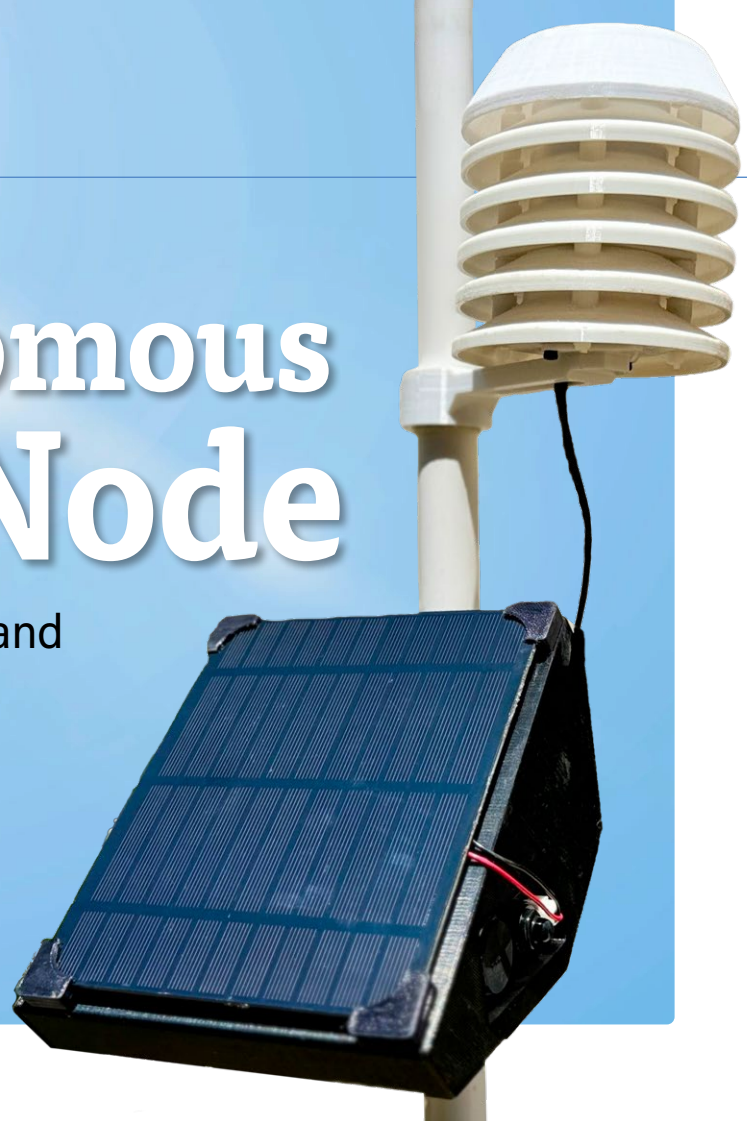
# An Autonomous Sensor Node

## LoRa-Based Data Transmission and Power by Solar Cells



Source: Adobe Stock

By Saad Imtiaz (Elektor)

This Sensor Node project, which uses low-energy and long-range LoRa data transmission, can be used for remote environmental monitoring. Utilizing the LoRa WIO E5 Module and an ESP32-C3 XIAO controller board, it integrates an SCD30 CO2 sensor and a soil moisture sensor, all powered by solar energy. Data is sent to The Things Network (TTN) and can be accessed via Datacake or Home Assistant for real-time insights and automation. This guide covers everything from setting up your LoRaWAN gateway to seamless integration with TTN, Datacake, and Home Assistant.

Remote environmental monitoring is crucial in addressing the challenges of climate change and resource management. However, many of the LoRa sensor nodes available on the market are costly and lack customization options, making them impractical for specific applications and remote areas. To overcome these limitations, this project introduces a versatile, cost-effective LoRa Sensor Node. This custom-built system provides an efficient solution for remote data collection, utilizing long-range LoRa communication and solar power to ensure continuous operation. By integrating customizable sensors, the system offers flexibility for various environmental monitoring needs.

*Figure 1: The Solar Powered LoRaWAN Sensor Node.*

One of the features of this project is its integration with the open-source automation platform Home Assistant, enabling powerful automation capabilities. For example, by measuring the soil moisture, one can automate garden irrigation with this, which can relieve owners from the burden of daily manual watering activities.

In this article, we will explore the setup process of this LoRaWAN Sensor Node seen in **Figure 1**, from configuring a LoRaWAN gateway and integrating with The Things Network (TTN) to visualizing data on platforms like Datacake and Home Assistant. This project demonstrates how affordable and adaptable technology can provide comprehensive monitoring solutions, addressing the shortcomings of existing market options and paving the way for broader applications.

### System Overview

The core of this project is the Seeed Studio XIAO ESP32-C3 microcontroller board. You can find this board and other modules used for this project in the Elektor Store. (See the **Related Products** text box.) The choice for the XIAO board was made due to its compact size and sufficient I/O capabilities required for this project. To enhance its I/O options, the Elektor eXpansion Board [1] was utilized. The eXpansion Board extends the microcontroller's functionality with six I²C connections. Although this project uses only one of these I²C connections, the customizable nature of this sensor node allows for the addition of more sensors if needed. In **Figure 2** you can see the block diagram of the project.

## What Are LoRa, LoRaWAN, a Gateway, and TTN?

**LoRa:** LoRa (Long Range) is a wireless technology designed for long-distance data transmission with minimal power consumption. Operating in sub-gigahertz frequency bands (868 MHz in Europe, 915 MHz in North America), it can transmit data up to 15 km in rural areas and 5 km in urban settings, ideal for IoT applications like environmental monitoring and smart agriculture.

**LoRaWAN:** LoRaWAN (Long Range Wide Area Network) is a network protocol on top of LoRa technology. It manages communication between LoRa devices and gateways, supporting large-scale IoT deployments with secure, bidirectional communication, adaptive data rates, and efficient network usage.

**Gateway:** A LoRaWAN gateway bridges LoRa devices and the internet. It receives data from LoRa devices and forwards it to a central server via Wi-Fi, Ethernet, or cellular connections. Gateways typically cover several kilometers and can handle multiple simultaneous device transmissions.

**TTN (The Things Network):** The Things Network (TTN) is a global, open-source LoRaWAN network. It offers infrastructure and tools for connecting LoRaWAN devices, including device registration, data routing, and integrations with platforms like Datacake and Home Assistant. TTN is supported by a global community, making it an excellent resource for deploying IoT solutions using LoRaWAN.

For LoRa communication, the Seeed Studio E5 WIO LoRa Module was chosen, which can be controlled by a host controller via a UART connection and AT commands for sending data and configuration. Its ease of setup and reasonable price made it a suitable option. This module ensures reliable long-range data transmission. In my testing, I was able to get the range of about 700 m in an urban environment with some trees and houses in the middle. Keeping in mind that my gateway was indoors next to a window, this is a good range, as like any other wireless communication system, line of sight plays a big role. The range can be much improved if the gateway is installed with a high antenna at a height. Also a better antenna can be used on the module itself to further improve its efficiency.

The solar charging system is powered by a Seeed Studio 3 W Solar Panel, connected to Solar Power Management Module by Waveshare which is basically an MPPT Tracker with some protection circuits. This setup charges two 18650 batteries in parallel. The MPPT (Maximum Power Point Tracking) Tracker plays a crucial role in optimizing the power output from the solar panel. It continuously adjusts the electrical operating point of the solar panel, ensuring it operates at its maximum efficiency. After testing multiple MPPT trackers modules, the Waveshare module turned out to be the most efficient and reliable for this application.

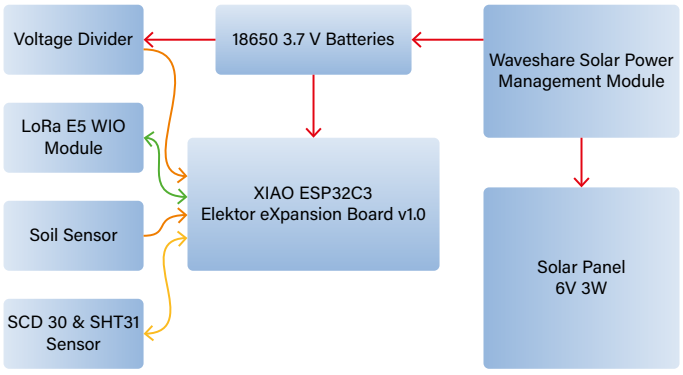



*Figure 2: Block diagram of the project.*

For sensors, the SCD30 by Seeed Studio was selected for its high accuracy in $CO_2$ sensing. As the SCD30 is an NDIR sensor, which is a "true" $CO_2$ sensor, that will tell you the $CO_2$ PPM (parts-per-million) composition of ambient air. Another nice aspect of this sensor is that it comes with an SHT31 temperature and humidity sensor already built-in. Since the SCD30 is not designed for outdoor use as a module, a custom 3D-printed Stevenson enclosure was created to protect from any bad weather coming its way.

The soil moisture sensor used in this project is also from Seeed Studio. Although it was selected due to its availability, it is not ideal for outdoor applications as the connectors and PCB are not waterproof as seen in **Figure 3**. In future iterations, a more suitable soil moisture sensor could be considered.

The Seeed Studio XIAO ESP32-C3 microcontroller communicates with the SCD30 sensor via I2C and the LoRa E5 module via UART. The soil moisture sensor connects to an analog pin, and the battery voltage is

*Figure 3: The soil sensor. As it has open pads and the JST connector, it can cause a short circuit if it catches moisture.*

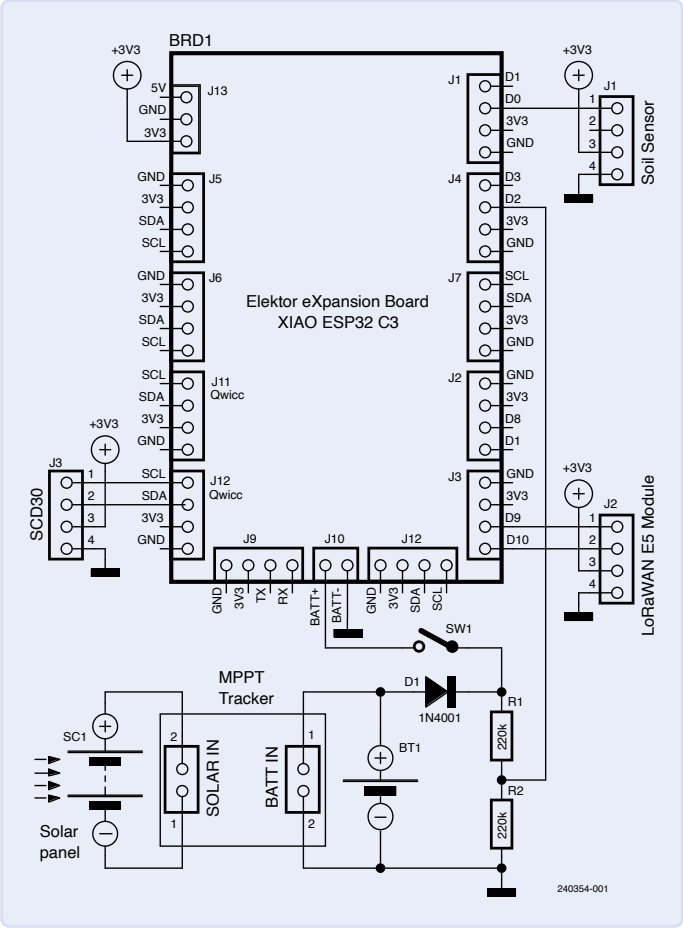



*Figure 4: The schematic diagram of the project.*

monitored using a voltage divider circuit. The circuit diagram of the project can be seen in **Figure 4**.

## Setting Up Your LoRaWAN Gateway

Before we dive into the more technical parts of the project, let's start with the core component: setting up your LoRaWAN gateway. As we are using LoRaWAN to transmit sensor data, it's essential to have a LoRaWAN gateway nearby so that our sensor node can send its collected data to the internet. While you might find some LoRaWAN gateways in your vicinity, you may need to set up your own if none are available.

Setting up your own LoRaWAN gateway is straightforward. First, you need to acquire a gateway; I chose the LPS8v2 LoRaWAN Gateway by Dragino. To use your gateway, you must connect it to a LoRaWAN network. I opted for The Things Network (TTN), but you can also use other networks like Helium, Datacake, and more, even use two networks simultaneously.

Here's a brief overview of the setup process:

- **Connecting to WiFi**: Start by powering on your LoRaWAN gateway and connecting it to your home Wi-Fi network. This enables the gateway to access the internet.

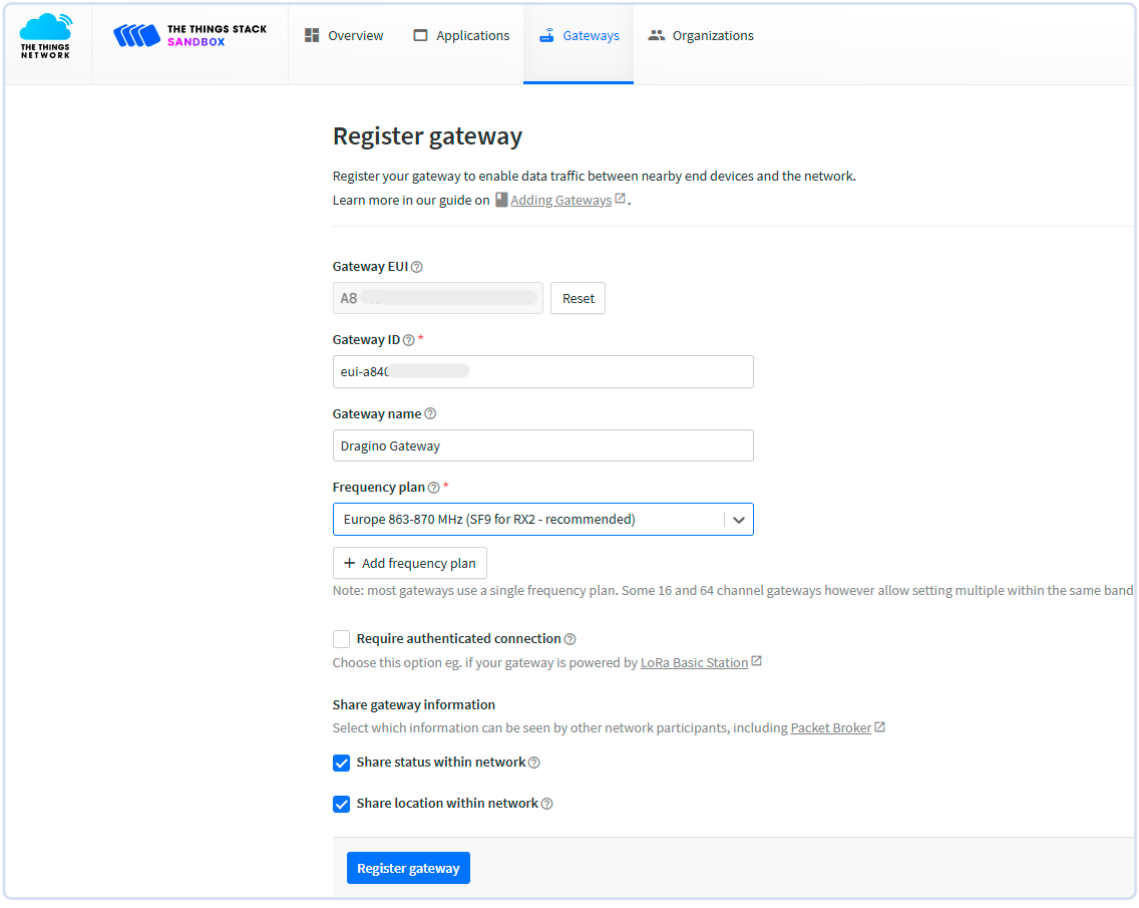



*Figure 5: Adding your gateway to The Things Network.*

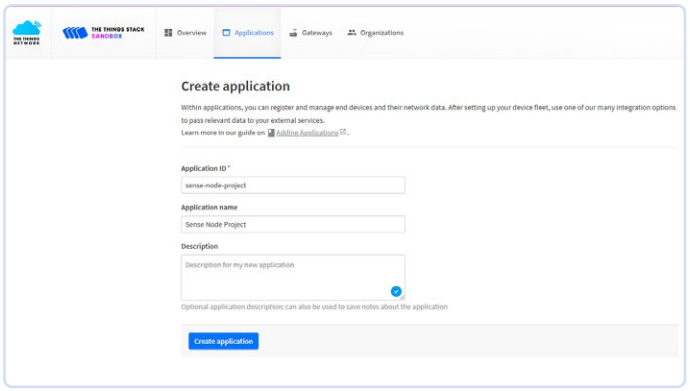



*Figure 6: Creating the application of TTN console.*

- **Registering on TTN**: Next, go to The Things Network console and create an account if you don't already have one. Once logged in, register your gateway by providing necessary details such as the gateway's EUI (a unique identifier for your gateway) and selecting your region, as shown in **Figure 5**.
- **Configuring the Gateway**: Follow the instructions on TTN to configure your gateway. This typically involves entering the network settings, selecting the appropriate frequency plan, and ensuring the gateway is set to communicate with TTN servers.
- **Final Steps**: After configuration, the gateway should connect to TTN and be ready to receive data from your LoRa nodes. You can monitor the status of your gateway on the TTN console to ensure it is functioning correctly.

There are many detailed guides available online [2][3] that provide step-by-step instructions for setting up various types of LoRaWAN gateways. Following these guides can help you troubleshoot any issues and ensure your gateway is correctly configured and connected. With your LoRaWAN gateway set up, now let's register our Sensor Node to the Things Network in the next section.

## Integrating Your Project with The Things Network

Once your gateway is set up, the next step is to integrate the LoRa E5 Module device with The Things Network (TTN) [4]. Begin by registering your device on the TTN console, first you have to create a new application, and then add your device. To create an application, you have to go to the TTN console and click on *Create application*, now just give an ID and a name and then click on *Create application*, as shown in **Figure 6**.

Now you need to register your device on the application you just made. To do so, you go to your application and click on *Register end device.* During this process, you'll obtain essential credentials such as the *Device EUI*, *Application EUI*, and *App Key*, as shown in **Figure 7**. These credentials are critical for configuring your device to communicate with TTN and will be used in the Arduino code discussed later.

After registering your device on the TTN console, you need to set up the payload format to ensure that TTN correctly interprets the data sent from your device. TTN allows you to define a custom payload formatter using JavaScript, which decodes the raw data sent by your device.

Here is a payload formatter in JavaScript:

```
function Decoder(bytes, port) {
  var decoded = {};
```

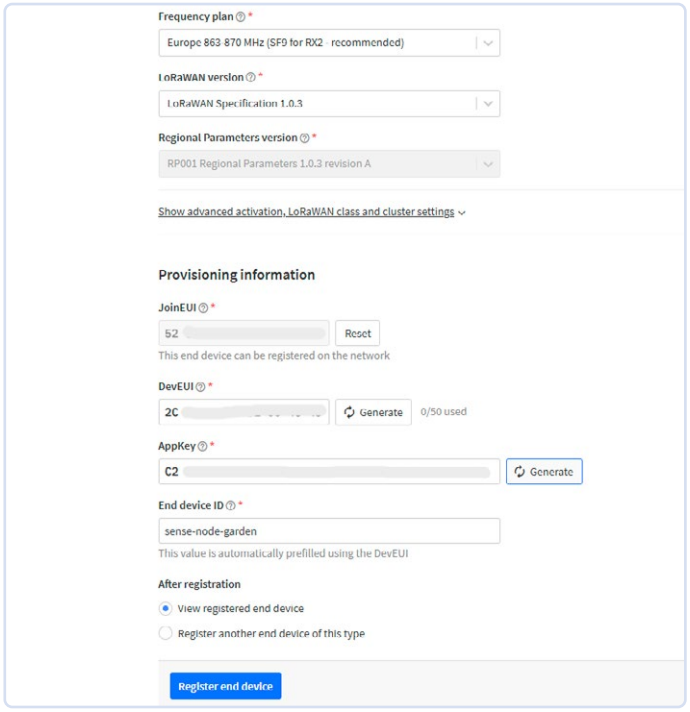



*Figure 7: Registering the end device on the application.*

```
  if (port === 8) {
    decoded.soilMoisture = (bytes[0] << 8) | bytes[1];
    decoded.temp = ((bytes[2] << 8) | bytes[3]);
    decoded.humi = (bytes[4] << 8) | bytes[5];
    decoded.co2 = (bytes[6] << 8) | bytes[7];
    decoded.battery = (bytes[8] << 8) | bytes[9];
  }
  return decoded;}
```

To implement this:

- **Navigate to Payload Formats**: In your TTN console, go to your application and select the *Payload Formats* tab.
- **Select Decoder Function**: Choose the *Decoder* function type.
- **Insert the Code**: Copy and paste the provided JavaScript code into the decoder function editor.
- **Save the Changes**: Save the changes to apply the decoder to your application.

This decoder function processes the bytes received from your sensor node, converting them into readable sensor values such as soil moisture, temperature, humidity, $CO_2$ levels, and battery voltage.

## The Software

The software is programmed to collect and transmit sensor data efficiently while managing power consumption through deep sleep modes. Let's break down the key components of the code and how they work together.

The code starts by including necessary libraries and defining hardware configurations. In the `setup` function, we initialize serial communication, configure the sensors, and set up the LoRa module. This involves setting parameters for the LoRa E5 module, including entering the App Key generated on the TTN console (see below). In **Listing 1**, you can see a minimized version of the code, as the entire code and all hardware files can be accessed on the GitHub repository of this project [5].

## Listing 1: Arduino Sketch (cutout).

```
#include <Arduino.h>
#include <SCD30.h>
#include <HardwareSerial.h>
#include <config.ino>

// Defining Hardware the second internal UART -
// Serial2 for the LoRaWAN E5 Module - Pin 9 and 10
HardwareSerial Serial2(1);

...
//*** Initializing variables ***//

void setup() {

  ...
  //*** Initializing Sensors, Serial 1 & Serial 2 ***//

  // Check if the AT command returns OK
  if (at_send_check_response("+AT: OK", 100, "AT\r\n")) {
    // Set the flag to indicate the LoRa module exists
    is_exist = true;
    // Send AT command to get AppEUI and check the response
    at_send_check_response("+ID: AppEui", 1000, "AT+ID\r\n");
    // Set the LoRa module to LWOTAA mode
    at_send_check_response("+MODE: LWOTAA", 1000, "AT+MODE=LWOTAA\r\n");
    // Set the data rate to EU868
    at_send_check_response("+DR: EU868", 1000, "AT+DR=EU868\r\n");
    // Set the channel number range
    at_send_check_response("+CH: NUM", 1000, "AT+CH=NUM,0-2\r\n");
    // Set the APP Key for authentication, replace with your generated APP Key from TTN
    at_send_check_response("+KEY: APPKEY", 1000,
             "AT+KEY=APPKEY,\"C2XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\"\r\n");
                         // Enter your generated APP Key here.
    // Set the LoRa module to Class A
    at_send_check_response("+CLASS: C", 1000, "AT+CLASS=A\r\n");
    // Set the port number to 8
    at_send_check_response("+PORT: 8", 1000, "AT+PORT=8\r\n");
    // Delay to ensure all commands are processed
    delay(200);
    // Print confirmation that the LoRaWAN setup is complete
    Serial.println("LoRaWAN");
    // Set the flag to indicate the module has joined the network
    is_join = true;
  }
  else {
    is_exist = false;
    Serial.print("No LoRa E5 module found.\r\n");
  }
}

void loop() {
  getSensors();
  sendData();
  // Configure the wake-up source and duration for deep sleep
  esp_sleep_enable_timer_wakeup(10 * 60 * 1000000);
                   // 10 minutes in microseconds
  // Enter deep sleep mode
  esp_deep_sleep_start();
}
```

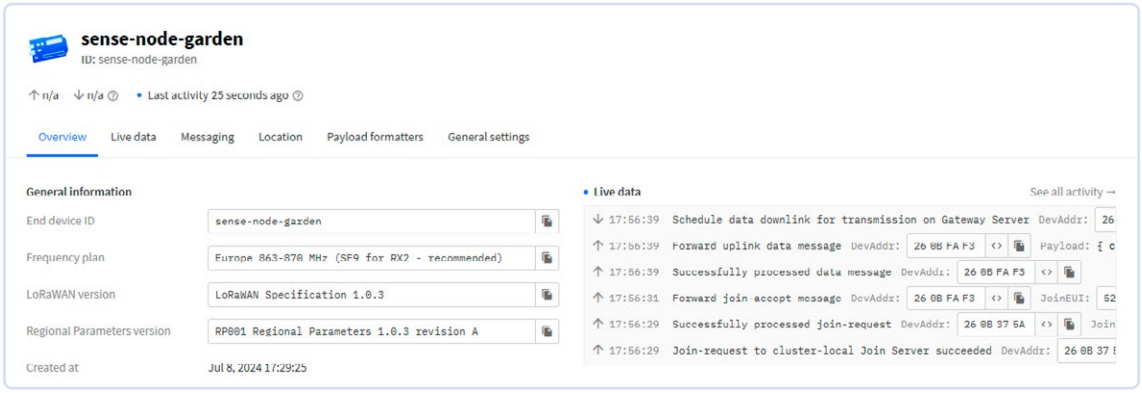



*Figure 8: Data sent from the Sensor Node being displayed on the TTN device overview section.*

The `loop` function is where the main operations happen. It collects sensor data, sends it to TTN, and then puts the microcontroller into deep sleep mode to save power. Using deep sleep mode significantly reduces power consumption by turning off most of the microcontroller's functionalities, only waking it up to perform necessary tasks. By entering deep sleep, the XIAO ESP32-C3 consumes around 142 µA instead of 8 mA, significantly extending the battery life. While in deep sleep mode, the ESP32C3 itself consumes around 5 µA and the rest current is consumed by the onboard voltage regular and battery management IC.The microcontroller wakes up at regular intervals to collect and transmit data before returning to deep sleep.

The `getSensors()` function reads values from the soil moisture sensor and the SCD30 sensor. It also calculates the battery voltage using a voltage divider and averages multiple readings for accuracy. By taking several readings, the system ensures stable and precise measurements.

The `sendData()` function formats the collected sensor data into a payload and sends it to TTN via the LoRa E5 module. It basically converts the sensor data to a hexadecimal string to send it via LoRa, as seen in the code snippet below. This function ensures that the data is correctly formatted and transmitted via LoRa by first joining to the nearest gateway and the sending sensor data, enabling real-time monitoring. After sending the data, the LoRa E5 Module is placed in a sleep mode to save power consumption.

```
// Prepare and send the sensor data
// as a hexadecimal string via LoRa
char cmd[128];
sprintf(cmd, "AT+CMSGHEX=
   \"%04X%04X%04X%04X%04X\"\r\n",
   (int)soilMoisturePercent, (int)temp,
   (int)humi, (int)co2, (int)battery);
```

## The App Key

Before uploading the code, you have to add the App Key, which was generated when you added your device on to the TNN application. It is essential because it authenticates your device with TTN. It is entered in the `setup` function and used in the `at_send_check_response` function.

```
at_send_check_response("+KEY: APPKEY", 1000,
"AT+KEY=APPKEY,\"XXXXXXXXXXXXXXXXX\"\r\n");
// Enter your generated APP Key here
```

This section of the code ensures that your device is registered and authenticated, enabling it to send data to TTN. if all is done correctly, you can see the forward uplink messages from your LoRa Sense Node on your TTN device live data section, as shown in **Figure 8**. You can find the entire project repository on the GitHub [5] which includes all the files for this project.

## Power Consumption

Any system powered by batteries requires special attention to power consumption, as no one wants to constantly recharge the system every few days. Additionally, outdoor systems should require minimal maintenance. Calculating power consumption, testing how much current each component draws, and checking the average current for the system, then calculating the Coulombs for calculation of the battery life and more can be a bit hectic. To overcome these challenges, tools like power profilers or DC energy analyzers can be used to get accurate readings and analyze the power consumption of every component in the entire system.

I recently got my hands on a Joulescope JS200 [6], and I must say, it made the process significantly easier. Its compact size and user-friendly interface make it ideal for all types of power consumption testing. Like any engineer, my initial system didn't have any power consumption optimizations; the priority was ensuring everything worked together. Optimization in code and power consumption comes next.

After the initial system build, I hooked it up with the Joulescope to see what was going on, as seen in **Figure 9**. And the first thing I noticed is the IR LED for the $CO_2$ measurement in the SCD30 sensor takes almost 68 mA every 2 s, resulting in an average current of 25.3 mA

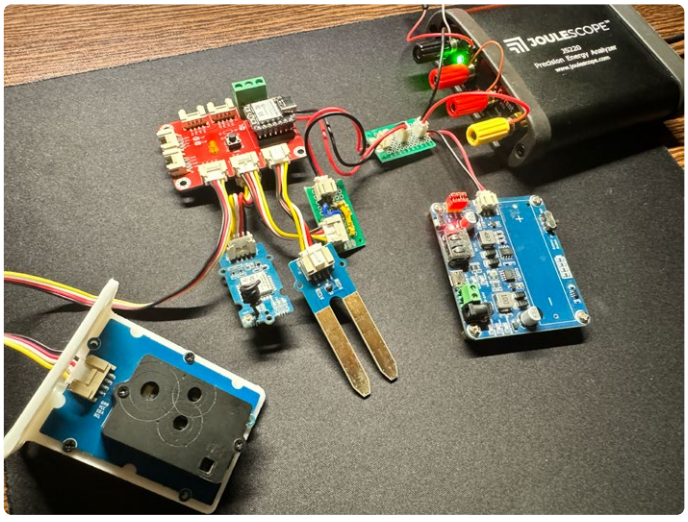



*Figure 9: Power consumption of the project being tested on JouleScope.*

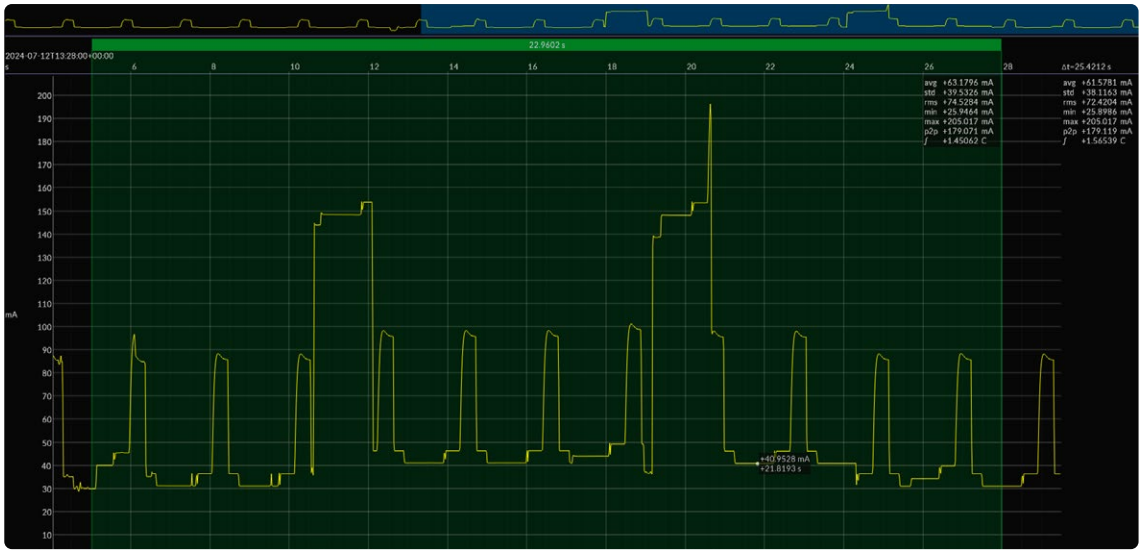


*Figure 10: The current waveform of the SCD30 and LoRa Module.*

over 30 s as seen in **Figure 10**. Upon reviewing the datasheet and the provided library, I found that the default measurement interval was set to 2 s. This was an easy fix as the measurement interval can be controlled, so the IR LED only turns on when a measurement is needed. This reduces the current spikes, lowering the average current draw. However, the SCD30 sensor still draws about 4.5 mA even when idle, and I couldn't find a software-based solution to reduce this.

Next, I focused on the LoRa E5 WIO module, which also drew significant current from the battery. During any LoRa operation, the module takes up to 110 mA, but this is acceptable as it only is for a second. But the main deal is to draw the least current from the battery when the system is idle and the ESP32-C3 is in deep sleep mode, as while the module is idle, it takes up to 10 mA. Which is a significant amount of current when we have to power the system from a battery for a long time. After going through some online resources, I found out that there is a sleep mode AT command which can be used, to put the module into sleep mode, reducing the idle current draw to 51.7 µA, which was a substantial improvement. In **Figure 11** you can see the current waveform of the LoRa E5 module before and after the sleep mode. The current draw of LoRa module can be further reduced to even 3 µA, if the LDO on the module is removed.

During further testing, I found that the soil moisture sensor increased the system's current draw by up to 4 mA when detecting moisture. This sensor uses capacitance to measure moisture, causing excessive power consumption as the soil becomes more conductive. This also leads to another issue: the sensor's bare tracks act as an electrolysis device, causing corrosion and mineral deposits on the probes. To mitigate the corrosion issue, a coated soil sensor can be used. However, to eliminate the excessive current when the moisture level increases requires cutting the power supply to the sensor when measurement is not required, which is a more hardware-based solution. For the first project version, I kept it simple, focusing on software-based optimizations. Future versions will include more hardware-based solutions.

## Battery Life

This moves us to the part of calculating the system's battery life, which was made significantly easy with the Joulescope. After all software-based power optimizations, I connected the system to the Joulescope. In **Figure 12** you can see the current waveform of the entire system after optimizations. Notice that the SCD30 turns the IR LED only during the ESP32-C3 is awake, and then stops taking measurements after the ESP32-C3 goes to deep sleep mode.

Using its accrue feature in its multimeter mode, I measured the average current and Coulombs consumed by the system over 10 minutes, with new sensor values being sent via LoRa every minute. As seen in **Figure 13**, the average current was 19.10 mA, which is high for my goal of a week-long operation on a single charge. Reducing the data reading frequency to every 10 minutes lowered the average current to 4.905 mA, making it acceptable for this application.

The number of coulombs consumed over time is needed to measure the battery life, which was 28.1403 C in 1 h and 35 min (5700 s), this was measured on the Joulescope. After that, we need to convert the battery capacity of our batteries to coulombs. I was using two Li-ion 18650 batteries in parallel with each having the battery capacity of 1800 mAh.

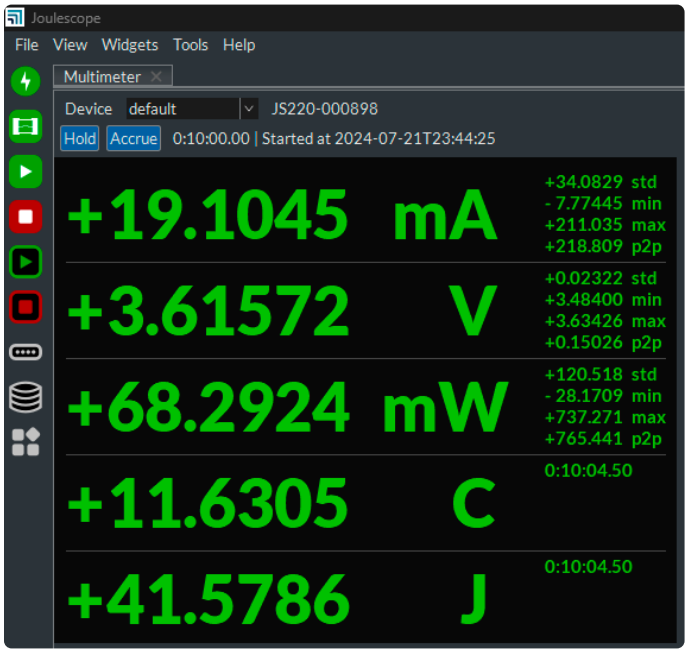



*Figure 13: Screenshot of Multimeter mode in JouleScope.*

*Figure 11: Current consumption of LoRa E5 Modoule, before (left) and after (right) implementing the sleep function.*

*Figure 12: Current waveform of the entire system, after power optimizations.*

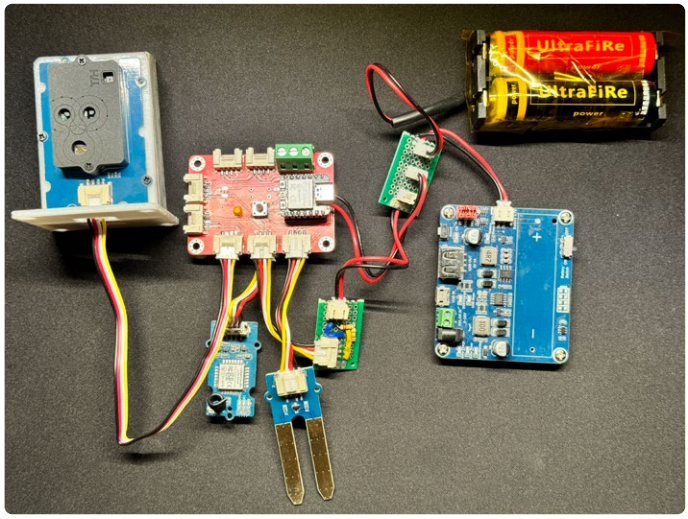



*Figure 14: All the project components assembled and ready to be placed into the enclosure.*

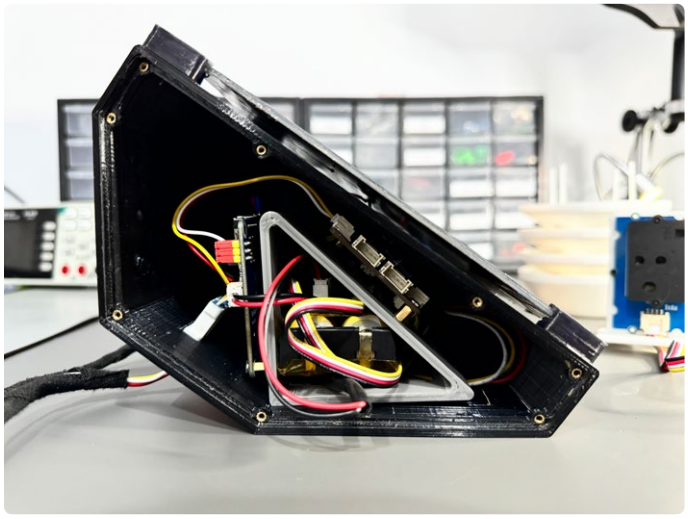


*Figure 15: Components placed inside the main enclosure with Solar Panel mounted on the top.*

Battery capacity in C (coulombs):

3.6 Ah × 3600 s/h = 12960 C

As the cut-off voltage is around 3.3 V, which is higher than the 3 V (which is the voltage when the battery is completely drained), we have to assume around the 80 to 90% usable capacity. Keeping in mind the self discharge rate of the batteries, let's use 85% as an approximation:

0.85 × 12960 C = 11016 C

Now let's calculate the consumption rate in coulombs per second:

28.1403 C / 5700 s = 0.004936 C/s

And now, finally, to calculate the battery life by dividing the usable battery capacity by the consumption rate to find the battery life in hours:

(11016 C / 0.004936 C/s) / 3600 s/h = 619.93 h.

Which is about 25 days of battery life on a single charge. Without any power optimizations, the entire system was going to last only six days! After I was satisfied with the power consumption after removing some more unwanted LEDs, I prepped the system for deployment, as seen in **Figure 14**.

## Components and Enclosure

A custom 3D-printed enclosure [5] was designed to house all the components of the LoRa Sensor Node project. The main enclosure was designed to be weather-resistant which enclosed all the components as seen in **Figure 15.** This required careful consideration of several factors during its design. The enclosure ensures that all electrical connections are waterproof, with special waterproof JST JWPF connectors used to connect external sensors and the solar panel power to the Solar Power Management Module. Epoxy was applied to fill any gaps between the connectors and the enclosure housing.

A waterproof button was installed to reboot or turn the system on or off, and a waterproof USB Type-C connector was used to upload new firmware or debug any issues without opening the enclosure, as seen in **Figure 16**. This port also allows for emergency battery charging if the weather is bad for several days and the batteries die due to lack of sunlight.

*Figure 16: Side view of the installation LoRa Sensor Node, showing the waterproof USB Type connector and the JST waterproof connectors.*

For the SCD30 temperature, humidity, and $CO_2$ sensor, a custom Stevenson enclosure as seen in **Figure 17** was designed to protect the sensor from rain, as it is not intended for outdoor use. This Stevenson screen can also house other sensors, with a component holder inside based on the Grove module PCB layout, allowing up to four small Grove sensors to be mounted.

After installing the system as shown in **Figure 18**, its waterproof capabilities were put to the test quite the very next day. Following a week of heavy rainfall and a storm, the system remained dry inside, and all components continued to function perfectly.

## Integration with Datacake

Integrating your LoRa Sensor Node with Datacake [7] allows for efficient data visualization and management. The process begins in The Things Network (TTN) console, where you first log in and navigate to your application. Under the *Integrations* tab, select *Webhooks* and click on *Add Webhook*. Choose Datacake from the list of available

*Figure 17: Low-angle view of the LoRa Sensor Node, showcasing the Stevenson enclosure.*

webhook templates and configure it by entering the necessary details, including your Datacake API Key, which can be found in your Datacake account under the *API* section.

Next, create an account on Datacake if you haven't already, and set up a new device corresponding to your LoRa Sensor Node. During the device setup, provide details such as the Device EUI and link it to The Things Network. This ensures that the data sent from TTN is received by Datacake.

*Figure 18: LoRa Sensor Node deployed.*

In the TTN console, navigate to your application and go to the *Payload Formats* tab. For Datacake we'll be using the same payload formater to decode the data we get from TTN. So simply just paste the same java script payload formatter code in the payload decoder section in configuration tab of device in Datacake and save the changes once the configuration is complete.

Now, return to Datacake and navigate to the dashboard of the device you created. Add fields to display the values of the sensors, such as soil moisture, temperature, humidity, $CO_2$ levels, and battery status. Set the appropriate data type for each field and map the TTN payload fields to the corresponding Datacake fields as shown in **Figure 19**. This mapping ensures that the data received from TTN is correctly displayed on your Datacake dashboard.

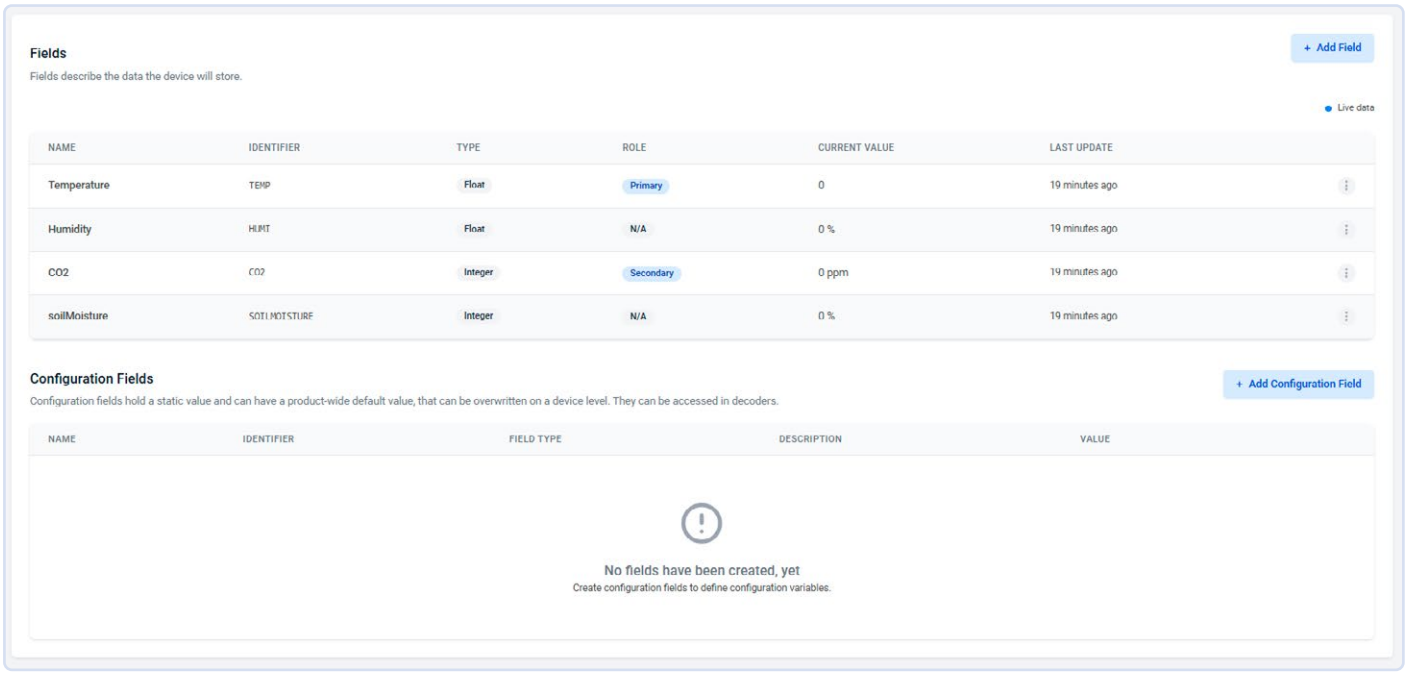



*Figure 19: Datacake field configuration of sensor data received from TTN.*

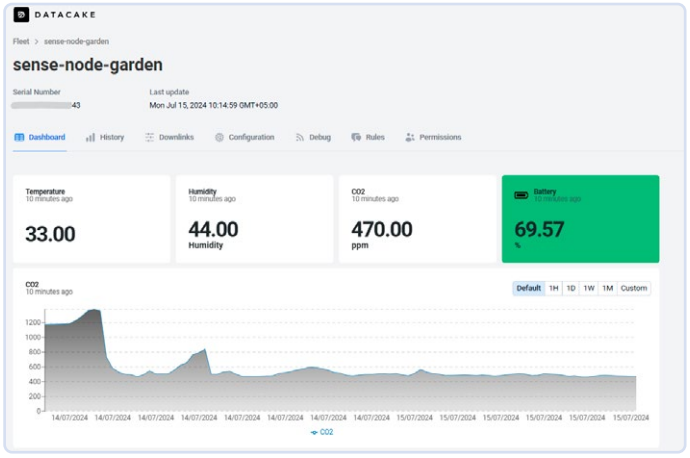



*Figure 20: Datacake dashboard showing all the sensor data collected.*

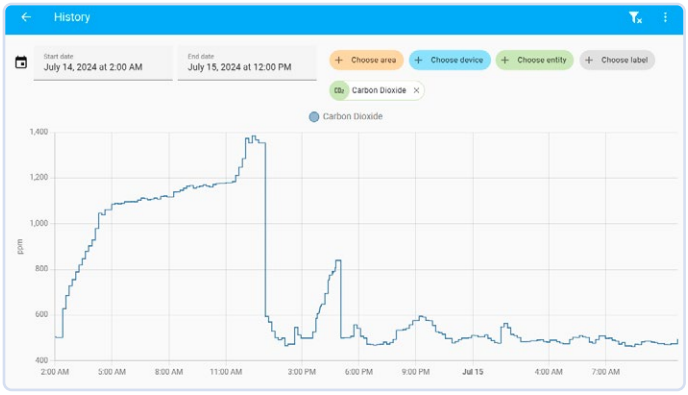



*Figure 21: History line graph in Home Assistant.*

Finally, verify that the data from your LoRa Sensor Node is being transmitted to TTN and forwarded to Datacake. Your Datacake dashboard should now display real-time sensor values, allowing you to monitor and analyze the data efficiently, as shown in **Figure 20**. This integration provides a powerful platform for visualizing and managing your sensor data, making it easier to track environmental conditions and make informed decisions based on the collected data.

## Integration with Home Assistant

Integrating your LoRa Sensor Node with Home Assistant [8] provides an excellent solution for long-term data storage and analysis without the storage limitations imposed by platforms like Datacake. Since Home Assistant runs on your home instance, there are no storage space or data point limits, making it ideal for continuous monitoring.

To integrate Home Assistant with The Things Network (TTN), start by generating an API Key in the TTN console. Navigate to your application, go to the *API Keys* section, and create a new key with the necessary permissions. Also, make sure you enable *Storage Integration* in *Integrations* in the TNN application dashboard; otherwise the sensor data won't show up in Home Assistant.

Next, in Home Assistant, go to *Devices & Services* and add *The Things Network* integration. When prompted, enter the name of your application in TTN and the API Key you generated. This will link your Home Assistant instance with your TTN application, enabling data flow from your LoRa Sensor Node to Home Assistant.

To visualize the sensor values in a line graph as seen in **Figure 21**, you need to define the unit of measurement for each sensor entity received from The Things Network integration. Add the following entries to your *configuration.yaml* file in Home Assistant:

```
sensor.lora_sense_node_temperature:
  unit_of_measurement: "°C"
sensor.lora_sense_node_co2:
  unit_of_measurement: "ppm"
sensor.lora_sense_node_humidity:
  unit_of_measurement: "%"
sensor.lora_sense_node_soilmoisture:
  unit_of_measurement: "%"
```

In these entries, `sensor.lora_sense_node_temperature` is the entity name where Home Assistant receives the temperature values from TTN. Similarly, define the unit of measurement for $CO_2$, humidity, and soil moisture sensors. This configuration ensures that the sensor values are displayed correctly on your Home Assistant dashboard.

By completing these steps, you can have a detailed and interactive card on your Home Assistant dashboard, as seen in **Figure 22**, showing all the sensor values your LoRa Sensor Node is sending. This setup provides a robust and flexible solution for monitoring and analyzing environmental data, leveraging the storage capabilities and customizable interface of Home Assistant.

## Future Improvements and Potential for Various Applications

While the current LoRa Sensor Node system is effective, there is significant room for future improvements, particularly in power efficiency. At idle, the system draws approximately 4.32 mA, which is quite high for long-term battery-powered applications. By integrating an external Real-Time Clock (RTC) with a power latch circuit, we can drastically reduce the idle current draw to as low as 50 nA.

In this improved setup, the external RTC would be controlled by the microcontroller, and the RTC would manage the power latch circuit. This circuit would cut off power to the entire system when it's not actively sending sensor values and power it back up at set intervals. This approach would significantly extend the battery life, making the system more suitable for remote and long-term deployments. This solution can be implemented in the next version of the eXpansion board or as a separate module that can be added to the existing setup. Watch out for an article about that!

Additionally, there are other hardware optimizations that can be explored. For example, replacing high-power consumption components with more efficient alternatives, optimizing the firmware to ensure minimal power usage, and enhancing the efficiency of the solar charging system are all potential improvements.

The modular nature of this system allows for the addition of various sensors to extend its functionality. By integrating more sensors, this setup can be used for a wide range of applications, including but not limited to:

- **Environmental Monitoring**: Adding sensors for air quality, light intensity, and sound levels can make this system a comprehensive environmental monitoring station.
- **Agricultural Applications**: Integrating soil pH, nutrient sensors, and weather sensors can provide valuable data for precision farming.

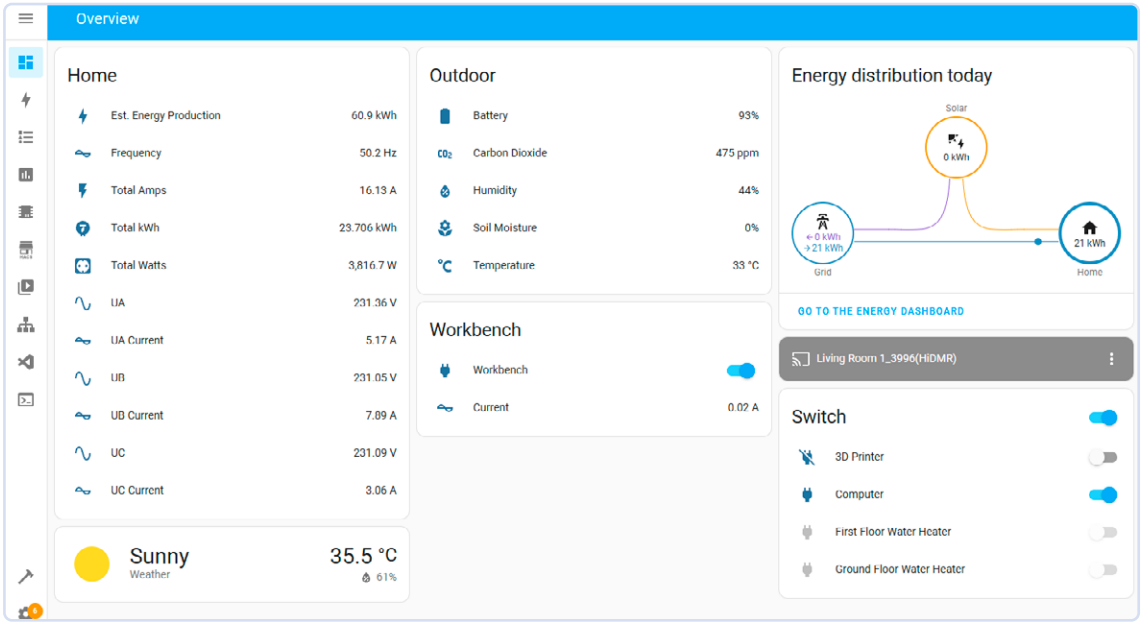



*Figure 22: LoRa Sense Node data being displayed on the Home Assistant dashboard.*

- **Industrial Monitoring**: Including sensors for gas leakage, vibration, and pressure can help in monitoring industrial environments for safety and efficiency.

In conclusion, the current LoRa Sensor Node system provides a robust and flexible platform for remote environmental monitoring. With future improvements focused on power efficiency and additional sensor integration, this system has the potential to be adapted for various applications, making it an invaluable tool for diverse monitoring needs.

240354-01



## About the Author

Saad Imtiaz, Senior Engineer at Elektor, is a mechatronics engineer with extensive experience in embedded systems and product development. His journey has seen him collaborate with a diverse array of companies, from innovative startups to established global enterprises, driving forward-thinking prototyping and development projects. With a rich background that includes a stint in the aviation industry and leadership of a technology startup, Saad brings a unique blend of technical expertise and entrepreneurial spirit to his role at Elektor. Here, he contributes to project development in both software and hardware.

## Questions or Comments?

If you have questions about this article, feel free to email the author at saad.imtiaz@elektor.com or the Elektor editorial team at editor@elektor.com.

## Related Products

- **Seeed Studio XIAO ESP32C3**
  www.elektor.com/20265
- **Seeed Studio Grove SCD30 CO2**
  www.elektor.com/20012
- **Seeed Studio LoRa-E5 STM32WLE5JC**
  www.elektor.com/19956
- **Seeed Studio Solar Panel (3 W)**
  www.elektor.com/19131
- **Waveshare Solar Power Management Module**
  www.elektor.com/20488
- **Dragino LoRa/LoRaWAN IoT Kit v3 (EU868)**
  www.elektor.com/20775

## WEB LINKS

[1] Saad Imtiaz, "Elektor eXpansion Board v1.0," Elektor 7-8/2024: https://elektormagazine.com/240250-01
[2] Setting up The Things Network V3 on Dragino: http://wiki.dragino.com/xwiki/bin/view/Main/Notes%20for%20TTN/
[3] Dragino LPS8N - Setup with The Things Network: https://www.thethingsindustries.com/docs/gateways/models/dragino-lps8/
[4] The Things Network: https://www.thethingsnetwork.org/
[5] LoRa Sensor Node Github Repository: https://github.com/ElektorLabs/lora-sensor-node
[6] Joulescope JS220: Precision Energy Analyzer: https://www.joulescope.com/products/js220-joulescope-precision-energy-analyzer
[7] Datacake: https://datacake.co/
[8] Home Assistant: https://www.home-assistant.io/