittent sensor failures and ensuring proper synchronization between tasks was time-consuming.

From a team perspective, version control and branch merging also introduced complexity. When multiple contributors modify interconnected modules (e.g., data structures in globals, queue definitions, or configuration models), integration conflicts can arise. Aligning structure definitions and maintaining consistency across files required disciplined version control practices and frequent integration testing.

On the communication side, MQTT debugging presented additional challenges. Ensuring correct topic structure, payload format, and authority handling between local and remote control required systematic testing. The arbitration logic between server commands and local UI commands was particularly sensitive and required careful design to prevent conflicting actuator behavior.

Through these challenges, the team learned the importance of modular architecture, clear interface definitions, and early integration testing. A system-level perspective is essential in IoT projects, where hardware, firmware, networking, and backend systems must operate cohesively.

13. Future Improvements

Although the system meets the project objectives, several improvements can enhance its robustness and scalability. First, implementing secure MQTT communication using TLS encryption would strengthen cybersecurity, particularly if deployed outside a local network. Additionally, integrating Over-The-Air (OTA) firmware update functionality would allow remote maintenance and system evolution without physical access to the device.

The control logic could be further improved by incorporating adaptive or predictive algorithms, such as model-based environmental regulation or machine learning-assisted irrigation scheduling. Multi-zone or multi-greenhouse support could also be implemented to scale the architecture for larger agricultural environments.

Power optimization could be enhanced by implementing deeper sleep modes or dynamic frequency scaling if battery or solar operation becomes a priority. A dedicated mobile application interface could complement the Grafana dashboard for improved usability.

Finally, structured automated testing and logging mechanisms could be added to improve maintainability and facilitate long-term deployment.

14. Conclusion

This project successfully demonstrated the design and implementation of an IoT-based smart greenhouse monitoring and control system using a layered architecture and a modular FreeRTOS-based firmware design. By integrating environmental sensing, real-time control logic, Wi-Fi communication, MQTT-based supervision, and dashboard visualization, the system achieved full-stack functionality from the physical layer to the application layer.

Beyond the technical implementation, the project highlighted the complexity of system integration in embedded IoT environments. The transition from independently developed modules to a synchronized, deterministic, and stable system required architectural discipline, careful queue design, and iterative debugging. As a system integrator, managing shared resources and ensuring compatibility across tasks emphasized the importance of well-defined interfaces and structured communication models.

The final prototype validates the feasibility of a scalable, cost-effective, and extensible smart greenhouse solution. The architecture provides a strong foundation for future enhancements in security, automation intelligence, and large-scale deployment.

15. References

1. AM2302 Sensor Datasheet

Aosong Electronics Co., Ltd. (n.d.). Digital relative humidity & temperature sensor AM2302/DHT22 [Data sheet]. Adafruit Industries. https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

2. LoPy4 Specsheet

Pycom Ltd. (2020, March). LoPy4 datasheet (Version 1.1) [Data sheet]. https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_LoPy4_v2.pdf

3. ESP32 Technical Reference Manual

Espressif Systems (Shanghai) Co., Ltd. (2024, February). ESP32 technical reference manual (Version 5.2). https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf

4. Arduino MKR WAN 1310 Tutorial

Arduino. (n.d.). LoRa send and receive. Arduino Documentation. Retrieved May 22, 2024, from https://docs.arduino.cc/tutorials/mkr-wan-1310/lora-send-and-receive/

5. FiPy Specsheet

Pycom Ltd. (2018, January). FiPy datasheet (Version 1.0) [Data sheet]. https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_FiPy_v2.pdf

6. WiFi LoRa 32 V3 Datasheet

Chengdu Heltec Automation Technology Co., Ltd. (2022, September). WiFi LoRa 32 V3.2: LoRa node development kit (Version 1.1) [Data sheet]. https://resource.heltec.cn/download/WiFi_LoRa_32_V3/HTIT-WB32LA_V3.2.pdf