

ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE UN PROTOTIPO CON SOC PARA CONTROL DE INVERNADERO MEDIANTE COMUNICACIÓN LORA

**TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO
REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR
EN REDES Y TELECOMUNICACIONES**

VICTOR SAMUEL INLAGO GUALOTUÑA

victor.inlago@epn.edu.ec

DIRECTOR: LEANDRO ANTONIO PAZMIÑO ORTIZ

leandro.pazmino@epn.edu.ec

DMQ, JULIO 2024

CERTIFICACIONES

Yo, VICTOR SAMUEL INLAGO GUALOTUÑA declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Víctor Samuel Inlago Gualotuña

Victor.inlago@epn.edu.ec

Samuel.inlago.ec@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por NOMBRE_ESTUDIANTE, bajo mi supervisión.

LEANDRO PAZMIÑO ORTIZ

DIRECTOR

leandro.pazmino@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Víctor Samuel Inlago Gualotuña

C.I. 1727622217

DEDICATORIA

A Dios, Este logro se lo dedico totalmente a Dios quién me supo dar fuerzas y energía en todo momento cuando sentía no poder, le doy toda mi gratitud y adoración por llenarme de sabiduría e inteligencia. Agradezco también por cada desafío que me ha tocado enfrentar en el transcurso de mis estudios, pues en ellos encontré oportunidades para crecer, aprender, ser una persona determinada y sobre todo por formarme como un excelente profesional.

AGRADECIMIENTO

Quiero expresar mi total gratitud a todos mis profesores de la universidad, pues en ellos adquirí el conocimiento suficiente para enfrentar varios desafíos. Sus conocimientos, exigencias y sobre todo su amor por dictar clases.

A mis padres por brindarme su apoyo económico y emocional cuando lo necesité y a mis compañeros de la universidad, pues en ellos conocí la amistad, el apoyo y, además, compartimos conocimientos entre todos en las materias más complicadas.

ÍNDICE DE CONTENIDOS

CERTIFICACIONES.....	I
DECLARACIÓN DE AUTORÍA.....	II
DEDICATORIA.....	III
AGRADECIMIENTO.....	IV
RESUMEN	VIII
<i>ABSTRACT</i>	IX
1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO	10
1.1 Objetivo general.....	11
1.2 Objetivos específicos.....	11
1.3 Alcance	11
1.4 Marco Teórico.....	11
Internet de las cosas (<i>IoT</i>).....	11
<i>LoRa</i>	12
<i>Heltec WiFi LoRa 32 (V3)</i>	12
Sensor <i>DHT22</i>	13
<i>System on a Chip (SoC)</i>	13
<i>Adafruit IO</i>	13
Protocolo <i>MQTT</i>	14
<i>Arduino IDE</i>	15
Invernadero	15
Humedad de suelo	15
Humedad Ambiental	15
2 METODOLOGÍA.....	16
3 RESULTADOS	16
3.1 Identificación de los requerimientos del prototipo	17
Dispositivo <i>System-on-a-Chip (SoC)</i>	17
Sistema de comunicación inalámbrica	17

Dispositivos de monitoreo y control de irrigación.....	17
Banda de frecuencia	18
Plataforma de desarrollo de <i>software</i> libre y de código abierto	18
Código de programación para el SoC	18
Plataforma de monitoreo <i>IoT</i>	19
Prototipo de invernadero	19
Diseño y construcción de un circuito electrónico	19
3.2 Selección del <i>Hardware</i> y <i>Software</i>	19
Selección del <i>Hardware</i>	19
Determinación del <i>Software</i>	27
3.3 Diseño del prototipo para control de invernadero.....	30
Diseño esquemático general del proyecto.....	30
Selección de plataforma para la programación	31
Creación de códigos en <i>Arduino IDE</i>	32
Configuración de panel de control <i>Web</i> en <i>Adafruit IO</i>	52
Configuración de <i>actions</i> en <i>Adafruit IO</i> para enviar alertas al correo electrónico	59
3.4 Implementación del prototipo para control de invernadero.....	62
Implementación del código en el SoC maestro y el SoC esclavo	62
Implementación del circuito electrónico	63
Implementación de protección al <i>PCB</i> y microcontrolador esclavo	68
Implementación de la maqueta	70
Implementación del Panel de Control.....	71
Implementación de alertas en <i>actions</i> en <i>Adafruit IO</i>	72
3.5 Realizar las pruebas de funcionamiento	73
Inicialización de todos los componentes del proyecto	73
Prueba de monitoreo de temperatura ambiental y humedad del suelo en el prototipo de invernadero	76
Prueba de control del sistema de irrigación.....	78
Prueba del ajuste de parámetros	79

	Prueba del sistema de alerta para condiciones anormales en el prototipo de invernadero por medio de Acciones en <i>Adafruit IO</i>	82
	Resumen de las pruebas de funcionamiento.....	83
	Costo del prototipo	84
4	CONCLUSIONES.....	85
5	RECOMENDACIONES	86
6	REFERENCIAS BIBLIOGRÁFICAS	87
7	ANEXOS	91
	ANEXO I: Certificado de Originalidad	i
	ANEXO II: Enlaces	ii
	ANEXO III: Códigos Fuente.....	iii
	Código fuente del SoC esclavo	iii
	Código fuente SoC maestro	viii

RESUMEN

La implementación de un prototipo basado en SoC para el control de invernadero, utilizando comunicación *LoRa* y una interfaz web, ofrece una solución innovadora. La tecnología *LoRa* permite el monitoreo de dispositivos a largas distancias con un bajo consumo de energía, lo que posibilita a los usuarios supervisar de forma remota sus invernaderos sin la necesidad de estar físicamente presentes.

El sistema se estructura en torno a tres pilares fundamentales: el primero se centra en un SoC esclavo que lee los parámetros ambientales de los sensores y controla el sistema de irrigación, activándolo o desactivándolo según sea necesario. El segundo pilar consiste en la comunicación bidireccional entre dos SoC mediante tecnología *LoRa*, permitiendo el intercambio de información entre ellos. El tercer pilar implica un SoC maestro que recibe los datos del SoC esclavo a través de *LoRa* y los transmite a una plataforma *IoT* mediante una conexión *WiFi*, donde se dispone de un panel de control para monitorear los datos. Además, este dispositivo maestro también envía datos a través de *LoRa* para activar el sistema de irrigación.

En la primera sección se establecieron los objetivos y se delimitaron los alcances del proyecto. Además, se detallaron los conceptos esenciales para el marco de desarrollo de este. También, en la segunda sección se ofrece una descripción detallada de la metodología utilizada para alcanzar los objetivos previamente establecidos.

En la tercera sección se presentan los resultados obtenidos durante el desarrollo del proyecto. Se abordan los requisitos del proyecto, la selección de componentes, el diseño del código para la programación de los SoC, y la integración de estos con *Adafruit IO*.

En la cuarta sección se destaca la implementación y validación del sistema mediante pruebas en una maqueta. Estas pruebas se centran en el monitoreo de parámetros ambientales y la humedad del suelo, así como en el control de un sistema de irrigación.

Finalmente, en la sección final se exponen las conclusiones del proyecto, destacando la eficacia del sistema. También se incluyen recomendaciones y referencias bibliográficas. Los anexos contienen los dos códigos fuente y un video demostrativo el funcionamiento.

PALABRAS CLAVE: *LoRa*, *SoC*, *Arduino IDE*, *Adafruit IO*, invernadero, *WiFi*, sensores

ABSTRACT

The implementation of an SoC-based prototype for greenhouse control, using LoRa communication and a web interface, offers an innovative solution. LoRa technology enables monitoring of devices over long distances with low power consumption, allowing users to remotely monitor their greenhouses without the need to be physically present.

The system is structured around three fundamental pillars: the first focuses on a slave SoC that reads the environmental parameters from the sensors and controls the irrigation system, activating or deactivating it as necessary. The second pillar consists of bidirectional communication between two SoCs using LoRa technology, allowing the exchange of information between them. The third pillar involves a master SoC that receives data from the slave SoC via LoRa and transmits it to an IoT platform via a WiFi connection, where a control panel is available to monitor the data. Additionally, this master device also sends data via LoRa to activate the irrigation system.

In the first section, the objectives were established, and the scope of the project was defined. In addition, the essential concepts for its development framework were detailed. The second section describes in detail the methodology used to achieve the previously established objectives.

The third section presents the results obtained during the development of the project. The project requirements, component selection, code design for SoC programming, and their integration with Adafruit IO are addressed.

The fourth section highlights the implementation and validation of the system through testing on a model. These tests focus on monitoring environmental parameters and soil moisture, as well as controlling an irrigation system.

Finally, in the last section the conclusions of the project are presented, highlighting the effectiveness of the system. Recommendations and bibliographical references are also included. The annexes contain the source code and a video demonstrating how it works.

KEYWORDS: LoRa, SoC, Arduino IDE, Adafruit IO, greenhouse, WiFi, sensors.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

En la actualidad, el desarrollo tecnológico y la Internet de las Cosas están transformando numerosos aspectos en nuestra vida cotidiana, incluida la agricultura. En este progreso tecnológico ha tenido un gran impacto el cultivo de plantas en invernaderos. En lugar de depender únicamente de métodos tradicionales, como la vigilancia de una persona de manera constante y física de las plantas, el uso de la *IoT* está revolucionando la forma en que gestionamos y controlamos los invernaderos y cultivos de plantas.

Con la llegada de la *IoT*, esta tarea se ha vuelto mucho más eficiente y precisa, puesto que los sistemas *IoT* permiten monitorear de manera continua y en tiempo real una variedad de variables, como la temperatura, humedad del suelo y la intensidad de luz solar, y todo esto de manera automatizada y remota.

La implementación de un prototipo con *SoC* para el control de invernadero mediante comunicación *LoRa* y a través de una interfaz web presenta una solución innovadora, puesto que permite monitorear un invernadero de manera remota verificando los cambios de temperatura ambiental y humedad del suelo a larga distancia de manera remota e inalámbrica usando comunicación *LoRa* y *WiFi*.

Para realizar este proyecto se ha empleado dos módulos *Heltec WiFi LoRa 32 (V3)* los cuales permiten una comunicación a larga distancia y de bajo consumo usando una tecnología de comunicación inalámbrica *LoRa*. Además, estos módulos llevan incorporado otras tecnologías como son el *Bluetooth* y *WiFi*, en este proyecto se aprovechó la tecnología *WiFi*, la cual permitirá conectarse a la red *WiFi* y a su vez la redireccionar información hacia la nube que se refleja en una interfaz web.

En este proyecto también se utilizó un sensor digital *DHT22 AM2302* que tiene la capacidad de medir temperatura y humedad ambiental en el prototipo de un invernadero y este sensor a su vez redirecciona la información recopilada hacia el *SoC* esclavo. También se utilizó otro sensor de humedad del suelo *FC-28* que de igual manera envía la información recopilada al *SoC* esclavo. Otro elemento muy importante que se utilizó es una mini bomba controlada por un relé que permite realizar un sistema de control de sistema de irrigación. Todos los elementos mencionados anteriormente permitieron monitorear el prototipo de invernadero y a su vez toda esa información recopilada por el sensor enviar a la nube por medio de comunicaciones inalámbricas como son *LoRa* y *WiFi*.

1.1 Objetivo general

Implementar un prototipo con SoC para control de invernadero mediante comunicación *LoRa*.

1.2 Objetivos específicos

- Definir los requisitos técnicos y funcionales para el control del invernadero.
- Seleccionar el *hardware* y *software* acorde a los requerimientos establecidos.
- Diseñar el prototipo del sistema de control del invernadero.
- Implementar el prototipo del sistema de control del invernadero.
- Realizar pruebas de funcionamiento del prototipo.

1.3 Alcance

El alcance del proyecto comprende la implementación de un prototipo funcional de control de invernadero que incluye las siguientes funcionalidades:

Monitoreo de temperatura ambiental y humedad del suelo en el invernadero.

Control del sistema de irrigación.

Comunicación a través de tecnología *LoRa* para la transmisión de datos de forma remota.

Implementación de un panel de control web para supervisión y ajuste de parámetros.

Diseño de un sistema de alerta en caso de condiciones anormales en el invernadero.

1.4 Marco Teórico

Internet de las cosas (*IoT*)

El Internet de las Cosas (*IoT*) constituye toda una red de objetos físicos o (cosas) los cuales llevan incorporados sensores de humedad, temperatura, nivel del agua, distancia, entre otras funcionalidades. Además, estos sensores permiten recolectar, compartir datos e interactuar y enviar información a plataformas informáticas en la nube en la red Internet [1].

Así mismo, el Internet de las cosas (*IoT*) puede ser considerada un paradigma en el que las cosas refiriéndonos a dispositivos se comunican entre sí y sobre todo con la ayuda de una plataforma en la nube. Esta interconexión permite la realización de diversas

tareas como son monitorear y controlar el dispositivo o el entorno circundante. En este contexto, los objetos o dispositivos se vuelven más inteligentes y colaborativos, facilitando la automatización de procesos y permitiendo nuevas oportunidades para mejorar la eficiencia y comodidad del usuario que interactúa con dispositivos *IoT* [2].

LoRa

Long Range (LoRa) es una tecnología de comunicación inalámbrica que facilita la transmisión de información a largas distancias patentada por Semtech. Además, es un tipo de modulación en radiofrecuencia y el tipo de tecnología de modulación que se emplea en la comunicación *LoRa* se denomina *Chirp Spread Spectrum (CSS)* [3].

Cabe mencionar que esta tecnología utiliza frecuencias no licenciadas (ISM), en el caso de Ecuador usa la frecuencia 915 (MHz) que es una de las menos saturadas en el país a comparación de la de 2.4 (GHz) [4].

Lo más relevante de esta tecnología es que combina dos factores muy importantes que es el alcance y el bajo consumo energético. *LoRa* permite una comunicación a larga distancia de hasta 5 y 10 (Km) con Línea de Vista (*LOS*) tomando en cuenta la zona de Fresnel. Además, como se mencionó anteriormente, esta tecnología tiene un bajo consumo energético que permite mantener dispositivos con baterías pequeñas y sin la necesidad de cargarlos de manera frecuente [5].

Heltec WiFi LoRa 32 (V3)

El dispositivo *Heltec WiFi LoRa 32* es una placa de desarrollo para *IoT* creada y fabricada por *Heltec Automation*. Lleva incorporado un microprocesador *ESP32-S3FN8 Extensa 32-bit LX7* de doble núcleo, estructura de *rack* de tuberías con cinco niveles, frecuencia de hasta 240 (MHz), chip de radio *SX1262* con *LoRa*. Además, en su última versión viene incorporado con una interfaz *USB* tipo *C* y cuenta con un regulador de voltaje completo, una protección *ESD*, protección frente a cortocircuitos, blindaje para *RF* y entre otras medidas de protección [6].

Además, lleva tres tecnologías de comunicación inalámbricas incorporadas como son *Wi-Fi*, *Bluetooth 5 (LE)* y *LoRa*. También, dispone de un *chip CP2102* a puerto serie integrado, conviene para bajar programas e imprimir datos de depuración y sobre todo apoyar al desarrollo *Arduino*. Con seguridad se puede decir que es la mejor opción para ciudades inteligentes, invernaderos, hogares, granjas, seguridad doméstica, lectura de medidores inalámbricos y desarrolladores *IoT* [6].

Sensor DHT22

El *DHT22 AM2302* es un sensor digital empleado para medir la temperatura y la humedad ambiental. Este sensor utiliza un termistor para medir el aire circundante y, además, integra un sensor capacitivo interno para medir la humedad y envía los datos a través de una señal digital en el pin de datos, recordar que este sensor no dispone de salida analógica. Así mismo, este sensor opera con voltaje de entre 3 (V) y 6 (V) y con un rango de temperatura de entre -40 ($^{\circ}$ C) a 80 ($^{\circ}$ C) [7].

Este sensor ofrece mejores características con comparación al sensor *DHT11* ofrece una mejor resolución y precisión en la medición tanto de la temperatura y humedad. Se usa en aplicaciones como son el control automático de temperatura, aire acondicionado y monitoreo ambiental en invernaderos [7].

System on a Chip (SoC)

Un System on a Chip (SoC), o sistema en un chip en español, se define en términos sencillos como un circuito integrado que contiene dentro de sí un sistema electrónico completo. En otras palabras, un SoC integra todos los componentes necesarios para que un sistema funcione en un solo chip [8].

Entre los componentes de un SoC se incluyen la *CPU*, puertos de entrada y salida, la memoria interna, y los bloques de entrada y salida tanto digitales como analógicos. Una de sus principales prestaciones es que proporciona una mayor densidad funcional, lo que significa que todos los componentes electrónicos están ubicados en un único chip [8].

Otra ventaja importante es la reducción de costos, ya que la integración de múltiples componentes en un solo chip disminuye los gastos de producción. Además, ofrece mayor flexibilidad al permitir que diversas tecnologías trabajen juntas en un mismo chip. Un SoC está compuesto por varios elementos, como la *BIOS*, *RAM*, *CPU*, chip de radio, *GPU*, periféricos, pantalla y *pines GPIO* [8].

Adafruit IO

Adafruit IO, es un servicio en la nube diseñado para alojar proyectos de *IoT*. Creada y gestionada por la conocida empresa *Adafruit Industries*, su base de datos es compatible con una gran variedad de *hardware*, como *Arduino*, *Raspberry pi*, *ESP32*, *ESP8266*, entre otros. Con esta plataforma se tiene la capacidad de añadir, guardar y observar datos en tiempo real desde dispositivos *IoT* conectados con esta plataforma. Además,

permite interactuar con estos dispositivos a través de paneles de control basados en la web. [9].

Una característica destacada de esta plataforma es su potente *API* con soporte para varios lenguajes de programación, también esta plataforma proporciona varias herramientas para el monitoreo, gestión y el control de datos. Además, ofrece gráficos, tablas, registros y otras herramientas permiten tener información en tiempo real de nuestros dispositivos *IoT* [9].

Protocolo *MQTT*

Message Queuing Telemetry Transport (MQTT), es el protocolo de mensajería más comúnmente utilizado para la *IOT* estandarizado por OASIS e ISO. Además, el protocolo *MQTT* es un conjunto de normas que especifica como los dispositivos *IoT* pueden publicar o suscribirse a datos a través de una red *WiFi*, está concebido como un medio transporte de mensajería muy ligero que permite enviar y recibir tramas de datos y también es óptimo para la conexión de dispositivos remotos con una huella de código reducida y un consumo mínimo de ancho de banda de red [10].

Actualmente, *MQTT* es utilizado por muchas empresas para conectar millones de dispositivos a Internet y se utiliza en muchas industrias, como en telecomunicaciones, automotriz, petróleo, textil, entre otras [10].

El protocolo *MQTT* se ejecuta sobre *TCP/IP* y utiliza una topología de tipo *Machine-to-Machine (M2M)* o de Publicación/Suscripción (*Publish/Subscribe*). Este protocolo está compuesto por tres elementos principales: el publicador, el *broker* y los suscriptores. En esta configuración, los dispositivos publicadores envían mensajes a un servidor central, conocido como *broker*, que luego los distribuye a los dispositivos suscriptores interesados. Esta arquitectura es eficiente para comunicaciones de baja latencia y alta eficiencia en redes con ancho de banda limitado [11].

Además, es importante mencionar que en el protocolo *MQTT*, los mensajes se publican en temas organizados jerárquicamente con barras (*/*) como delimitadores, similar a un árbol de directorios.

Por ejemplo, sensores/temperatura/externo y sensores/humedad/interior permiten a los suscriptores recibir datos específicos de temperatura exterior y humedad interior, respectivamente. Los temas no se crean explícitamente; si un *broker* recibe datos en un nuevo tema, este se crea automáticamente, facilitando la suscripción inmediata [11].

Arduino IDE

El entorno de desarrollo integrado (*IDE*) de Arduino es una plataforma de *software* de código abierto y gratis, diseñada para programar y desarrollar proyectos en el lenguaje de programación C++. Además, esta herramienta simplifica la escritura, compilación y carga de código principalmente en las placas Arduino, pero también es compatible con numerosas placas de terceros que soportan el lenguaje de programación Arduino [12].

También, permite a los usuarios crear y ejecutar una amplia variedad de proyectos electrónicos. El *IDE* soporta tanto a principiantes como a desarrolladores avanzados, ofreciendo características como autocompletado, depuración y soporte para la integración con Arduino Cloud [12].

Invernadero

Un invernadero es una estructura cerrada con una armazón metálica y una cubierta exterior de plástico diseñada para impedir la entrada de lluvia. Uno de sus principales objetivos es crear condiciones climáticas ideales para el cuidado, crecimiento y desarrollo de plantas [13]. Dentro del invernadero, se puede controlar un microclima específico a través de la regulación de la humedad y la temperatura, lo que permite realizar diversos cultivos en un entorno óptimo [14].

Humedad del suelo

La humedad del suelo indica la cantidad de agua que se encuentra en los poros del suelo o en su superficie, medida en porcentaje de volumen, o en pulgadas de agua por pie de suelo. Esta cantidad varía dependiendo del clima, el tipo de suelo y las especies vegetales presentes. Es fundamental en la agricultura debido a su impacto en la salud del suelo, la cantidad de agua disponible para las plantas, y su capacidad para predecir fenómenos climáticos como inundaciones o sequías [15].

Humedad ambiental

La humedad ambiental es una medida que indica la proporción de vapor de agua presente en el aire con una comparación con la cantidad máxima que puede albergar a una temperatura y presión específicas. Con el aumento de la temperatura, la capacidad para retener vapor de agua también procede a incrementar. Este término se usa comúnmente en los informes meteorológicos para describir la cantidad de humedad en el aire en relación con la cantidad máxima que podría albergar a esa temperatura. Además, el vapor de agua en la atmósfera se genera principalmente por la evaporación de grandes masas de agua, como océanos, lagos y mares, que son las principales fuentes de vapor de agua en el ciclo hidrológico terrestre [16].

2 METODOLOGÍA

Se realizó un análisis para identificar cuáles son los requisitos para llevar a cabo el diseño del prototipo para poder cumplir con el alcance establecido para una comunicación *LoRa*.

Tras identificar los requerimientos del prototipo, se analizó las características del *hardware* y *software* necesarias para permitir el control de invernadero a distancia. También, se llevó a cabo una investigación para identificar cuales plataformas *IoT* se utilizará para el desarrollo de una aplicación *web* que se encargue de monitorizar y controlar a distancia el invernadero por medio de un panel de control.

Una vez definido el *hardware* y *software* y a la aplicación *web* se procedió con la creación de los respectivos códigos en *Arduino IDE* que deben ser programados en el *SoC* para el control del invernadero. Así mismo, para el establecimiento de una comunicación inalámbrica mediante *LoRa*, finalmente se creó el panel de control *Web* en la plataforma *IoT* para visualizar los cambios de humedad y temperatura por medio de la interfaz *web*.

Luego, de haber creado los códigos, escogido la plataforma *IoT* y también de haber definido el *hardware* y *software* se implementó el sistema, en donde se procedió a integrar las partes previamente desarrolladas y por consiguiente se creó la maqueta o prototipo para así poder simular el despliegue del sistema de control de invernadero y verificar su funcionamiento correcto.

Finalmente, tras realizar la implementación del prototipo se llevaron a cabo pruebas para validar el funcionamiento correcto del prototipo. Estas pruebas serán cruciales y de suma importancia porque permitirán identificar cualquier error y corregir los errores si existen para asegurarse de que el prototipo de invernadero cumpla con los requerimientos y garantizar que sea funcional.

3 RESULTADOS

En esta sección, se presentará a detalle todos los requerimientos para realizar el prototipo con *SoC* para control de invernadero, que incluye la selección de *hardware* y *software* y además poner en funcionamiento el sensor y el sistema de irrigación y sobre todo que exista una comunicación con el *SoC* esclavo. Y a su vez que estos dispositivos se comuniquen mediante comunicación inalámbrica *LoRa* y enviar la información a la nube y visualizarlos en el *dashboards* de *Adafruit IO*.

3.1 Identificación de los requerimientos del prototipo

En base al análisis realizado anteriormente para la ejecución del prototipo se analizó todos los requerimientos que el prototipo necesita, el cual se presenta a continuación:

Dispositivos SoC

Para llevar a cabo este proyecto, se necesitan dos dispositivos SoC que realizarán la comunicación mediante *LoRa*. El primer SoC actuará como esclavo, encargado de leer los datos del sensor de temperatura y humedad, y de activar el sistema de irrigación.

El segundo SoC funcionará como maestro. Pues, su tarea principal será recibir los datos transmitidos por el esclavo a través de *LoRa*. También, este SoC se conectará a la red *WiFi* para enviar toda la información recopilada a una plataforma *IoT*, permitiendo el monitoreo y control remoto del sistema de irrigación.

Ambos SoC deben ser compatibles con la tecnología *LoRa* y *WiFi*, y tener la capacidad de integrarse con los sensores y actuadores necesarios para la correcta operación del sistema de irrigación inteligente.

Sistema de comunicación inalámbrica

El prototipo requiere de conectividad inalámbrica *WiFi* para permitir que el dispositivo maestro envíe la información recolectada a la nube. Para este propósito, se necesitará un SoC (*System on Chip*) que cuente con la tecnología *WiFi* según el estándar 802.11 b/g/n. Además de la conectividad *WiFi*, el SoC debe estar equipado con tecnología *LoRa*, lo que permitirá una comunicación de largo alcance y baja potencia. Esto asegurará que el sistema pueda transmitir datos incluso en áreas donde la cobertura *WiFi* es limitada.

Integrar ambas tecnologías en un solo SoC garantizará una conectividad robusta y versátil para el prototipo, permitiendo que los datos se envíen de manera eficiente a la nube para su análisis y control remoto del sistema de irrigación.

Dispositivos de monitoreo y control de irrigación

Para realizar el prototipo se va a necesitar de un dispositivo que permita medir la temperatura ambiental y la humedad del suelo. También, que permita enviar la información recolectada al SoC esclavo y que sea compatible con dispositivo. Además, para asegurar una irrigación eficaz en el prototipo de invernadero, se utilizará una mini bomba. Esta bomba permitirá llevar a cabo la irrigación de manera controlada. Para activar la mini bomba según las necesidades del sistema, se empleará un relé.

El relé actuará como un interruptor controlado electrónicamente por el SoC esclavo. Cuando sea necesario iniciar el proceso de irrigación, el relé recibirá una señal eléctrica del sistema de control, lo que activará la mini bomba. Esta acción permitirá que el agua se distribuya de manera adecuada en el invernadero. El uso del relé para activar la mini bomba garantiza una operación segura y precisa del sistema de irrigación. Así, se cumplen los requerimientos necesarios para tener las condiciones óptimas de humedad del suelo en el invernadero.

Banda de frecuencia

Es importante destacar que para establecer una comunicación *LoRa* se requiere de una banda de frecuencia Industrial, Científica y Médica (ISM) que no tenga costo alguno y también que no sea tan saturada para evitar interferencias. En este contexto, se necesita dispositivos que operen en las bandas de frecuencia ISM 863 – 928 (MHz). Por lo tanto, se verificó en la página de la Agencia de Regulación y Control de las Telecomunicaciones (ARCOTEL), encargada de asignar frecuencias ISM en Ecuador. Se decidió trabajar con la banda de 915 (MHz) para la comunicación *LoRa*, puesto que es una banda de frecuencia libre y una de las menos saturadas.

Plataforma de desarrollo de *software* libre y de código abierto

Para cumplir con los requerimientos del prototipo, es imprescindible contar con una plataforma de desarrollo de *software* libre y código abierto. Esta plataforma debe ser compatible con el SoC seleccionado, incluyendo las librerías necesarias para su programación. Además, debe integrarse sin problemas con la plataforma web utilizada para visualizar el monitoreo del sistema invernadero por medio de un panel de control.

La plataforma de desarrollo debe ofrecer un entorno de programación intuitivo y flexible, permitiendo la configuración y control del SoC de manera eficiente. También debe admitir la integración con sensores y actuadores del sistema de irrigación.

Código de programación para el SoC

Para programar el SoC, es necesario utilizar un código de programación en lenguaje C++. Esto es fundamental para interactuar adecuadamente con el *hardware* y llevar a cabo tareas específicas. Estas tareas incluyen la recepción de información de sensores de temperatura y humedad, el control del sistema de irrigación, así como la gestión de comunicaciones a través de *LoRa* y *WiFi*. Además, es esencial para la integración y comunicación con plataformas como *Adafruit IO* para el monitoreo y control remoto de los datos del invernadero por medio de un panel de control.

Plataforma de monitoreo *IoT*

Para este proyecto, se requiere una plataforma *IoT* en la nube que permita el control y monitoreo de los dispositivos *IoT*, asegurando su compatibilidad con los SoC seleccionados. Esta plataforma debe ofrecer un panel de control personalizable que permita visualizar los cambios en el prototipo de invernadero, así como integrar *feeds* de datos provenientes de sensores y dispositivos. Además, se necesita que la plataforma garantice la seguridad de la información transmitida y almacenada, protegiendo los datos del sistema de invernadero contra accesos no autorizados. La plataforma *IoT* en la nube será fundamental para gestionar el sistema de irrigación de manera eficiente, permitiendo la supervisión y control de manera eficiente.

Prototipo de invernadero

Para la realización de este proyecto, es esencial construir un prototipo de control para un invernadero. Este prototipo deberá simular el funcionamiento de un invernadero real, incluyendo la regulación de condiciones cruciales como la temperatura y la humedad. Además, se deberán incorporar materiales específicos tanto para el sistema de riego como para la construcción estructural del invernadero. Dado que se trata de un prototipo, es importante tener en cuenta que puede haber variaciones y ajustes en el diseño y funcionamiento durante su desarrollo.

Diseño y construcción de un circuito electrónico

Además, es fundamental construir un circuito electrónico en el cual se integren el SoC esclavo y por medio de *headers* se integren el sensor de humedad del suelo, sensor de temperatura ambiental y el relé. Es crucial que haya una comunicación eficiente y estable entre estos componentes. Por lo tanto, el diseño del circuito debe asegurar la seguridad y compatibilidad de todos los elementos involucrados. Un diseño correcto del circuito garantizará el excelente funcionamiento del prototipo de control del invernadero.

3.2 Selección del *hardware* y *software*

Selección del *hardware*

En el mercado existe una amplia gama de dispositivos inteligentes, como SoC y sensores, que me permiten desarrollar el prototipo para este proyecto. He analizado diversos de estos dispositivos que pueden ayudarme a desarrollar el proyecto y se realizó una comparativa entre varios elementos para seleccionar la mejor opción para su incorporación en el prototipo.

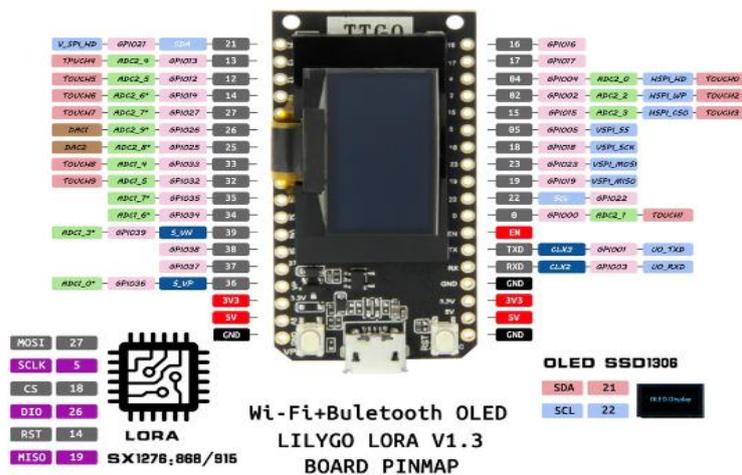


Tabla 3.1 Comparativa entre SoCs [17] [18] [6] [19]

Componente	<i>Heltec WiFi LoRa 32 (V3)</i>	LILYGO TTGO <i>LoRa V1.3</i>	chip SX 1276
<i>WiFi</i>	802.11 b/g/n, up to 150Mbps	802.11 b/g/n	No integrado
Voltaje (V)	3.3 a 7	2.7 a 3.6	1.8 a 3.7
<i>LoRa Chip</i>	<i>SX1262</i>	<i>SX1276</i>	<i>SX1276</i>
Temperatura (°C)	-20 a 70	-40 a 85	-55 a 115
<i>MCU</i>	<i>ESP32</i>	<i>ESP32</i>	NO integrado
Velocidad (MHz)	240	240	100
Dimensiones (mm)	50.2 x 25.25	53 x 25	65 x 53.3
Sensibilidad del receptor (dBm)	-136	-148	-148
<i>Versión Bluetooth</i>	<i>Bluetooth 5 (LE)</i>	<i>Bluetooth 4.2</i>	No integrado
Precio en dólares americanos	19.90	17.36	10.01

El análisis realizado en la Tabla 3.1 permitió identificar la opción más viable, determinando que el *Heltec WiFi LoRa 32 (V3)* es la mejor alternativa para llevar a cabo el proyecto. Este dispositivo cuenta con las características principales necesarias, como la tecnología *LoRa* incorporada, *WiFi* y un *microcontrolador* compatible con *Arduino IDE* para la compilación de códigos. Cabe mencionar que el dispositivo *LILY GO TTGO (V1.3)* también cumple con estas características; sin embargo, su disponibilidad en el país es limitada.

Determinación de la bomba de agua para el sistema de irrigación

Para llevar a cabo este proyecto y desarrollar el sistema de irrigación del prototipo, es necesario utilizar una bomba de agua, la cual facilitará la implementación del sistema de riego. En el mercado se encontraron dos tipos de bombas las cuales son la Mini Bomba de agua sumergible de 5 (V) y la Bomba de agua 12 (V) Sumergible *Brushless* 240 (L/H).

La mini bomba de agua sumergible de 5 (V) tiene la capacidad de trasladar líquidos desde un recipiente hacia otro destino deseado, lo que la hace adecuada para diversas aplicaciones de irrigación y transferencia de fluidos. Esta bomba no solo es eficiente en

el movimiento de líquidos, sino que también es compatible con microcontroladores, lo que permite su integración en sistemas automatizados. Sin embargo, para su correcta operación es necesario utilizar un relé o un transistor, los cuales actúan como intermediarios para manejar la corriente y proteger el microcontrolador [20]. A continuación, en la Figura 3.4 se puede visualizar a la mini bomba de agua.



Figura 3.4 Mini Bomba de agua 5 (V) [20]

En el mercado también se encuentra la bomba de agua sumergible Brushless de 12 (V) con una capacidad de 240 (L/H). Una de sus principales características es su impermeabilidad, lo que permite su uso constante en el agua después del sellado con resina. Además, esta bomba está diseñada para funcionar de manera continua durante 24 horas, y está fabricada con materiales resistentes y duraderos, lo que garantiza su fiabilidad en aplicaciones exigentes [21]. En la Figura 3.5 se ilustra a la bomba de agua 12 (V).



Figura 3.5 Bomba de agua 12 (V) sumergible [21]

A continuación, en la Tabla 3.2 se puede visualizar la comparativa entre las dos bombas de agua para así poder determinar la mejor opción para poder realizar el proyecto y poner en acción el sistema de irrigación.

Tabla 3.2 Comparación de bombas de agua [21] [20]

	Mini bomba de agua sumergible 5 (V)	Bomba de agua 12 (V) sumergible brushless 240 L/H
Voltaje (V)	2.5 a 6	12
Cantidad de flujo (L/H)	80 a 120	240
Temperatura de trabajo (°C)	-20 a 50	0 a 55
Precio en dólares americanos	3.99	10.99

A partir de la comparación realizada en la Tabla 3.2, se pudo constatar que la mejor opción para este proyecto es la mini bomba de agua sumergible de 5 (V). Esta bomba cumple con todas las características necesarias y, además, es más económica en comparación con la bomba de agua de 12 (V). Dado que se trata de un prototipo de sistema de irrigación, la mini bomba resulta muy útil y adecuada para las necesidades del proyecto.

Determinación del sensor de temperatura y humedad ambiental

El sensor *DHT22 AM2302*, es un sensor que me permite leer la humedad y temperatura ambiental en un mismo sensor. Este sensor mide la temperatura del aire circundante mediante un termistor y utiliza un sensor capacitivo interno para medir la humedad. La información se transmite a través de una señal digital en el pin de datos, puesto que no dispone de una salida analógica. Funciona con un voltaje de 3 a 6 (V) y puede operar en un rango de temperatura de -40 (°C) a 80 (°C) [7]. Además, este sensor es compatible con plataformas como *Arduino*, *Raspberry Pi* y *NodeMCU*, en cuanto al aspecto *hardware*, solo se requiere conectar el pin de alimentación VCC a un voltaje entre 3 y 5 (V), el pin GND a tierra que tiene un valor de 0 (V) y el pin de datos a un pin digital en el SoC [22]. Este sensor que se muestra en la Figura 3.6 es ideal para realizar este proyecto ya que cuenta con las características necesarias para el prototipo.

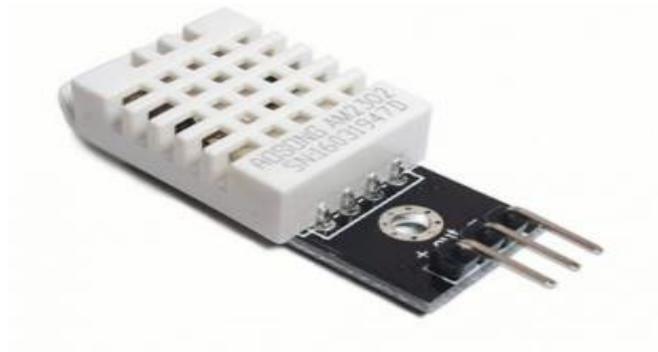


Figura 3.6 Sensor DHT22 AM2302 [7]

Determinación del sensor de humedad del suelo

El sensor seleccionado para este proyecto es el sensor de humedad del suelo *FC-28*, puesto que cumple con las características requeridas. Este sensor facilita la medición de la humedad del suelo de manera sencilla a través de electrodos resistivos. Su operación se fundamenta en medir la resistencia entre los electrodos colocados en el suelo. La resistencia cambia según la humedad de la tierra: en tierra muy húmeda, la resistencia es baja, mientras que, en tierra muy seca, la resistencia es alta [23].

El sensor *FC-28* está conectado a una tarjeta de acondicionamiento *YL-38*, la cual ofrece salidas digital y analógica. La salida analógica (AO) es generada por un divisor de tensión y varía desde 0 (V) en suelos muy húmedos hasta 5 (V) en suelos muy secos. Además, este sensor es compatible con microcontroladores como *Arduino*, *PIC*, *ESP8266*, *ESP32* y el *SoC Heltec WiFi LoRa 32 (V3)* puesto que este se basa en un chip *ESP32* [23]. En la Figura 3.7 se ilustra al sensor de humedad del suelo.

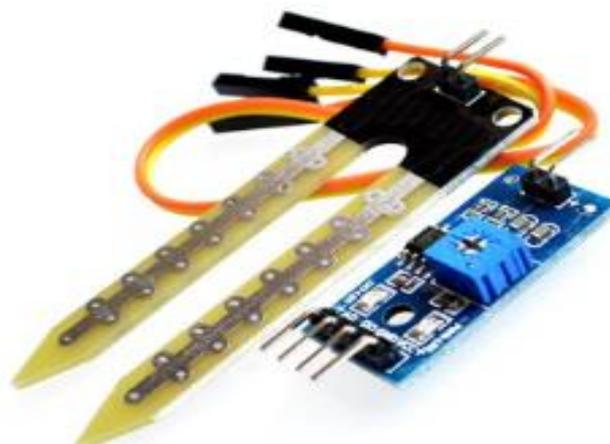


Figura 3.7 Sensor de humedad del suelo FC-28 [23]

Determinación de una fuente de alimentación para los SoCs

El adaptador de corriente USB C de 10 (W) de LJO-EEIH que se aprecia en la Figura 3.8 ofrece una solución eficiente para la carga de varios dispositivos, incluidos auriculares, así como altavoces Bluetooth. También es compatible con dispositivos como el *Heltec WiFi LoRa 32 (V3)* y *ESP32*, proporcionando una entrada universal de 100-240 (V) y una salida de 5 (V) con hasta 2.5 (A). Este adaptador garantiza una carga rápida y segura gracias a sus características de protección contra sobretensión y cortocircuitos, además de incluir un cable de carga USB tipo C de 5 pies de largo para mayor comodidad y versatilidad [24]. Es importante mencionar que se determinó dos unidades de esta.



Figura 3.8 Fuente de alimentación tipo C LJO-EEIH [24]

Determinación del relé para activar la bomba de agua

El relé seleccionado para este proyecto es el *LAOGOG JQC-3FF-S-Z* que se aprecia en Figura 3.9 es un módulo de 5 (V) de un canal, ideal para controlar dispositivos eléctricos como una bomba en un sistema de irrigación. Funciona con un disparo de nivel bajo, activándose con una señal baja desde un microcontrolador. Es capaz de manejar cargas de hasta 10 (A) a 250 (VAC) o 15 (A) a 125 (VAC), proporcionando una interfaz simple de 3 pines *VCC*, *GND* y señal de entrada para una fácil integración. Este relé es perfecto para el sistema de irrigación requerido para este proyecto, permitiendo controlar la bomba de agua de manera eficiente y segura [25].



Figura 3.9 Relé LAOGOG JQC-3FF-S-Z [25]

Determinación del software

Para realizar este proyecto, es fundamental seleccionar un *software* adecuado que permita la compilación del código necesario para poner en funcionamiento el módulo SoC. Entre las opciones disponibles se encuentran *Arduino IoT Cloud* y *Arduino IDE*. Además, es esencial contar con un *software* que facilite el monitoreo de los datos enviados por el sensor de humedad y temperatura, así como el control del sistema de irrigación, todo a través de un panel de control intuitivo y eficiente.

Selección del software para el control del SoC

Arduino IDE es un entorno de desarrollo integrado de código abierto diseñado para la creación de proyectos electrónicos. Este programa informático, que requiere su instalación en un ordenador, se basa en *hardware* y *software* libre, ofreciendo una plataforma accesible y fácil de usar para los desarrolladores. Su interfaz intuitiva y su amplia compatibilidad con diversas placas y módulos lo convierten en una herramienta esencial para el desarrollo y la implementación de proyectos electrónicos [12].

Arduino IoT Cloud es también una plataforma de desarrollo que simplifica la programación y configuración de dispositivos *IoT*, permitiendo a los desarrolladores poner en funcionamiento un dispositivo en pocos minutos. Para utilizar este *software*, solo se necesita un navegador web y una cuenta en la plataforma. Además, proporciona varias técnicas de interacción, tales como *API REST*, *HTTP*, *MQTT*, instrumentos de línea de comandos, *Javascript* y *WebSockets*. Esta plataforma requiere el uso de las placas de la familia MKR, las cuales facilitan la creación de nodos *IoT* y dispositivos avanzados, proporcionando múltiples opciones de conectividad y compatibilidad con dispositivos de *hardware* de terceros y de varios sistemas en la nube [26].

En la siguiente Tabla 3.3 se visualiza las características principales de las dos herramientas de *software* para escoger la mejor opción para desarrollar este proyecto.

Tabla 3.3 Comparativa entre Arduino IoT Cloud y Arduino IDE [26] [12] [27]

Característica	Arduino IoT Cloud	Arduino IDE
Soporte para múltiples bibliotecas y extensiones	No	Si
Compatibilidad con Heltec WiFi LoRa 32 (V3)	No	Si
Monitor Serial integrado	No	Si

Característica	Arduino IoT Cloud	Arduino IDE
Código abierto	Si	Si
Soporte para dispositivos IoT	Si	Si
Amplia compatibilidad de placas	No	Si

Luego de haber realizado una comparativa detallada entre *Arduino IoT Cloud* y *Arduino IDE* en la Tabla 3.3. Se puede concluir que la alternativa óptima para el desarrollo de este proyecto es *Arduino IDE*. Esto ocurre debido a que *Arduino IDE* ofrece una mayor compatibilidad con el dispositivo *Heltec WiFi LoRa 32 (V3)*, el cual será utilizado en este proyecto. Es importante mencionar que, aunque *Arduino IoT Cloud* ofrece beneficios como la compilación y subida de código en la nube y un *dashboard* intuitivo para monitoreo y control, no es compatible con el SoC que se va a usar en este proyecto.

Arduino IDE no solo permite una instalación local y acceso sin necesidad de internet, sino que también proporciona una amplia compatibilidad con una variedad de placas, incluida la *Heltec WiFi LoRa 32 (V3)*. Además, facilita la programación y depuración a través de su interfaz clásica, ofreciendo herramientas avanzadas como el Monitor Serial para la visualización de datos en tiempo real.

Selección del software para el monitoreo web del prototipo

Hoy en día, hay múltiples plataformas que permiten conectar sistemas *IoT*, recoger datos de dispositivos conectados, almacenarlos y analizarlos en tiempo real. Entre una de las principales se tiene a *Adafruit IO*, desarrollada por *Adafruit Industries*. Este *software* permite agregar, almacenar y visualizar datos en la nube desde dispositivos *IoT*, y facilita la interacción a través de paneles de control *web* o llamados *dashboard*. Es compatible con varios dispositivos como *Arduino*, *Raspberry Pi* y *ESP8266*, y trabaja con *APIs*, *REST* y *MQTT*. La versión gratuita permite 30 datos por minuto, almacenamiento por 30 días, alertas cada minuto, y hasta 5 paneles con 10 campos cada uno [28].

Además, también se tiene otra opción que es *Blynk IoT*, que es una plataforma diseñada para facilitar la conexión y gestión de dispositivos *IoT*. Permite a los usuarios desarrollar aplicaciones móviles para controlar sus dispositivos conectados de manera sencilla y eficiente. Con *Blynk*, los usuarios pueden crear interfaces personalizadas mediante un sistema de *widgets*, sin necesidad de conocimientos avanzados en programación. Además, la plataforma es compatible con múltiples tipos de *hardware*, incluyendo *Arduino*, *Raspberry Pi* y *ESP8266*, lo que la hace extremadamente versátil para diversos

proyectos *IoT*. *Blynk* también soporta protocolos de comunicación como *MQTT* y *HTTP*, proporcionando una infraestructura confiable para el manejo de datos y control [29].

En la siguiente Tabla 3.4 se aprecia las características de las dos plataformas *IoT* para escoger la mejor opción para desarrollar este proyecto:

Tabla 3.4 Comparativa entre *Adafruit IO* y *Blynk IOT* [9] [28] [29]

Característica	<i>Adafruit IO</i>	<i>Blynk IoT</i>
Interfaz intuitiva	Si	Si
Compatibilidad con <i>Arduino IDE</i>	Si	Si
Plan gratuito con suficientes funciones	Si	Si
Envío de alertas	Si	No
Plan de pago accesible	Si	No
Personalización con un <i>dashboard</i>	Si	Si

Luego de realizar la comparativa en la Tabla 3.4 entre ambas plataformas se concluye que *Adafruit IO* es la opción óptima para este proyecto. Esta plataforma cumple con los requisitos indispensables, como ser compatible con *Arduino IDE*, lo cual es fundamental para la integración con nuestros dispositivos. Además, *Adafruit IO* permite la creación de *feeds* y la configuración de *dashboards* para monitorear los cambios de temperatura ambiental y humedad del suelo, así como para implementar un sistema de irrigación para el prototipo.

Una característica destacada de *Adafruit IO* es su integración para configurar alertas en su plataforma, lo que posibilita enviar alertas a través de correo electrónico basadas en los datos recolectados en los *feeds*. Esto proporciona una solución eficaz para la gestión y notificación ante condiciones críticas o eventos específicos dentro del prototipo.

Selección del *software* para enviar alertas al correo electrónico

El *software* seleccionado para enviar alertas por correo electrónico es *actions* en *Adafruit IO*, que permite una integración eficiente con los *feeds*. Esta configuración automatiza el envío de notificaciones, garantizando que las alertas se envíen automáticamente cuando se detectan eventos o condiciones anormales en los datos recibidos desde los sensores de temperatura ambiental, humedad ambiental y humedad del suelo conectados a *Adafruit IO* [30].

3.3 Diseño del prototipo para control de invernadero

Diseño esquemático general del proyecto

El diseño general de este proyecto para un prototipo de control de invernadero mediante comunicación *LoRa* consta de varios elementos que interactúan entre sí. El núcleo de este sistema son los dispositivos *SoC Heltec WiFi LoRa 32 (V3)*, que se comunican de manera bidireccional, enviándose información mutuamente. Además, el sistema integra sensores de humedad de suelo y temperatura ambiental, así como un sistema de irrigación.

En la sección de *software*, la información recopilada por los sensores es transmitida a la plataforma *Adafruit IO*, donde se almacena en *feeds* y se visualiza a través de un *dashboard*. Adicionalmente, mediante *Adafruit* y la herramienta de *actions* de esta, se pueden enviar alertas por correo electrónico, facilitando así el monitoreo y control del invernadero. En la Figura 3.10 se puede visualizar el esquema.



Figura 3.10 Esquema de prototipo de invernadero con comunicación *LoRa*

En la Figura 3.11 se ilustra de manera detallada todos los elementos del prototipo de invernadero con comunicación *LoRa*. Los elementos principales del sistema son dos dispositivos *SoC Heltec WiFi LoRa 32 (v3)*, que se comunican entre sí mediante una comunicación inalámbrica *LoRa*, permitiendo un intercambio bidireccional de información.

El *SoC* denominado esclavo se comunica con el sensor *DHT22* y sensor de humedad del suelo, captando y enviando los datos de las variaciones al *SoC* maestro a través de *LoRa*. Además, el *SoC* esclavo tiene la función de activar o desactivar un relé que controla la mini bomba de agua para el sistema de irrigación.

Por otro lado, el *SoC* denominado maestro recibe la información enviada por el *SoC* esclavo sobre los cambios de temperatura ambiental y humedad del suelo mediante *LoRa*. Este *SoC* maestro, a su vez, se conecta a una red *WiFi* para transmitir toda la

información recibida del SoC esclavo a la plataforma *Adafruit IO* mediante el protocolo *MQTT*. Adicionalmente, el SoC maestro recibe información de *Adafruit IO* relacionada con el estado del *relé* y la envía a través de *LoRa* al SoC esclavo, completando así el ciclo de control y monitoreo del invernadero. Finalmente, el usuario puede interactuar en el panel de control y monitorear de manera remota los parámetros ambientales en el prototipo, así como controlar el sistema de irrigación y además en el correo electrónico verificar las alertas cuando existan.

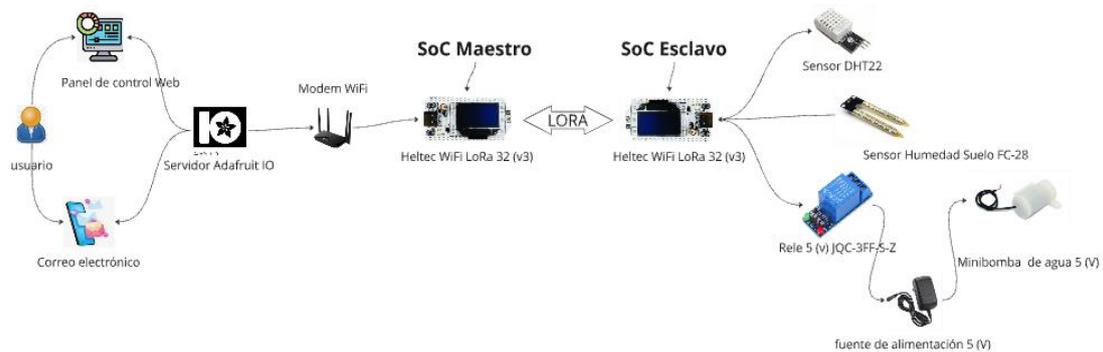


Figura 3.11 Elementos de todo el sistema

Selección de plataforma para la programación

La plataforma elegida para la programación y la interacción con la plataforma web es *Arduino IDE* que se aprecia en la Figura 3.12, ya que es compatible con los dispositivos *Heltec WiFi LoRa 32 (V3)*. A través de esta plataforma y la integración de diversas librerías, es posible enviar información al servidor *Adafruit IO* [12]. Además, con el uso de librerías facilita la integración de sensores como el *DHT22*, que es totalmente compatible con *Arduino IDE*.

A continuación, se va a presentar características sobre esta plataforma:

- *Arduino IDE*, es un *software* libre y de código abierto compatible con varios sistemas operativos como: *Windows*, *macOS* y *Linux*. [31].
- La interfaz gráfica de *Arduino* es intuitiva para el usuario y fácil de usar. [31].
- Utiliza en lenguaje de programación *Arduino C++*, *Python* y *JavaScript*.
- Incluye un monitor serial que permite verificar en tiempo real la depuración e integración del *SoC*.
- Este *software* incluye varias librerías compatibles con el *SoC Heltec WiFi LoRa 32 (V3)*, lo que facilita y simplifica la creación de los códigos necesarios.

- Además, esta plataforma cuenta con actualizaciones regulares y proporciona un soporte continuo.
- Esta plataforma dispone de un gestor de bibliotecas, lo que le permite al usuario buscar e instalar más recientes y actualizadas [31].

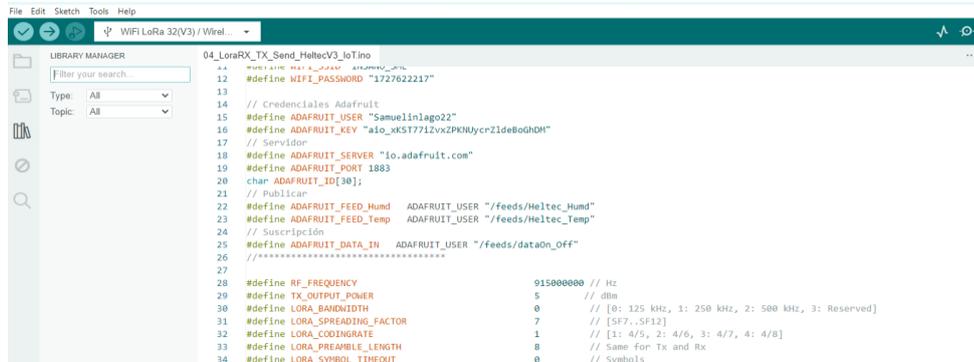


Figura 3.12 Plataforma de Arduino IDE

Creación de códigos en *Arduino IDE*

Previo a utilizar la plataforma *Arduino IDE* el primer paso es crear una cuenta en su página *web* y descargar el *software* en nuestro ordenador. Es importante mencionar que esta plataforma es totalmente gratuita y de código abierto. Además, en esta sección se explicará cómo se creó los códigos necesarios para programar los dispositivos *SoC* tanto esclavo como maestro y cómo integrarlos con *Arduino IDE* y *Adafruit IO*. Además, es importante destacar que se desarrollarán dos códigos diferentes para cada *SoC*.

Integración de los *SoC* con *Arduino IDE*

El primer paso para utilizar los dispositivos y poder programarlos en *Arduino IDE* es ir a la página de *Heltec Automation* donde se sigue una serie de pasos para la instalación de la biblioteca y el marco de desarrollo de la serie *Heltec ESP32* [32]. A continuación, se explicará todos los pasos para la instalación:

- El primer paso es abrir *Arduino IDE*, y presionar click en archivo y luego en preferencias, como se visualiza en la Figura 3.13.

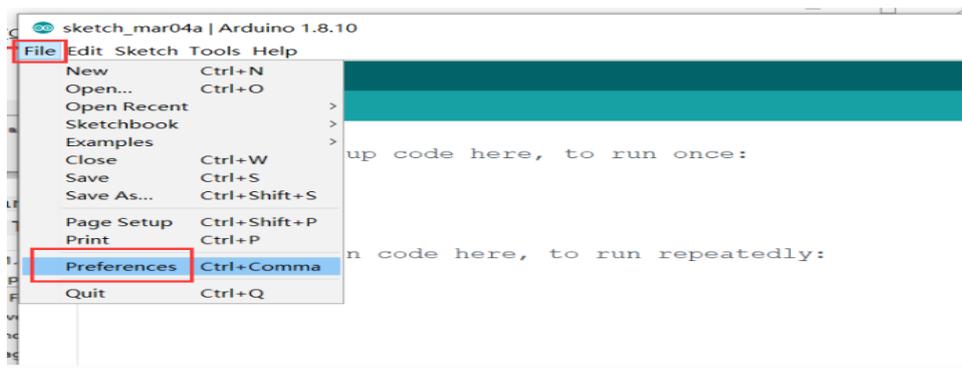
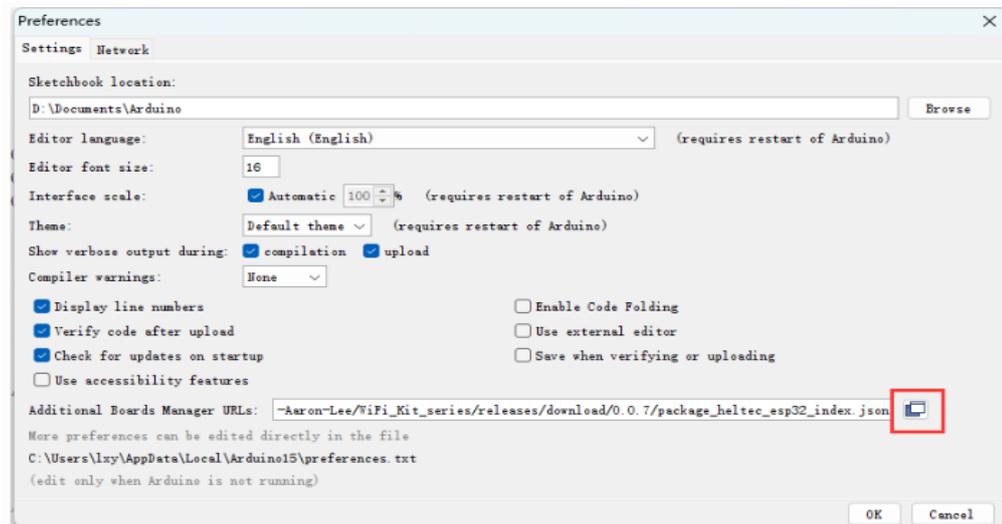


Figura 3.13 Guía de instalación de la librería Heltec ESP32 [33]

- El siguiente paso es pegar la *URL* del paquete *ESP32* en la sección *URLs* adicionales del gestor de placas como se ilustra en la Figura 3.14. A continuación, se debe cerrar *Arduino IDE* y reiniciarlo.



2. Input the last ESP32 package URL:

https://resource.heltec.cn/download/package_heltec_esp32_index.json

Figura 3.14 Guía de instalación de la librería Heltec ESP32 [28]

- Por último, se accede a herramientas y luego a gestor de placas se procede a buscar “*Heltec ESP32*” en el cuadro de búsqueda, y se selecciona la última versión como se ilustra en la Figura 3.15.

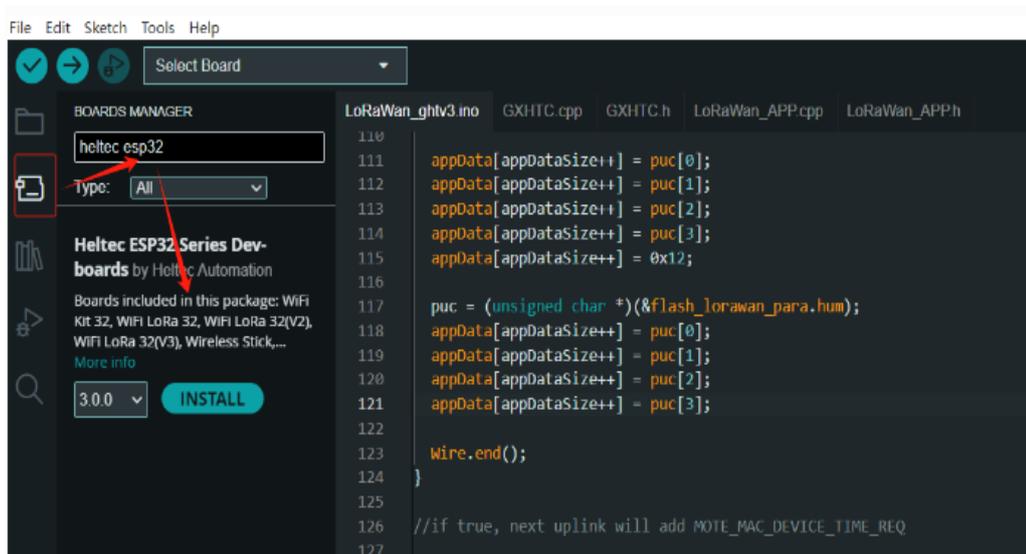


Figura 3.15 Guía de instalación de la librería Heltec ESP32 [28]

Después de haber instalado la librería siguiendo los pasos mencionados anteriormente, se visualiza en la Figura 3.16 que el *Heltec WiFi LoRa 32 (V3)* ya está disponible y se procede a seleccionar esta opción para trabajar con este *microcontrolador*.

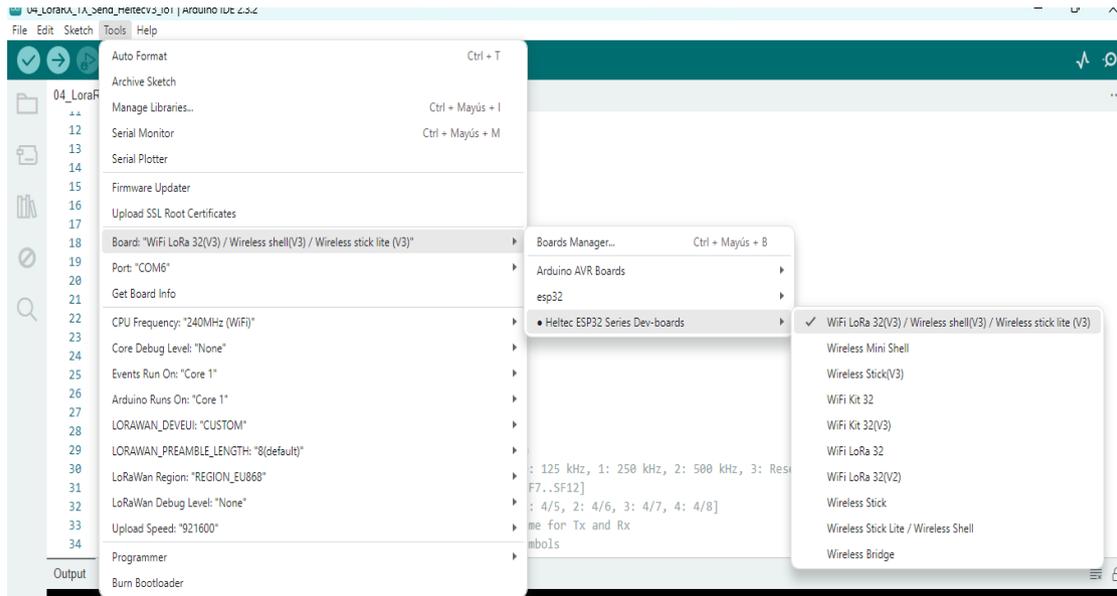


Figura 3.16 Verificación de la correcta instalación de la librería Heltec ESP32

Además, se procede a pasar a la sección de ejemplos, como se muestra en la Figura 3.17, para facilitar la programación a base de los ejemplos ya existentes en esta plataforma de programación.

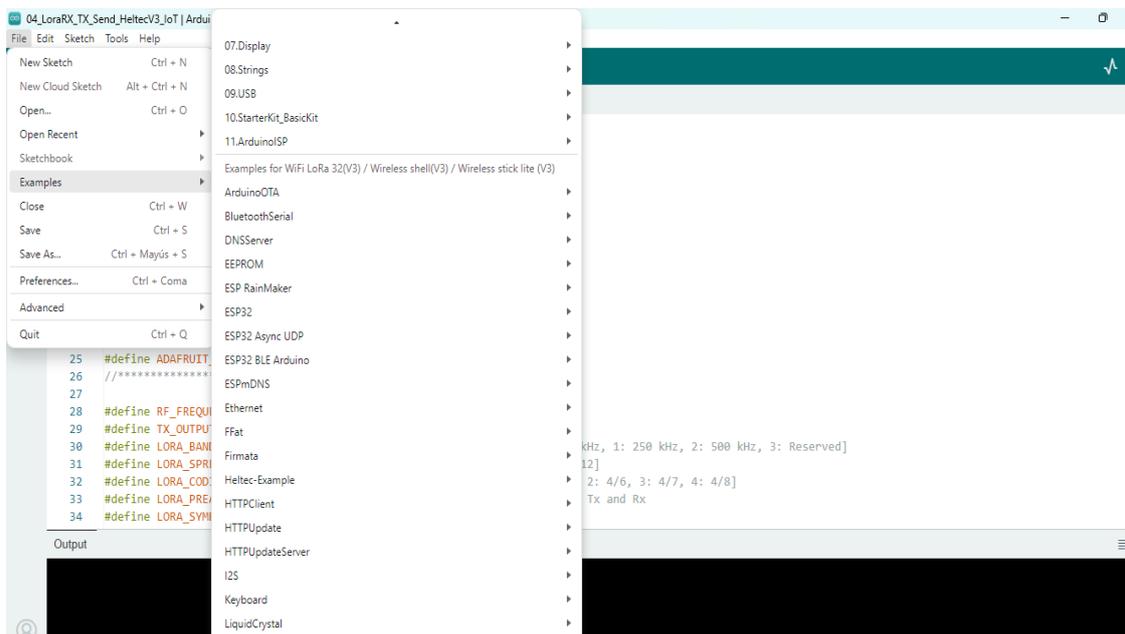


Figura 3.17 Verificación de los ejemplos en *Arduino IDE*

Librerías para el SoC esclavo

Un aspecto fundamental previo desarrollar los códigos para los microcontroladores es la selección de las librerías adecuadas. Estas librerías son esenciales, ya que proporcionan funciones y definiciones que simplifican la programación y el manejo del *hardware*.

En la Figura 3.18 se especifica la funcionalidad de cada librería en el SoC esclavo. El uso de librerías adecuadas proporciona un excelente funcionamiento de todos los elementos que conforman el prototipo para el proyecto a realizarse.

```
#include "LoRaWan_APP.h" //Librería para habilitar la comunicación LoRa y su respectiva transmisión
#include "Arduino.h" //Librería base de Arduino, configuración de pines, temporización, manejo de interrupciones.
#include <Adafruit_Sensor.h> // Esta librería define una interfaz estándar para todos los sensores de Adafruit
#include "DHT.h" //Librería para el manejo del sensor DHT22
```

Figura 3.18 Librerías del SoC esclavo

Configuración de sensores y pines de salida en el SoC esclavo

En esta sección del código del SoC esclavo como se muestra en la Figura 3.19 se establece los pines de salida que van conectados a los sensores y relé. Además, la configuración del sensor DHT se realiza asignando el pin *DHTPIN* al número 4 y especificando el tipo de sensor como *DHT22*.

Además, se asigna las salidas al pin número 6 para el sensor de humedad del suelo y el 7 para controlar el relé. Esta configuración asegura que el sensor esté correctamente inicializado para medir la temperatura y la humedad ambiental.

```
#define DHTPIN 4 // Se define el pin al que esta conectado el sensor DHT22
#define DHTTYPE DHT22 // Se define el tipo de sensor DHT que se está utilizando en este caso es un DHT22
DHT dht(DHTPIN, DHTTYPE); //Se crea un objeto DHT llamado dht utilizando el pin definido en DHTPIN y el tipo de sensor DHTTYPE

#define PIN_SALIDA_1 7 //Se define el pin de salida 1 como el pin 7 utilizado para controlar el relé
#define Hum_Suelo 6 //Se define el pin Hum_Suelo como el pin 6 utilizado para el sensor de humedad del suelo
```

Figura 3.19 Configuración de pines y sensores en el SoC esclavo

Configuración de parámetros de comunicación LoRa en el SoC esclavo

Un aspecto crucial en el desarrollo de los códigos para los dispositivos SoC maestro y esclavo es la definición de los parámetros de comunicación *LoRa*. Estos parámetros permiten establecer una conexión bidireccional entre los dos *microcontroladores*, facilitando el intercambio de información de manera eficiente y confiable a través de la tecnología *LoRa*. Esta configuración es fundamental para asegurar una comunicación efectiva y continua entre los dispositivos. A continuación, en la figura Figura 3.20 se puede observar que están configurados parámetros como frecuencia de operación de (Hz), Potencia de transmisión en (*dBm*), entre otros parámetros para la comunicación *LoRa* en el SoC esclavo.

```

#define RF_FREQUENCY          915000000 //Define la frecuencia de radio en la que opera el dispositivo LoRa, establecida en 915 MHz
#define TX_OUTPUT_POWER      5          // Establece la potencia de salida para las transmisiones LoRa en 5 dBm, que es la potencia nominal
#define LORA_BANDWIDTH       0          // Define el ancho de banda utilizado para las comunicaciones LoRa. Aquí, 0 representa un ancho de l
#define LORA_SPREADING_FACTOR 7          // Especifica el factor de expansión para las transmisiones LoRa. Un valor de 7 indica un factor de l
#define LORA_CODINGRATE      1          //Configura la tasa de codificación utilizada en las transmisiones LoRa. Aquí, 1 representa una tasa
#define LORA_PREAMBLE_LENGTH 8          //Define la longitud del preámbulo utilizado en las transmisiones LoRa, especificado en 8 símbolos.
#define LORA_SYMBOL_TIMEOUT  0          //Establece el tiempo de espera de símbolos para las comunicaciones LoRa. En este caso, se configura
#define LORA_FIX_LENGTH_PAYLOAD_ON false // Habilita o deshabilita el uso de longitud fija de carga útil en las transmisiones LoRa. Aquí, está
#define LORA_IQ_INVERSION_ON false      // Controla la inversión de la modulación IQ en las transmisiones LoRa. En este caso, está configurad

#define RX_TIMEOUT_VALUE     1000      //Define el tiempo de espera para la recepción en milisegundos. Aquí se establece en 1000 ms
#define BUFFER_SIZE          30        // Especifica el tamaño del búfer para almacenar datos, configurado en 30 bytes para esta aplicación.

```

Figura 3.20 Parámetros *LoRa* en el SoC esclavo

Declaración de variables globales para la comunicación *LoRa* y sensores en el SoC esclavo

Es imprescindible la declaración de variables para realizar el código. En la Figura 3.21 se puede apreciar en las primeras dos líneas declaran *arrays* de caracteres, *txpacket* y *rxpacket* con un tamaño definido por *BUFFER_SIZE* definido anteriormente. Estos *arrays* se utilizan para almacenar los datos que se transmitirán y recibirán a través de la comunicación *LoRa*.

La línea *static RadioEvents_t RadioEvents* declara una estructura estática llamada *RadioEvents*. Esta estructura es utilizada para gestionar los diferentes eventos del radio, como la transmisión y recepción de datos. Al ser estática, la variable *RadioEvents* tendrá un alcance limitado al archivo en el que está declarada y su duración será durante toda la ejecución del programa. Además, se declaran varias variables globales en esta sección de líneas de código:

- *txNumber* para llevar un conteo del número de transmisiones realizadas.
- *rssi* para almacenar la fuerza de la señal recibida (indicada en dBm).
- *rxSize* para almacenar el tamaño del paquete recibido.
- *LoRa_idle* es una variable booleana que indica si el módulo *LoRa* está en estado de inactividad *true* o activo *false*.

También se tiene, las variables *StrDataResp* y *StrDataIn* son cadenas de tipo *String* utilizadas para almacenar los datos de respuesta que se envían y los datos de entrada que se reciben. Finalmente se tiene, *ValHum* y *ValTem* que son variables de punto flotante para almacenar los valores de humedad y temperatura leídos del sensor *DHT22*. Inicialmente, ambos valores se establecen en 0.

```

char txpacket[BUFFER_SIZE]; // Define un array de caracteres para almacenar los datos a transmitir
char rxpacket[BUFFER_SIZE]; // Define un array de caracteres para almacenar los datos recibidos

static RadioEvents_t RadioEvents; // Define una estructura de eventos de radio estática

int16_t txNumber; // Variable para almacenar el número de transmisiones
int16_t rssi, rxSize; // Variables para almacenar la fuerza de la señal recibida y el tamaño del paquete recibido
bool lora_idle = true; // Variable para indicar el estado de inactividad de LoRa

String StrDataResp, StrDataIn; // Variables para almacenar las cadenas de datos de respuesta y de entrada
float ValHum,ValTem = 0; // Variables para almacenar los valores de humedad y temperatura

```

Figura 3.21 Declaración de variables SoC esclavo

Declaración de funciones de eventos de comunicación LoRa en el SoC esclavo

Estas líneas de código en la Figura 3.22 se declaran tres funciones que manejan eventos en la comunicación LoRa. *OnTxDone* se invoca cuando una transmisión se completa exitosamente, *OnTxTimeout* se llama cuando una transmisión no se finaliza dentro del tiempo límite establecido, y *OnRxDone* se ejecuta al recibir datos, proporcionando el *payload* recibido, su tamaño, la fuerza de la señal *RSSI* y la relación señal a ruido SNR. Estas funciones permiten al sistema responder adecuadamente a eventos de transmisión y recepción, gestionando la comunicación LoRa eficiente.

```

void OnTxDone(void); // Función que se llama cuando la transmisión se completa
void OnTxTimeout(void); // Función que se llama cuando la transmisión se agota el tiempo de espera
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr); // Función que se llama cuando la recepción se completa

```

Figura 3.22 Declaración de funciones para la comunicación LoRa

Función void setup en el SoC esclavo

En la función *setup ()* que se aprecia en la Figura 3.23 se realizó las configuraciones iniciales necesarias para preparar el entorno de ejecución del programa. En primer lugar, se establece el modo de los pines utilizados por el sistema. En este caso, el pin digital 7 en *PIN_SALIDA_1* se configura como salida mediante *pinMode (PIN_SALIDA_1, OUTPUT)*, y se asegura de que esté en estado cero con *digitalWrite(PIN_SALIDA_1, LOW)*, lo que asegura que cualquier dispositivo conectado a este pin comience en un estado de apagado.

Posteriormente, se inicializan la comunicación serial a una velocidad de 115200 *baudios* con *Serial. Begin (115200)*. Esto permite la comunicación con el puerto *serial* en *Arduino*, útil para mostrar información durante la ejecución del programa. Luego, se realizó la inicialización del SoC con *Mcu.begin()*.

Además, se inicializó variables importantes para el funcionamiento del programa, como *txNumber* y *rssí*, que se utilizan para llevar un conteo de los paquetes transmitidos y para almacenar la intensidad de la señal recibida, respectivamente.

Luego, se configura la comunicación *LoRa* con parámetros específicos como la frecuencia *RF_FREQUENC*), potencia de salida *TX_OUTPUT_POWER*, y configuraciones de modulación *LORA_BANDWIDTH*, *LORA_SPREADING_FACTOR*, etc. mediante llamadas a funciones de la biblioteca de radio utilizada en el proyecto. Finalmente se inicializó al sensor *DHT22* con *dht.begin()*, estas configuraciones son fundamentales para establecer la comunicación *LoRa* adecuada.

```
void setup() {  
  
    pinMode(PIN_SALIDA_1, OUTPUT); // Configura el pin de salida 1 como salida  
    digitalWrite(PIN_SALIDA_1, LOW); // Establece el pin de salida 1 en 0  
  
    Serial.begin(115200); // Inicializa la comunicación serial a 115200 baudios  
    Mcu.begin(); // Inicializa el MCU o microcontrolador  
  
    txNumber = 0; // Inicializa el número de transmisiones en 0  
    rssi = 0; // Inicializa la fuerza de la señal recibida en 0  
    // Asigna las funciones de eventos a la estructura de eventos de radio  
    RadioEvents.RxDone = OnRxDone; // Asigna la función OnRxDone al evento RxDone  
    RadioEvents.TxDone = OnTxDone; // Asigna la función OnTxDone al evento TxDone  
    RadioEvents.TxTimeout = OnTxTimeout; // Asigna la función OnTxTimeout al evento TxTimeout  
    Radio.Init(&RadioEvents); // Inicializa la comunicación LoRa con la estructura de eventos  
    Radio.SetChannel(RF_FREQUENCY); // Establece el canal de radio a la frecuencia definida  
    Radio.SetRxConfig(MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,  
                      LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,  
                      LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,  
                      0, true, 0, 0, LORA_IQ_INVERSION_ON, true); // Configura los parámetros de recepción del radio  
    dht.begin(); //Inicializa el sensor DHT22  
}
```

Figura 3.23 Función *Setup* en el SoC esclavo

Función *void loop* en el SoC esclavo

En general, la función *void loop* que se aprecia en la Figura 3.24 se encarga de mantener la operación continua del programa asegurando que la comunicación *LoRa* esté siempre lista para recibir datos. Entonces, en esta función se controla el flujo principal de ejecución del programa, verificando si la radio *LoRa* está en estado inactivo *LoRa_idle*. Si está inactiva, se cambia a estado activo *LoRa_idle = false*, se imprime un mensaje indicando que la radio está entrando en modo de recepción ("*into RX mode*"), y se pone la radio en modo de recepción *Radio.Rx(0)*. La función *Radio.IrqProcess()* se llama continuamente para procesar cualquier interrupción en la comunicación *LoRa*, asegurando que los eventos de recepción y transmisión de datos se manejen adecuadamente. Este ciclo se repite indefinidamente, permitiendo que el programa responda a las transmisiones y recepciones de datos de manera continua.

```

void loop() {
  if (lora_idle) { // Si lora_idle es verdadero
    lora_idle = false; // Establece lora_idle en falso
    Serial.println("into RX mode"); // Imprime en la consola "into RX mode"
    Radio.Rx(0); // Pone la radio en modo RX (recepción)
  }
  Radio.IrqProcess(); // Procesa las interrupciones de la radio
}

```

Figura 3.24 Función *void loop*

Función *void OnRxDone* en el SoC esclavo

La función *void OnRxDone ()* que se ilustra en la Figura 3.25 es un manejador de eventos que se invoca cuando el SoC recibe correctamente un paquete de datos enviados a través de comunicación *LoRa* del SoC maestro. Dentro de esta función, se actualizan varias variables globales para almacenar información sobre la señal recibida, incluyendo *rssI* que es el indicador de la intensidad de la señal recibida y *rxSize* que es el tamaño del paquete recibido. El contenido del paquete recibido se copia al *buffer rxpacket*, y se agrega un carácter nulo `\0` al final para asegurarse de que el *buffer* se trate como una cadena de caracteres terminada en nulo. Luego, radio *LoRa* se pone en modo de reposo para ahorrar energía hasta que sea necesario recibir o transmitir nuevamente.

Después de procesar el paquete recibido, la función convierte el contenido del *buffer rxpacket* a una cadena de caracteres *StrDataIn* y la imprime en el monitor serial. Además, dependiendo del contenido de esta cadena, se realizan acciones específicas como encender o apagar el rele al pin *PIN_SALIDA_1*

```

void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr) {
  rssi = rssi; // se asigna rssi a sí mismo sin necesidad de cambio
  rxSize = size; // Asigna size a rxSize
  memcpy(rxpacket, payload, size); // Copia el payload recibido a rxpacket
  rxpacket[size] = '\0'; // Termina rxpacket con el carácter nulo '\0'
  Radio.Sleep(); // Pone la radio en modo Sleep
  StrDataIn = String(rxpacket); // Convierte rxpacket a un objeto String StrDataIn
  Serial.print("StrDataIn: "); // Imprime "StrDataIn: "
  Serial.println(StrDataIn); // Imprime StrDataIn en la consola

  // Verificar si el mensaje es "ON" o "OFF"
  if (StrDataIn == "OFF") { // Si StrDataIn es igual a "OFF"
    digitalWrite(PIN_SALIDA_1, LOW); // Establece PIN_SALIDA_1 en cero
  } else if (StrDataIn == "ON") { // Si StrDataIn es igual a "ON"
    digitalWrite(PIN_SALIDA_1, HIGH); // Establece PIN_SALIDA_1 en uno o encendido
  }

  ValHum = dht.readHumidity(); // Lee la humedad del sensor DHT22
  ValTem = dht.readTemperature(); // Lee la temperatura del sensor DHT22

  if (isnan(ValHum) || isnan(ValTem)) { // Si ValHum o ValTem no son un número válido
    ValHum = 0; // Establece ValHum en 0
    ValTem = 0; // Establece ValTem en 0
    Serial.println(F("Error de lectura del sensor DHT22!")); // Imprime un mensaje de error
  }

  int sensorValue = analogRead(Hum_Suelo); // Lee el valor del sensor de humedad de suelo del sensor FC-28
  HumSueloVal = map(sensorValue, 0, 4095, 99, 0); // Mapea el valor del sensor a un rango de 0 a 99
}

```

Figura 3.25 Función *void OnRxDone*

Luego, se lee la humedad y la temperatura utilizando el *sensor DHT22* y se verifican las lecturas para asegurarse de que sean válidas. Adicionalmente, se lee el valor de humedad del suelo a través del *sensor analógico FC-28* conectado al pin *Hum_Suelo*. Además, esta función se encarga de comparar los datos que recibe, esto es crucial para activar el relé según los parámetros establecidos en *Adafruit IO*.

Luego, los datos se formatean en una cadena (*StrDataResp*) que se convierte en un paquete de datos para ser enviado de vuelta a través de la radio *LoRa*. Finalmente, la función envía este paquete y establece el estado de la radio *LoRa* a inactivo, preparándola para la siguiente operación. Esto se ilustra en la Figura 3.26

```

StrDataInVal = StrDataIn.toInt();
// Verificar y controlar el PIN_SALIDA_1 según la comparación de valores
if (HumSueloVal < StrDataInVal) {
    digitalWrite(PIN_SALIDA_1, HIGH); // Enciende PIN_SALIDA_1
    while (HumSueloVal < StrDataInVal) {
        sensorValue = analogRead(Hum_Suelo); // Lee el valor del sensor de nuevo
        HumSueloVal = map(sensorValue, 0, 4095, 99, 0); // Actualiza el valor mapeado
        delay(1000); // Espera 1 segundo antes de la próxima lectura
    }
    digitalWrite(PIN_SALIDA_1, LOW); // Apaga PIN_SALIDA_1 cuando la condición ya no se cumple
}

Serial.print("% Humedad: "); // Imprime el valor de Humedad ambiental
Serial.print(ValHum);
Serial.print("% Temperatura: "); // Imprime el valor de Temperatura
Serial.print(ValTem);
Serial.print("%C Humedad Suelo: "); // Imprime el valor de humedad de suelo
Serial.print(HumSueloVal);
Serial.println();
StrDataResp = String("DataRespuesta") + String("@") + String(ValHum, 2) + String("@") + String(ValTem, 2) + String("@") + String(HumSueloVal); // Concatena

int str_len = StrDataResp.length() + 1; // Calcula la longitud de StrDataResp más uno
char char_TramaSend[str_len]; // Crea un buffer para la trama de datos a enviar
StrDataResp.toCharArray(char_TramaSend, str_len); // Convierte la cadena de datos de respuesta a un array de caracteres
sprintf(txpacket, char_TramaSend); // Copia la trama de datos a enviar en el buffer txpacket
Serial.printf("\r\nEnviando Respuesta \"%s\" , longitud %d\r\n", txpacket, strlen(txpacket)); // Imprime la trama de datos a enviar y su longitud en el mon
Radio.Send((uint8_t *)txpacket, strlen(txpacket)); // Envía la trama de datos a través de la radio LoRa
lora_idle = false; // Cambia el estado de LoRa a activo

```

Figura 3.26 Función *void OnRxDone* envió del mensaje por *LoRa*

Función *OnTxDone* en el SoC esclavo

La función *void OnTxDone(void)* que se visualiza en la Figura 3.27 es un manejador de eventos que se invoca cuando el SoC esclavo ha completado la transmisión de un paquete de datos. En esta función, se imprime un mensaje en el monitor serial indicando que la transmisión ha finalizado. Luego, se cambia el estado de la variable *LoRa_idle* a *true*, lo que señala que el módulo *LoRa* está listo para la siguiente operación. Finalmente, la función pone la radio en modo de recepción *Radio.Rx(0)*, preparándola para recibir nuevos datos entrantes. Esta secuencia asegura una transición correcta entre los estados de transmisión y recepción, manteniendo el SoC esclavo en funcionamiento continuo y eficiente.

```

void OnTxDone(void) {
    Serial.println("TX done....."); // Imprime un mensaje indicando que la transmisión ha finalizado
    lora_idle = true; // Cambia el estado de LoRa a inactivo
    Radio.Rx(0); // Pone la radio en modo de recepción
}

```

Figura 3.27 Función *void OnTxDone*

Función *void OnTxTimeout* en el SoC esclavo

La función *void OnTxTimeout(void)* que se muestra en la Figura 3.28 se invoca cuando el SoC esclavo experimenta un tiempo de espera durante la transmisión, es decir, cuando la transmisión de un paquete de datos no se completa dentro del tiempo especificado. En esta función, el SoC se pone en modo de reposo *Radio.Sleep()* para ahorrar energía y detener cualquier actividad de transmisión pendiente. Luego, se imprime un mensaje en el monitor serial *TX Timeout* para notificar al usuario que ha ocurrido un tiempo de espera. Finalmente, la variable *LoRa_idle* se establece en *true*, indicando que el SoC LoRa está inactivo y listo para la próxima operación, ya sea intentar una nueva transmisión o cambiar a modo de recepción. Esta función es crucial para manejar errores de transmisión y mantener la estabilidad de la comunicación.

```

void OnTxTimeout(void) {
    Radio.Sleep(); // Pone la radio en modo de reposo
    Serial.println("TX Timeout....."); // Imprime un mensaje indicando que la transmisión ha superado el tiempo de espera
    lora_idle = true; // Cambia el estado de LoRa a inactivo
}

```

Figura 3.28 Función *void OnTxTimeout* en el SoC esclavo

Librerías para el SoC maestro

En el SoC maestro, las librerías incluidas en esta sección del código que se ilustra en la Figura 3.29 son esenciales para la comunicación inalámbrica y la conectividad a Internet. La librería *LoRaWan_APP.h* habilita el uso del protocolo *LoRa*, ideal por su bajo consumo de energía y largo alcance [16]. *Arduino.h* proporciona las funciones básicas del *Arduino IDE*. Las librerías *WiFi.h* y *PubSubClient.h* permiten conectar el dispositivo a una red *WiFi* y comunicarse con un broker MQTT, facilitando la publicación y suscripción de mensajes en *Adafruit IO*. Finalmente, *Ticker.h* permite crear temporizadores no bloqueantes, cruciales para ejecutar tareas repetitivas sin interferir con el flujo principal del programa.

```

#include "LoRaWan_APP.h" //Librería para habilitar la comunicación LoRa y su respectiva transmisión y recepción
#include "Arduino.h" //Librería base de Arduino, configuración de pines, temporización, manejo de interrupciones.
#include <WiFi.h> //Esta librería permite la conexión del microcontrolador con redes WiFi
#include <PubSubClient.h> //Esta librería implementa el protocolo MQTT y facilita la comunicación con Adafruit IO.
#include <Ticker.h> //Permite la creación y gestión de temporizadores periódicos.
Ticker tiempo_1; //timer de cada que tiempo solicito una llamada al otro heltec, permite ejecutar funcion_1 cada 10 segundos

```

Figura 3.29 Librerías para el SoC maestro

Credenciales para conectarse a la red *WiFi* en el SoC maestro

En la Figura 3.30 se aprecia la sección del código donde se definen las credenciales de la red *WiFi* a la que se conectará el SoC maestro. Esto permite la transmisión de información hacia *Adafruit IO*.

```

//*****
// Credenciales Red WiFi
#define WIFI_SSID "Moises_Fullnet" // Define el SSID de la red WiFi a la que el SoC se va a conectar
#define WIFI_PASSWORD "1727622217Sa$" // Define la contraseña WiFi

```

Figura 3.30 Credenciales para la red *WiFi* a conectarse

Credenciales para *Adafruit IO* en el SoC maestro

En esta sección del código del SoC maestro, representada en la Figura 3.31, se establecen las credenciales y configuraciones necesarias para la comunicación con *Adafruit IO* mediante *MQTT*. Primero, se definen las credenciales de usuario *ADAFRUIT_USER* y la clave de *API ADAFRUIT_KEY* que permiten la autenticación en *Adafruit IO* para el envío y recepción de datos. Además, se especifica la dirección del servidor *MQTT de Adafruit IO ADAFRUIT_SERVER* y el puerto utilizado *ADAFRUIT_PORT*, asegurando la correcta conexión con el servicio *IoT*.

Además de las configuraciones de conexión, se definen las direcciones de los *feeds MQTT* donde se publicarán los datos de humedad *ADAFRUIT_FEED_Humd* y temperatura *ADAFRUIT_FEED_Temp* generados por el SoC esclavo. Estos *feeds* facilitan la organización y el envío de datos específicos a través de *Adafruit IO*. Asimismo, se establece un *feed* para la suscripción de datos entrantes desde *Adafruit IO ADAFRUIT_DATA_IN* y *ADAFRUIT_ADJUST_HUMIDITY*, completando así la configuración necesaria para la comunicación bidireccional entre el SoC maestro y *Adafruit IO* mediante el protocolo *MQTT*.

```

// Credenciales Adafruit
#define ADAFRUIT_USER "Samuelinlago22" // Define el nombre de la cuenta de usuario en Adafruit IO.
#define ADAFRUIT_KEY "aio_xKST77iZvxZPKNlYcrZldeBoGhDM" // Define la clave de API (API Key) para la cuenta de Adafruit IO
#define ADAFRUIT_SERVER "io.adafruit.com" // Define la dirección del servidor MQTT de Adafruit IO
#define ADAFRUIT_PORT 1883 // Define el puerto del servidor MQTT de Adafruit IO
char ADAFRUIT_ID[30]; // Declara un arreglo de caracteres String llamado ADAFRUIT_ID con un tamaño de 30 caracteres

// Publicar
#define ADAFRUIT_FEED_Humd ADAFRUIT_USER "/feeds/Heltec_Humd" // Define la dirección del feed MQTT para publicar datos de humedad
#define ADAFRUIT_FEED_Temp ADAFRUIT_USER "/feeds/Heltec_Temp" // Define la dirección del feed MQTT para publicar datos de temperatura
#define ADAFRUIT_FEED_HumSuelo ADAFRUIT_USER "/feeds/Heltec_HumSuelo" // Define la dirección del feed MQTT para publicar datos de humedad del suelo

// Suscripción
#define ADAFRUIT_DATA_IN ADAFRUIT_USER "/feeds/dataOn_Off" // Define la dirección del feed MQTT para suscribirse y recibir datos, se usa para recib:
#define ADAFRUIT_ADJUST_HUMIDITY ADAFRUIT_USER "/feeds/Ajuste_Humedad" // Define la dirección del feed MQTT para suscribirse y recibir datos, se usa |

```

Figura 3.31 Definición de credenciales para servidor de Adafruit IO

Configuración de parámetros LoRa en el SoC maestro

Al igual que el SoC esclavo, el SoC maestro también necesita configurar los parámetros de LoRa para establecer una comunicación efectiva entre ambos como se ilustra en la Figura 3.32. La frecuencia de radio se define en 915 (MHz), y la potencia de salida se establece en 5 (dBm). El ancho de banda de LoRa se configura en 0, el factor de expansión en 7, y la tasa de codificación en 1. El preámbulo tiene una longitud de 8 símbolos, con un tiempo de espera de símbolo de 0. La longitud fija del *payload* está desactivada, al igual que la inversión de IQ. El valor de tiempo de espera de recepción se establece en 1000 (ms), y el tamaño del buffer es de 30 (bytes).

```

#define RF_FREQUENCY 915000000 //Define la frecuencia de radio en la que opera el dispositivo LoRa, establecida en 915 MHz
#define TX_OUTPUT_POWER 14 // Establece la potencia de salida para las transmisiones LoRa en 14 dBm, que es la potencia nomi
#define LORA_BANDWIDTH 0 // Define el ancho de banda utilizado para las comunicaciones LoRa. Aquí, 0 representa un ancho d
#define LORA_SPREADING_FACTOR 7 // Especifica el factor de expansión para las transmisiones LoRa. Un valor de 7 indica un factor i
#define LORA_CODINGRATE 1 //Configura la tasa de codificación utilizada en las transmisiones LoRa. Aquí, 1 representa una t
#define LORA_PREAMBLE_LENGTH 8 //Define la longitud del preámbulo utilizado en las transmisiones LoRa, especificado en 8 símbolo
#define LORA_SYMBOL_TIMEOUT 0 //Establece el tiempo de espera de símbolos para las comunicaciones LoRa. En este caso, se config
#define LORA_FIX_LENGTH_PAYLOAD_ON false // Habilita o deshabilita el uso de longitud fija de carga útil en las transmisiones LoRa. Aquí, es
#define LORA_IQ_INVERSION_ON false // Controla la inversión de la modulación IQ en las transmisiones LoRa. En este caso, está configura

#define RX_TIMEOUT_VALUE 1000 //Define el tiempo de espera para la recepción en milisegundos. Aquí se establece en 1000 ms
#define BUFFER_SIZE 30 // Especifica el tamaño del búfer para almacenar datos, configurado en 30 bytes para esta aplicación.

```

Figura 3.32 Parámetros LoRa en el SoC maestro

Declaración de variables globales y funciones en el SoC maestro

Esta sección del código en el SoC maestro que se ilustra en la Figura 3.33 se enfoca en la declaración de variables globales y funciones que son esenciales para el desarrollo de este código. Las variables globales como *ValorFloat*, *LoRa_idle*, *BandSendData*, *StrTramaSend*, *StrDataLoRaIn*, *StrIndx*, *Val_Humd*, *Val_Temp*, y *Val_HumSuelo* se utilizan para almacenar y manejar datos críticos a lo largo del programa. Estas variables

permiten almacenar valores del sensor de humedad, temperatura y humedad del suelo, gestionar el estado del módulo *LoRa*, si está listo para enviar o recibir, y preparar datos para la transmisión. Por ejemplo, *LoRa_idle* es un indicador que muestra si el módulo *LoRa* está inactivo y listo para enviar o recibir datos, mientras que *BandSendData* indica si hay datos listos para ser enviados.

Además, se declaran prototipos de funciones que definen las operaciones críticas del sistema, incluyendo la configuración de *WiFi setup_WiFi*, la gestión de eventos *MQTT callback, reconnect, mqtt_publish*, y la manipulación de eventos de transmisión y recepción *LoRa OnTxDone, OnTxTimeout, OnRxDone*. También se incluye una estructura de eventos *RadioEvents* para manejar las interrupciones de *LoRa*. Estas funciones permiten configurar la conexión *WiFi*, gestionar la comunicación con el servidor *MQTT*, y procesar la transmisión y recepción de datos a través de *LoRa*. La función *getValue* es una utilidad para extraer valores de una cadena de datos basándose en un separador específico, y *funcion_1* es una función periódica invocada por un temporizador para indicar que hay datos listos para enviar.

En conjunto, esas declaraciones establecen la estructura básica del programa y preparan las operaciones necesarias para la comunicación y procesamiento de datos

```
//Declaración de variables globales y funciones
float ValorFloat; // Variable flotante para almacenar un valor que posiblemente se incremente después de cada transmisión
bool lora_idle = true; // Bandera que indica si el módulo LoRa está en modo inactivo y listo para enviar/recibir
bool BandSendData = false; // Bandera que indica si hay datos listos para ser enviados
String StrTramaSend; // Cadena que almacenará los datos que se van a enviar a través de LoRa
String StrDataLoraIn, StrIndx; // Cadenas para almacenar datos recibidos a través de LoRa y el índice extraído de los datos
float Val_Humd, Val_Temp, Val_HumSuelo; // Variables flotantes para almacenar valores de humedad, temperatura y humedad del suelo

static RadioEvents_t RadioEvents; // Estructura que contiene los eventos del radio (transmisión y recepción)
void OnTxDone(void); // Declaración de la función de callback que se ejecuta cuando la transmisión LoRa se completa
void OnTxTimeout(void); // Declaración de la función de callback que se ejecuta cuando la transmisión LoRa excede el tiempo de espera
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr); // Declaración de la función de callback que se ejecuta cuando la recepción LoRa
String getValue(String data, char separator, int index); // Declaración de la función que extrae un valor de una cadena basado en un índice y un separador

//*****
WiFiClient espClient; // Objeto para manejar la conexión WiFi
PubSubClient client(espClient); // Cliente MQTT que utiliza la conexión WiFi
void setup_wifi(); // Declaración de la función para configurar la conexión WiFi
void callback(char* topic, byte* payload, unsigned int length); // Declaración de la función de callback que se ejecuta cuando se recibe un mensaje MQTT
void reconnect(); // Declaración de la función para reconectar al servidor MQTT si la conexión se pierde
void mqtt_publish(String feed, float val); // Declaración de la función para publicar un valor en un feed MQTT
void get_MQTT_ID(); // Declaración de la función para obtener el ID único del microcontrolador para MQTT
//*****
void funcion_1(); // Declaración de la función que se ejecuta periódicamente por el temporizador
```

Figura 3.33 Declaración de variables globales y funciones en el SoC maestro

Función *void setup* en el SoC maestro

Esta función que se aprecia en la Figura 3.34 tiene la función de inicializar varios componentes, incluyendo la configuración de comunicación serie, la conexión *WiFi*, y la configuración del módulo *LoRa*. Al comenzar, se inicia la comunicación serie con una velocidad de 115200 *baudios* para la depuración, lo que permite monitorear la actividad del *microcontrolador* a través del monitor serial. Luego, se obtiene el *ID* único del *microcontrolador* utilizando la función *get_MQTT_ID()* y se establece la conexión *WiFi* con la función *setup_WiFi()*. A continuación, se configura el cliente MQTT para conectarse al servidor de *Adafruit IO* en el puerto establecido y se establece la función de *callback* que se llamará cuando se reciba un mensaje *MQTT*. Estas configuraciones son esenciales para asegurar que el SoC esclavo esté conectado a la red y pueda comunicarse con el servidor MQTT.

Posteriormente, se inicializa el SoC con *Mcu.begin()* y se configuran los eventos del módulo de radio *LoRa*, asignando las funciones correspondientes a los eventos de transmisión y recepción *OnTxDone*, *OnTxTimeout*, *OnRxDone*. La comunicación *LoRa* se inicializa con los eventos configurados y se establece el canal de radio en la frecuencia de 915 (MHz). Se configuran los parámetros de transmisión y recepción del *LoRa*, incluyendo el ancho de banda, el factor de expansión, la tasa de codificación, la longitud del preámbulo y otros parámetros necesarios para la comunicación *LoRa*. Finalmente, se configura un temporizador que llama a la función *funcion_1* cada 10 segundos, lo que permite realizar tareas periódicas como la recopilación y el envío de datos.

```
void setup() {
  Serial.begin(115200); // Inicializa la comunicación serie con una velocidad de 115200 baudios para la depuración

  //*****
  get_MQTT_ID(); // Llama a la función para obtener el ID único del SoC y lo guarda en la variable ADAFRUIT_ID
  setup_wifi(); // Llama a la función para conectar el microcontrolador a la red WiFi utilizando las credenciales definidas
  client.setServer(ADAFRUIT_SERVER, ADAFRUIT_PORT); // Configura el cliente MQTT para conectarse al servidor de Adafruit IO utilizando el puerto definido
  client.setCallback(callback); // Establece la función de callback que se llamará cuando se reciba un mensaje MQTT
  //*****

  Mcu.begin(); // Inicializa el SoC, configurando cualquier hardware o periféricos necesarios
  RadioEvents.TxDone = OnTxDone; // Asigna la función OnTxDone al evento TxDone de LoRa, que se llama cuando una transmisión LoRa se completa.
  RadioEvents.TxTimeout = OnTxTimeout; // Asigna la función OnTxTimeout al evento TxTimeout de LoRa, que se llama cuando una transmisión LoRa excede el tiempo
  RadioEvents.RxDone = OnRxDone; // Asigna la función OnRxDone al evento RxDone de LoRa, que se llama cuando una recepción LoRa se completa.

  Radio.Init(&RadioEvents); // Inicializa el módulo LoRa con los eventos configurados en RadioEvents
  Radio.SetChannel(RF_FREQUENCY); // Configura el canal LoRa en la frecuencia definida (915 MHz)
  Radio.SetTxConfig(MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH, // Configura los parámetros de transmisión LoRa
    LORA_SPREADING_FACTOR, LORA_CODINGRATE,
    LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
    true, 0, 0, LORA_IQ_INVERSION_ON, 3000);
  Radio.SetRxConfig(MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR, // Configura los parámetros de recepción LoRa
    LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
    LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
    0, true, 0, 0, LORA_IQ_INVERSION_ON, true);

  tiempo_1.attach(10, funcion_1); // Configura al temporizador para llamar a la funcion_1 cada 10 segundos
}
```

Figura 3.34 Función *void setup* en el SoC maestro

Función *void loop* en el SoC maestro

La función *loop* que se ilustra en la Figura 3.35 se encarga principalmente de gestionar la conexión *MQTT* y el envío de datos a través del módulo *LoRa*. Primero verifica si la conexión *MQTT* está activa, si no lo está, intenta reconectarse utilizando la función *reconnect()*. Luego, mantiene la conexión *MQTT* activa mediante *client.loop()*. Además, la función verifica si hay datos listos para enviar *BandSendData* es *true* y si el módulo *LoRa* está inactivo *LoRa_idle* es *true*. Si ambas condiciones se cumplen, prepara y envía los datos por *LoRa*, marcando el módulo como ocupada *LoRa_idle* a *false* para evitar interferencias durante la transmisión.

Además, en términos de procesamiento de datos, cuando hay datos listos para enviar, la función convierte una cadena de texto *String* a un arreglo de caracteres *char array* y prepara el paquete de transmisión *txpacket*. Este paquete se imprime en el monitor serial y luego se envía mediante la función *Radio.Send()*. Después de la transmisión, se incrementa un contador *ValorFloat*. Finalmente, la función *Radio.IrqProcess()* se llama para manejar las interrupciones del módulo *LoRa*, procesando eventos como transmisiones y recepciones completadas.

```
void loop() {
  if (!client.connected()) { // Si el cliente MQTT no está conectado
    | | reconnect(); // Se reconecta al servidor MQTT si la conexión se pierde
  }
  client.loop(); // Mantiene la conexión MQTT activa, se encarga de gestionar la comunicación y llamadas de retorno

  if (BandSendData && lora_idle) { // Si BandSendData es true y lora_idle es true, significa que hay datos listos para enviar y el módulo LoRa está inactivo
    BandSendData = false; // Resetea la estructura de envío de datos a false para evitar reenvíos no deseados

    // ACA SE REALIZA LA CONVERSION DE String a CharArray que se requiere de enviar en ese formato
    int str_len = StrTramaSend.length() + 1; // Calcula la longitud de la cadena a enviar más el carácter nulo al final
    char char_TramaSend[str_len]; // Declara un arreglo de caracteres con la longitud necesaria
    StrTramaSend.toCharArray(char_TramaSend, str_len); // Convierte la cadena String a un arreglo de caracteres

    sprintf(txpacket, char_TramaSend); // Copia la cadena de caracteres al paquete de transmisión txpacket

    Serial.printf("\r\nEnviando Paquete \"%s\", longitud %d\r\n", txpacket, strlen(txpacket)); // Imprime en el monitor serie el paquete que se va a enviar
    Radio.Send((uint8_t *)txpacket, strlen(txpacket)); // Envía el paquete por LoRa. La función Send toma como parámetros un puntero al paquete de datos y l
    lora_idle = false; // Marca el módulo LoRa como ocupado
    ValorFloat++; // Incrementa la variable ValorFloat, que podría ser utilizada para llevar un registro del número de transmisiones
  }
  Radio.IrqProcess(); // Procesa las interrupciones de LoRa, manejando eventos como transmisiones y recepciones completadas
}
```

Figura 3.35 Función *void loop* en el SoC maestro

Función *void funcion_1* en el SoC maestro

La función *funcion_1()* que se ilustra en la Figura 3.36 se utiliza para indicar que hay datos listos para ser enviados a través de la comunicación *LoRa*. Esto se logra estableciendo la variable *BandSendData* en *true*. Además, esta función es llamada periódicamente por el temporizador configurado en la función *setup* utilizando la librería *Ticker*, que llama a *funcion_1()* cada 10 segundos.

```

void funcion_1() {
| | BandSendData = true; // Indica que hay datos listos para enviar por LoRa
}

```

Figura 3.36 Función *void funcion_1* en el SoC maestro

Función *void OnTxDone(void)* en el SoC maestro

La función *OnTxDone* que se aprecia en la Figura 3.37 se ejecuta automáticamente cuando el SoC LoRa ha finalizado la transmisión de datos. Su objetivo es gestionar el estado del SoC después de la transmisión. Al ejecutarse, la función imprime un mensaje en el monitor serial para indicar que la transmisión ha concluido, cambia el estado de la variable *LoRa_idle* a *true* para señalar que el módulo LoRa está inactivo y disponible para nuevas tareas, y configura el módulo para entrar en modo de recepción. Esto permite al dispositivo cambiar de manera eficiente entre los estados de transmisión y recepción, asegurando que esté listo para recibir nuevos datos inmediatamente después de completar una transmisión.

```

void OnTxDone(void) {
| Serial.println("TX done....."); // Imprime un mensaje indicando que la transmisión ha finalizado
| lora_idle = true; // Establece la bandera lora_idle a true, indicando que el módulo LoRa está inactivo y listo para la siguiente tarea
| Radio.Rx(0); // Habilita el modo receptor después de la transmisión del módulo LoRa, permitiendo que el dispositivo esté listo para recibir datos
}

```

Figura 3.37 Función *void OnTxDone* en el SoC maestro

Función *OnTxTimeout* en el SoC maestro

La función *OnTxTimeout* se ejecuta cuando una transmisión de datos mediante el SoC maestro no se completa a tiempo. Esta función que se ilustra en la Figura 3.38 coloca al SoC esclavo en modo de suspensión para ahorrar energía, imprime un mensaje de tiempo de espera en el monitor serial, y establece la variable *LoRa_idle* a *true*, indicando que el SoC está listo para nuevas operaciones. Entonces, esta función asegura que el sistema maneje adecuadamente las fallas en la transmisión mediante comunicación LoRa, permitiendo que futuras transmisiones puedan intentarse nuevamente.

```

void OnTxTimeout(void) {
| Radio.Sleep(); // Coloca el módulo LoRa en modo de suspensión para ahorrar energía
| Serial.println("TX Timeout....."); // Imprime un mensaje en el monitor serie indicando que ha ocurrido un tiempo de espera en la transmisión
| lora_idle = true; // Establece la bandera lora_idle a true, indicando que el módulo LoRa está inactivo y listo para la siguiente tarea
}

```

Figura 3.38 Función *void OnTxTimeout* en el SoC maestro

Función *OnRxDone* en el SoC maestro

La función *void OnRxDone (uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)* que se ilustra en la Figura 3.39 tiene la función de procesar los datos recibidos a través de LoRa. Primero, copia los datos del *payload* recibido al *buffer rxpacket* y añade un

carácter nulo al final para formar una cadena de caracteres válida. Luego, imprime el contenido del paquete, la fuerza de la señal recibida *RSSI* y la longitud del paquete en el monitor serial. A continuación, convierte el *buffer rxpacket* en una cadena *String* y utiliza la función *getValue* para extraer valores específicos de la cadena basados en un separador '@', como índice, humedad, temperatura y humedad del suelo. Estos valores se imprimen en el monitor serial para verificación.

Después de extraer y mostrar la información, la función pública los valores de humedad, temperatura y humedad del suelo en los correspondientes *feeds* de *Adafruit IO* usando *MQTT*. Cada publicación se realiza con un breve retraso *delay (10)* o 10 milisegundos entre ellas para asegurar una comunicación correcta. En resumen, la función *OnRxDone* no solo maneja la recepción de datos *LoRa*, sino que también se encarga de la descomposición y publicación de los datos recibidos a través de *MQTT*.

```
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr) {
  memcpy(rxpacket, payload, size); // Copia los datos recibidos (payload) al buffer rxpacket, con el tamaño especificado por size
  rxpacket[size] = '\0'; // Agrega un carácter nulo al final del buffer rxpacket para convertirlo en una cadena de caracteres
  // Imprime el paquete recibido, el RSSI (indicador de la fuerza de la señal) y la longitud del paquete en el monitor serie
  Serial.printf("\r\nPaquete Recibido \"%s\" con RSSI %d , LONGITUD %d\r\n", rxpacket, rssi, size);

  StrDataLoraIn = String(rxpacket); // Convierte el buffer rxpacket a un objeto String y lo almacena en StrDataLoraIn
  // Extrae valores específicos de la cadena StrDataLoraIn usando el separador '@' y los índices correspondientes
  StrIdx = getValue(StrDataLoraIn, '@', 0);
  Val_Humd = (getValue(StrDataLoraIn, '@', 1)).toFloat();
  Val_Temp = (getValue(StrDataLoraIn, '@', 2)).toFloat();
  Val_HumSuelo = (getValue(StrDataLoraIn, '@', 3)).toFloat();
  // Imprime los valores extraídos (índice, humedad, temperatura y humedad del suelo) en el monitor serial
  Serial.print("Índice: "); Serial.println(StrIdx);
  Serial.print(" Val_Humd: "); Serial.print(Val_Humd);
  Serial.print(" Val_Temp: "); Serial.print(Val_Temp);
  Serial.print(" Val_HumSuelo: "); Serial.println(Val_HumSuelo);
  // Publica los valores extraídos en los feeds de Adafruit IO correspondientes.
  mqtt_publish(ADAFRUIT_FEED_Humd, Val_Humd);
  delay(10);
  mqtt_publish(ADAFRUIT_FEED_Temp, Val_Temp);
  delay(10);
  mqtt_publish(ADAFRUIT_FEED_HumSuelo, Val_HumSuelo);
  delay(10);
}
```

Figura 3.39 Función *OnRxDone* en el SoC maestro

Función *String getValue* en el SoC maestro

Primero, es importante mencionar que la función *getValue* se utiliza en la función *OnRxDone* para extraer datos específicos de una cadena recibida a través de *LoRa*. En *OnRxDone*, después de convertir el paquete recibido en una cadena *String*, se llama a *getValue* para extraer valores individuales como el índice, humedad, temperatura y humedad del suelo de la cadena.

La función *getValue* que se ilustra en la Figura 3.40 se utiliza para extraer un valor específico de una cadena basada en un índice y un separador. Primero, inicializa una variable *found* para contar cuántos separadores se han encontrado, y un arreglo *strIdx* para almacenar los índices de inicio y fin del valor a extraer. Luego, recorre la cadena de entrada data carácter por carácter. Si encuentra un separador o llega al final de la

cadena, incrementa el contador *found*, y actualiza los índices en *strIndex* para señalar el inicio y el fin del valor encontrado.

Además, si el número de separadores encontrados es mayor que el índice solicitado, la función devuelve la *subcadena* correspondiente al valor entre los índices almacenados en *strIndex*. Si no se encuentra el valor porque el índice solicitado es mayor que la cantidad de separadores, la función devuelve una cadena vacía. De esta forma, *getValue* permite extraer partes específicas de una cadena de texto que están separadas por un carácter delimitador, como puede ser una coma, un espacio, o en este caso, el símbolo @.

```
String getValue(String data, char separator, int index) { // Función para obtener un valor específico de una cadena basada en un índice y un separador
    int found = 0; // Variable para contar cuántos separadores se han encontrado
    int strIndex[] = { 0, -1 }; // Arreglo para almacenar el índice de inicio y fin del valor extraído
    int maxIndex = data.length() - 1; // Índice máximo de la cadena

    for (int i = 0; i <= maxIndex && found <= index; i++) { // Bucle para recorrer la cadena hasta que se encuentre el valor deseado o se llegue al final
        if (data.charAt(i) == separator || i == maxIndex) { // Si se encuentra un separador o se llega al final de la cadena
            found++; // Incrementar el contador de separadores encontrados
            strIndex[0] = strIndex[1] + 1; // Actualizar el índice de inicio al carácter siguiente después del último separador encontrado
            strIndex[1] = (i == maxIndex) ? i+1 : i; // Actualizar el índice de fin al separador encontrado o al final de la cadena
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : ""; // Devolver el valor extraído si se encontró, o una cadena vacía si no se encontró
}
```

Figura 3.40 Función *getValue* en el SoC maestro

Función *void mqtt_publish ()* en el SoC maestro

La función *mqtt_publish* que se ilustra en la Figura 3.41 se encarga de publicar un valor en un tópico específico de *MQTT*. Primero, convierte el valor flotante *val* a una cadena de caracteres *String* con dos decimales. Luego, verifica si el cliente *MQTT* está conectado utilizando *client.connected()*. Si está conectado, utiliza el método *client.publish()* para publicar la cadena de valor en el *feed* de *Adafruit IO*. Finalmente, imprime en el monitor serial un mensaje indicando el *feed* y el valor que se ha publicado.

Esta función es crucial para enviar datos a *Adafruit IO*, permitiendo la comunicación eficiente de los valores de los sensores. Además, es importante mencionar que en la función *OnRxDone*, *mqtt_publish* se llama para enviar los valores de humedad, temperatura y humedad del suelo recibidos.

```
// Función para Publicar por MQTT
void mqtt_publish(String feed, float val) { //se encarga de publicar un valor en un tópico específico de MQTT
    String value = String(val, 2); // Convierte el valor float a una cadena con 2 decimales
    if (client.connected()) { // Verifica si el cliente MQTT está conectado
        client.publish(feed.c_str(), value.c_str()); // Publica el valor en el tópico especificado
        Serial.println("Publicando al tópico: " + String(feed) + " | mensaje: " + value); // Imprime en el Serial el tópico y el valor publicado
    }
}
```

Figura 3.41 Función *void mqtt_publish* en el SoC maestro

Función `setup_WiFi` en el SoC maestro

En la Figura 3.42 la función inicia con un pequeño retraso `delay(10)` para estabilizar el sistema antes de proceder con la conexión. Luego, imprime mensajes informativos en la consola serie mediante `Serial.println()` para indicar el inicio del intento de conexión a la red `WiFi` especificada por `WIFI_SSID`. Luego, utiliza `WiFi.begin(WIFI_SSID, WIFI_PASSWORD)` para iniciar el proceso de conexión, donde `WIFI_SSID` y `WIFI_PASSWORD` son variables que contienen el nombre de la red `WiFi` y la contraseña respectivamente.

Después, entra en un bucle `while` que se ejecuta continuamente hasta que el estado de conexión reportado por `WiFi.status()` sea igual a `WL_CONNECTED`, lo cual indica que se ha establecido una conexión exitosa. Durante este bucle, se muestra en la consola una serie de puntos para indicar visualmente que el dispositivo está intentando conectarse. Una vez que la conexión se establece, se imprime un mensaje confirmando la conexión exitosa junto con la dirección `IP` local del dispositivo obtenida mediante `WiFi.localIP()`, la cual es esencial para la comunicación a través de la red `WiFi`.

```
/**   CONEXIÓN WIFI   **  
/*****  
void setup_wifi() {  
    delay(10); // Breve pausa inicial para asegurar la estabilidad  
  
    // Nos conectamos a nuestra red Wifi  
    Serial.println();  
    Serial.print("Conectando a ");  
    Serial.println(String(WIFI_SSID)); // Imprime el nombre del SSID de la red WiFi a la que se va a conectar  
  
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD); // Inicia la conexión a la red WiFi utilizando las credenciales proporcionadas anteriormente  
  
    while (WiFi.status() != WL_CONNECTED) { // Espera hasta que se establezca la conexión  
        delay(500); // Pausa de medio segundo antes de volver a verificar el estado de la conexión  
        Serial.print("."); // Muestra un punto en la consola para indicar que aún se está intentando conectar  
    }  
  
    Serial.println(""); // Imprime una línea en blanco para separar visualmente los mensajes anteriores  
    Serial.println("Conectado a red WiFi!"); // Indica que la conexión WiFi ha sido establecida con éxito  
    Serial.println("Dirección IP: "); //Imprime la dirección Ip  
    Serial.println(WiFi.localIP()); // Imprime la dirección IP local asignada al dispositivo por el enrutador WiFi  
}
```

Figura 3.42 Función `void setup_WiFi` en el SoC maestro

Función `void callback` en el SoC maestro

La función `callback` que se aprecia en la Figura 3.43 maneja la recepción de mensajes `MQTT`, procesándolos y almacenándolos para su posterior transmisión. Primero, convierte el tema recibido `topic` en una cadena de caracteres de tipo `String` y construye el mensaje a partir de los datos del `payload`. Luego, elimina cualquier espacio en blanco alrededor del mensaje y lo imprime junto con el tema en el monitor serial.

Después, la función verifica si el tema del mensaje coincide con `ADAFRUIT_DATA_IN` o `ADAFRUIT_ADJUST_HUMIDITY`. Si el tema es uno de estos, guarda el mensaje en la variable global `StrTramaSend` y establece la bandera `BandSendData` en `true`, indicando que hay datos listos para ser enviados a través de la comunicación `LoRa`. Esto permite que el dispositivo solo procese y transmita mensajes relevantes según los temas de interés definidos que sea enviado a través de comunicación `LoRa`.

```
// Función para capturar datos por MQTT
void callback(char *topic, byte *payload, unsigned int length) { // maneja la recepción de mensajes MQTT, procesa y almacena los datos recibidos
    String mensaje = ""; // Declara una cadena vacía para almacenar el mensaje MQTT recibido
    String str_topic(topic); // Convierte el tema MQTT (char*) en un objeto String llamado str_topic

    for (uint16_t i = 0; i < length; i++) {
        mensaje += (char)payload[i]; // Concatena cada byte del payload en mensaje como caracteres
    }

    mensaje.trim(); // Elimina cualquier espacio en blanco al inicio o final de la cadena mensaje

    Serial.println("Tópico: " + str_topic); // Imprime en consola el tema del mensaje recibido
    Serial.println("Mensaje: " + mensaje); // Imprime en consola el contenido del mensaje MQTT

    if (str_topic == ADAFRUIT_DATA_IN || str_topic == ADAFRUIT_ADJUST_HUMIDITY) { // verifica si el tema del mensaje recibido por MQTT es uno de los dos temas
        StrTramaSend = mensaje; // Se guarda el mensaje en la variable global StrTramaSend para enviarlo vía LoRa
        BandSendData = true; // Se indica que hay datos listos para enviar por comunicación LoRa
    }
}
```

Figura 3.43 Función `void callback` en el SoC maestro

Función `void get_MQTT_ID` en el SoC maestro

Esta función `get_MQTT_ID` se encarga de obtener el identificador único del `chip ESP32` del SoC maestro usando `ESP.getEfuseMac()`, que es almacenado en una variable `uint64_t` llamada `chipid`. Luego, utiliza `snprintf` para formatear este identificador como una cadena de caracteres y lo guarda en la variable `ADAFRUIT_ID`. Este identificador único es importante para identificar de manera única cada dispositivo `ESP32` cuando se comunica con `Adafruit IO` a través de `MQTT`. En la Figura 3.44 se puede observar las líneas de código de esta función.

```
// Capturar el ChipID para Id de MQTT
void get_MQTT_ID() {
    uint64_t chipid = ESP.getEfuseMac(); // Obtiene el identificador único del chip ESP32 (EFUSE MAC)
    snprintf(ADAFRUIT_ID, sizeof(ADAFRUIT_ID), "%llu", chipid); // Formatea el identificador del chip como una cadena y lo guarda en ADAFRUIT_ID
}
```

Figura 3.44 Función `void get_MQTT_ID` en el SoC maestro

Función `void reconnect` en el SoC maestro

Esta función `reconnect` que se ilustra en la Figura 3.45 asegura que el dispositivo mantenga una conexión estable con el servidor `MQTT` de `Adafruit IO`. Itera continuamente mientras el cliente no esté conectado (`! client.connected()`). Cuando intenta conectarse `client.connect()`, utiliza las credenciales proporcionadas `ADAFRUIT_ID`, `ADAFRUIT_USER`, `ADAFRUIT_KEY`. Si la conexión es exitosa, se

suscribe automáticamente al *tópico* especificado para recibir datos *ADAFRUIT_DATA_IN* y *ADAFRUIT_ADJUST_HUMIDITY* En caso de fallo en la conexión, muestra el estado actual del cliente *MQTT* y espera 5 segundos antes de volver a intentar, lo que ayuda a manejar eficazmente los intentos de conexión fallidos y evita sobrecargar el servidor.

```
void reconnect() {
  while (!client.connected()) { // Mientras el cliente MQTT no esté conectado, se ejecuta este bucle
    if (client.connect(ADAFRUIT_ID, ADAFRUIT_USER, ADAFRUIT_KEY)) { // Intenta conectar al servidor MQTT de Adafruit con las credenciales y el ID
      Serial.println("MQTT conectado!"); // Si la conexión es exitosa, imprime que MQTT está conectado
      client.subscribe(ADAFRUIT_DATA_IN); // Se suscribe al tópico especificado para recibir datos de ON y OFF
      client.subscribe(ADAFRUIT_ADJUST_HUMIDITY); // Se suscribe al tópico especificado para recibir datos de Ajuste de Humedad
      Serial.println("Suscrito a los tópicos: " + String(ADAFRUIT_DATA_IN) + " y " + String(ADAFRUIT_ADJUST_HUMIDITY)); // Imprime el tópico al
    } else {
      Serial.print("Falló :( con error -> "); // Si la conexión falla, imprime un mensaje de error
      Serial.print(client.state()); // Imprime el estado actual del cliente MQTT
      Serial.println(" Intentamos de nuevo en 5 segundos"); // Indica que se intentará nuevamente la conexión en 5 segundos
      delay(5000); // Espera 5 segundos antes de intentar reconectar
    }
  }
}
```

Figura 3.45 Función *void reconnect* en el SoC maestro

Configuración de panel de control *web* en *Adafruit IO*

En el desarrollo de este prototipo para el control de un invernadero mediante comunicación *LoRa*, se ha seleccionado *Adafruit IO* como la plataforma para monitoreo *web*. Esta elección se fundamenta en la capacidad de *Adafruit IO* para facilitar la creación de un panel de control *web*, esencial para la supervisión y el ajuste de los parámetros del prototipo de invernadero de este proyecto.

Configuración de *feeds* en *Adafruit IO*

Una de las ventajas que ofrece *Adafruit IO* es la capacidad de crear *feeds* para almacenar los datos enviados desde el SoC maestro a través de *WiFi* utilizando el protocolo *MQTT*. Esto permite una gestión eficiente y organizada de la información recogida. A continuación, se detallará paso a paso el proceso para crear *feeds* en *Adafruit IO*, facilitando así la integración y visualización de datos en el panel de control *web*:

- El primer paso es crear una cuenta en la plataforma *Adafruit IO*. Una vez registrada, es crucial guardar las credenciales y la clave API, ya que estas son necesarias para establecer la comunicación con el SoC maestro. Estas credenciales permiten el envío y recepción de datos de forma eficiente entre el dispositivo y la plataforma. En la Figura 3.46 luego de registrarse en la plataforma se muestra el nombre de usuario y contraseña de *Adafruit IO* para la integración con el código en *Arduino IDE*.

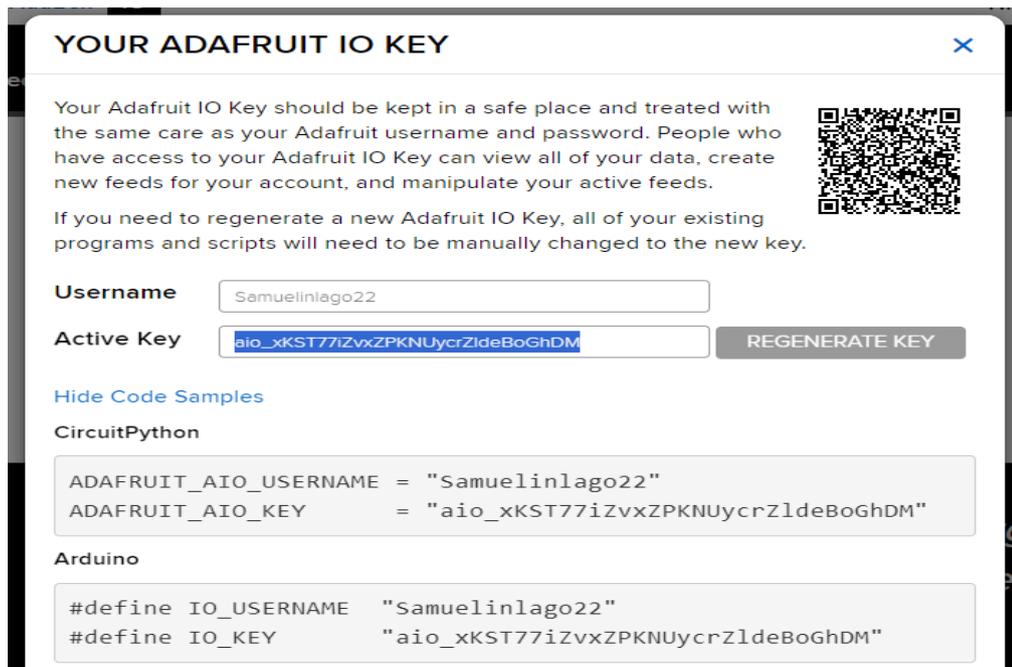


Figura 3.46 Credenciales Adafruit IO

- El siguiente paso como se ilustra en la Figura 3.47 para la creación de *feeds* en *Adafruit IO* es dirigirse a la sección de *Feeds* y seleccionar *New Feed*. Luego, se debe asignar un nombre a este feed, que será el encargado de almacenar los datos enviados desde el SoC maestro.

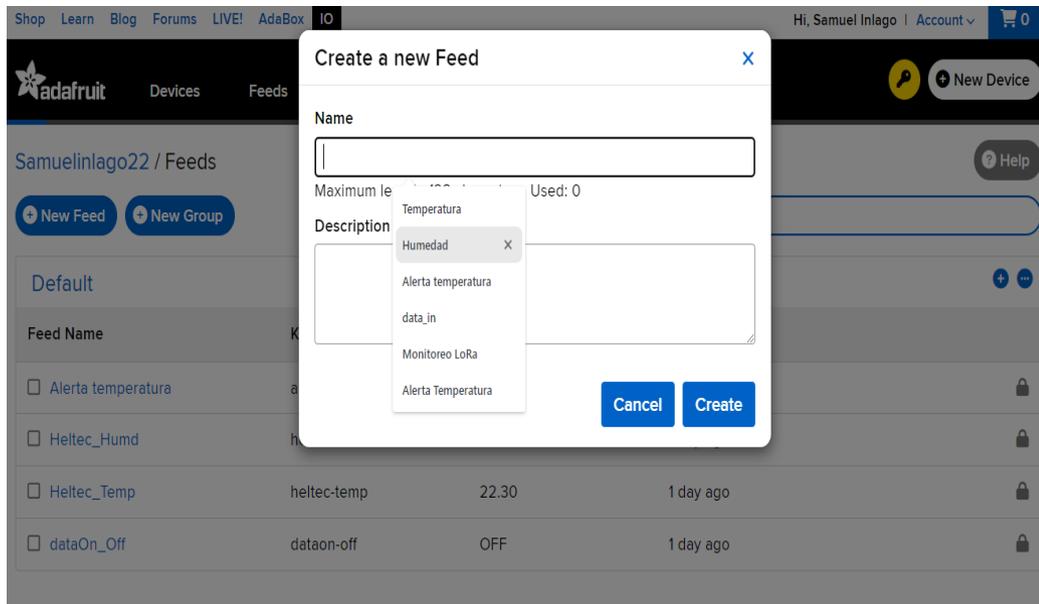


Figura 3.47 Creación de feeds en Adafruit IO

En la Figura 3.48 se puede visualizar todos los *feeds* creados para la integración con el proyecto. Toda la información enviada desde el SoC maestro se almacenan en estos *feeds* y estos mismo serán utilizados para crear los bloques en el panel de control.

Samuelinlago22 / Feeds Help

New Feed New Group

Default +			
Feed Name	Key	Last value	Recorded
<input type="checkbox"/> Ajuste_Humedad	ajuste-humedad	0	about 19 hours ago 🔒
<input type="checkbox"/> Heltec_HumSuelo	heltec-humsuelo	51.00	about 19 hours ago 🔒
<input type="checkbox"/> Heltec_Humd	heltec-humd	51.70	about 19 hours ago 🔒
<input type="checkbox"/> Heltec_Temp	heltec-temp	21.90	about 19 hours ago 🔒
<input type="checkbox"/> Temperatura_Ambiental	temperatura-ambiental	0	about 21 hours ago 🔒
<input type="checkbox"/> dataOn_Off	dataon-off	OFF	about 19 hours ago 🔒

Figura 3.48 Feeds creados en Adafruit IO

Configuración de un panel de control en Adafruit IO

Para la configuración de un panel de control destinado al monitoreo de los parámetros de humedad de suelo, temperatura, humedad ambiental, y configuración de parámetros se procedió de la siguiente manera. Inicialmente, se accedió a la sección de *dashboard* o panel de control dentro de la plataforma *Adafruit IO*. Posteriormente, se selecciona la opción para crear un nuevo *dashboard*. Durante este proceso, se asignó un nombre específico al *dashboard*, el cual reflejaba su propósito de monitoreo del prototipo de invernadero, en este caso fue “Control de invernadero”. En la Figura 3.49 se puede observar la creación del panel de control.

adafruit Devices Feeds Dashboards Actions Power-Ups 🔒 New Device

Samuelinlago22 / Dashboards Help

New Dashboard

Name	Key	Created At
<input type="checkbox"/> Control de invernadero	monitoreo-lora	May 28, 2024 🔒

Loaded in 0.31 seconds.

Get Help IO Status ●

Quick Guides Learn

API Documentation IO Plus

FAQ News

"Be quick, but don't hurry"

– John Wooden

Figura 3.49 Creación de un panel de control en Adafruit IO

Creación de bloques en el panel de control de Adafruit IO

La creación de bloques en el panel de control de *Adafruit IO* es fundamental, ya que estos bloques permiten monitorear de manera efectiva los datos almacenados en los *feeds*. A continuación, se presentan los pasos para crear bloques en el panel de control.

El primer paso, es ingresar al panel de control denominado "Control de invernadero", luego, se accede a la opción *Create New Block*. Al seleccionarla, se desplegarán varias opciones de bloques como se ilustra en la Figura 3.50. A continuación, se elige el bloque que mejor se ajuste al tipo de monitoreo.

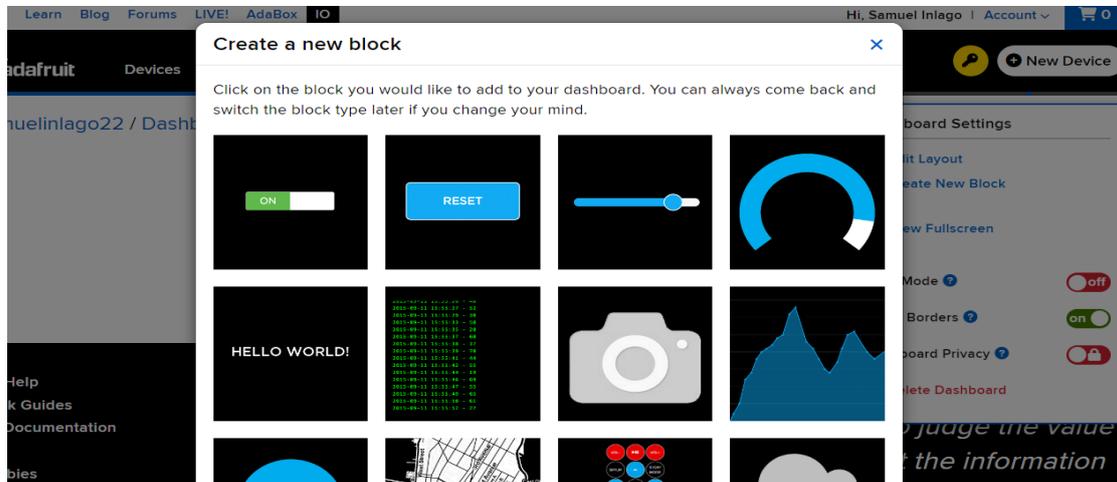


Figura 3.50 Creación de bloques en el panel de control

Una vez elegido el bloque que se utilizará en el panel de control, como se ilustra en la Figura 3.51 se procede a vincularlo con el *feed* que se desea visualizar en dicho bloque.



Figura 3.51 Selección de feed para vincular al bloque

A continuación, en la Figura 3.52, Figura 3.53 y Figura 3.54 se muestra la configuración de los bloques indicadores encargados de monitorear los datos de temperatura ambiental, humedad ambiental y humedad del suelo en el panel de control de *Adafruit IO*. Cada uno de estos bloques está configurado para cambiar de color cuando los datos alcanzan niveles críticos.

En el caso del bloque de la humedad ambiental, si el valor es menor al 10 (%), el color cambia para indicar una condición anormal, como se ilustra en la Figura 3.52.

Block settings



In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)	Block Preview
<input type="text" value="Humedad Ambiental"/>	
Gauge Min Value	
<input type="text" value="0"/>	
Gauge Max Value	
<input type="text" value="100"/>	
Gauge Width	
<input type="text" value="25px"/>	
Gauge Label	
<input type="text" value="%"/>	
Low Warning Value	
<input type="text" value="10"/>	
	<p>Gauge A gauge is a read only block type that shows a fixed range of values.</p>
	<p>Test Value</p> <input type="text" value="45"/>

Figura 3.52 Configuración del bloque para la humedad ambiental

En el caso del bloque de temperatura ambiental, si el valor supera los 40 (°C), el color cambia para indicar una condición anormal. En la Figura 3.53 se aprecia la configuración para este bloque.

Block Title (optional)	Block Preview
<input type="text" value="Temperatura Ambiental"/>	
Gauge Min Value	
<input type="text" value="0"/>	
Gauge Max Value	
<input type="text" value="100"/>	
Gauge Width	
<input type="text" value="25px"/>	
Gauge Label	
<input type="text" value="°C"/>	
Low Warning Value	
<input type="text"/>	
<small>Optional. If no low warning value is given, the gauge will only change color when the value is out of bounds.</small>	
High Warning Value	
<input type="text" value="40"/>	
	<p>Gauge A gauge is a read only block type that shows a fixed range of values.</p>
	<p>Test Value</p> <input type="text" value="45"/>

Figura 3.53 Ajuste del bloque para la temperatura ambiental

En el bloque para la humedad del suelo se configuró para que el bloque cambie de color si obtiene valores menores o igual al 10 (%), indicando así una condición anormal y, además, me indica que el suelo está totalmente seco en el prototipo. En la Figura 3.54 se aprecia el ajuste de este bloque.

Block settings



In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)	Block Preview
<input type="text" value="Humedad de suelo"/>	
Gauge Min Value	
<input type="text" value="0"/>	
Gauge Max Value	
<input type="text" value="100"/>	
Gauge Width	
<input type="text" value="25px"/>	
Gauge Label	
<input type="text" value="% Humedad Suelo"/>	
Low Warning Value	
<input type="text" value="10"/>	
Test Value	
<input type="text"/>	

Figura 3.54 Ajuste del bloque para la humedad del suelo

Para el sistema de irrigación se ha seleccionado un bloque en forma de palanca que se puede apreciar en la Figura 3.55. Su función es enviar los datos de *ON* y *OFF* al *feed* denominado *data_On_Off*. Este bloque cumple la función general de controlar el sistema de irrigación, permitiendo su activación y desactivación a través de estos datos por medio del panel de control.

Block settings



In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)	Block Preview
<input type="text" value="Sistema de Irrigación"/>	
Button On Text	
<input type="text" value="ON"/>	
Limit of 6 characters for the toggle text. Use the block title to be more descriptive.	
Button On Value (uses On Text if blank)	
<input type="text"/>	
Button Off Text	
<input type="text" value="OFF"/>	
Limit of 6 characters for the toggle text. Use the block title to be more descriptive.	
Toggle	
<input type="text"/>	

Figura 3.55 Control del sistema de Irrigación

Para la configuración de parámetros, se seleccionaron bloques de control deslizante que permiten capturar valores en un rango de 1 a 100 y almacenarlos en un *feed* específico seleccionado en *Adafruit IO*. Estos datos pueden utilizarse para enviarlos a través de *LoRa* o para compararlos con otros *feeds*, permitiendo generar alertas

mediante las Acciones en *Adafruit IO*. En la Figura 3.56 y Figura 3.57 se puede apreciar estos bloques.

En el bloque que se aprecia en la Figura 3.56 se configuró para utilizarlo para el ajuste de temperatura ambiental, además se estableció en un rango de temperatura de 0 a 50 (°C).

Block settings ✕

In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)	Block Preview
<input type="text" value="Ajuste de temperatura Ambiental"/>	 <p>45 °C</p> <p>Slider The slider works well if you have a range of values you need to send.</p> <p>Test Value</p>
Slider Min Value	
<input type="text" value="0"/>	
Slider Max Value	
<input type="text" value="50"/>	
Slider Step Size	
<input type="text" value="1"/>	
Slider Label	
<input type="text" value="°C"/>	
Decimal Places	
<input type="text" value="4"/>	

Figura 3.56 Ajuste de temperatura ambiental

En configuración del bloque que se aprecia en la Figura 3.57 se lo utilizó para poder realizar el ajuste de humedad deseado en el prototipo y se lo estableció entre 0 y 99 (%).

Block settings ✕

In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)	Block Preview
<input type="text" value="Ajuste de Humedad de suelo"/>	 <p>45 %</p> <p>Slider The slider works well if you have a range of values you need to send.</p> <p>Test Value</p>
Slider Min Value	
<input type="text" value="0"/>	
Slider Max Value	
<input type="text" value="99"/>	
Slider Step Size	
<input type="text" value="5"/>	
Slider Label	
<input type="text" value="%"/>	
Decimal Places	
<input type="text" value="4"/>	

Figura 3.57 Configuración de ajuste de humedad del suelo

En la Figura 3.58 se observa cómo queda finalmente configurado el panel de control completo, con todos los bloques vinculados a los *feeds* que almacenan los datos medidos por los sensores. Este panel de control permitirá monitorear de manera eficiente los parámetros de temperatura ambiental, humedad ambiental y humedad de

suelo del prototipo de invernadero. Al tener todos los bloques correctamente configurados, se puede visualizar de forma clara cualquier cambio en las condiciones ambientales del prototipo. Además, por medio de este panel de control configurado se puede activar o apagar el sistema de irrigación como también realizar el ajuste de parámetros.

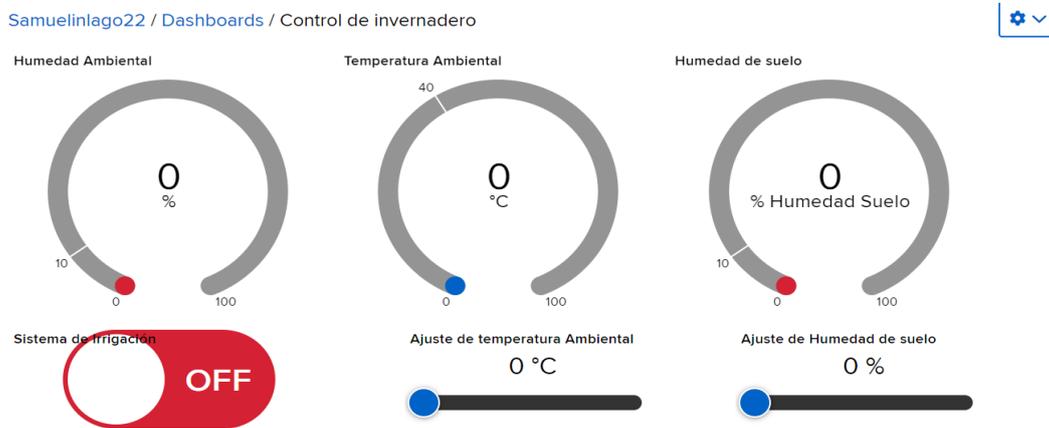


Figura 3.58 Panel de control configurado

Configuración de *actions* en *Adafruit IO* para enviar alertas al correo electrónico

Las Acciones de *Adafruit IO* permiten comparar los datos de los *feeds* y enviar correos electrónicos cuando se cumplen ciertas condiciones. Para configurar las Acciones en *Adafruit IO*, primero me dirijo a la opción Acciones en *Adafruit IO*. Luego, se selecciona Nueva Acción y aparecerá lo siguiente que se ilustra en la Figura 3.59.

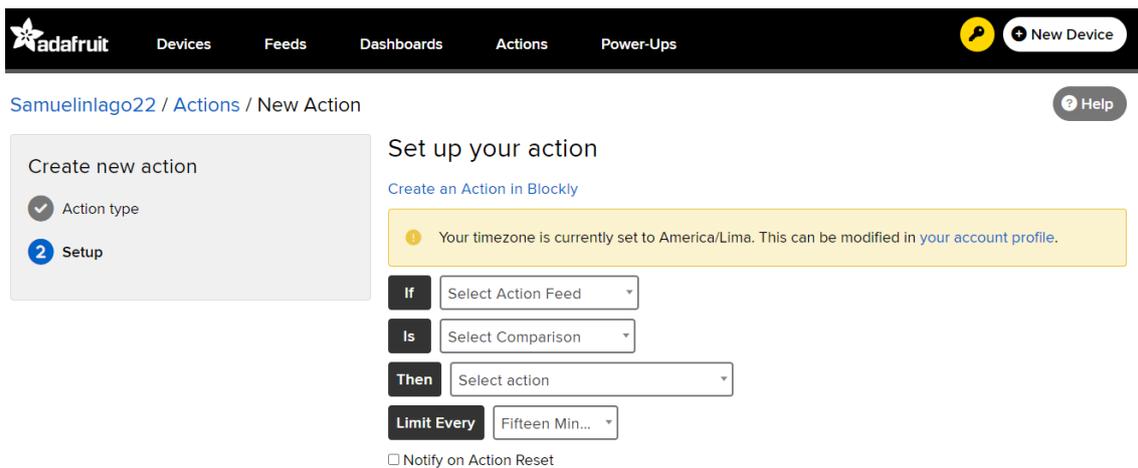


Figura 3.59 Configuración de acciones en *Adafruit IO*

También, se procede a configurar las acciones en *Blockly* como se aprecia en la Figura 3.60. Se configura las secciones de *If*, *Is*, *Then*, *Subject* y *Body*. En la sección *If*, se selecciona el *feed* que se va a comparar. Luego, en la opción *Is*, se selecciona *mayor*

que y se elige el otro feed con el cual se desea hacer la comparación. A continuación, en la opción *Then*, se selecciona enviarme un correo electrónico. Finalmente, se llena los campos *Subject* y *Body* para especificar cómo se desea que llegue el correo electrónico a la cuenta de *Gmail*, ya que la cuenta de *Adafruit IO* está vinculada a mi cuenta de *Gmail*.

Además, en la Figura 3.60 se visualiza la configuración para enviar alerta al correo electrónico en caso de un ajuste de parámetro en la humedad del suelo. En este caso nos enviara un correo electrónico cuando el sensor de humedad del suelo tenga un valor mayor al del parámetro deseado en el bloque de ajuste de humedad del suelo.

The screenshot shows the 'Set up your action' configuration interface. At the top, there is a yellow notification box stating: 'Your timezone is currently set to America/Lima. This can be modified in your account profile.' Below this, the configuration is as follows:

- If:** Heltec_HumSuelo
- Is:** greater than
- Then:** email me
- Value:** Ajuste_Humedad
- Unit:** value and time.
- Subject:** Alerta! el valor de la Humedad del Suelo sobrepasa el parámetro establecido que es de: {{value}}
- Body:** El valor de la Humedad del Suelo sobrepasa el límite establecido en {{value}} % con fecha {{created_at}}.

Figura 3.60 Configuración de alerta de ajuste de humedad del suelo

También, en la Figura 3.61 se puede visualizar la configuración para enviar una alerta en caso de un ajuste de parámetro en la temperatura ambiental. En este caso, se envía un correo electrónico cuando el valor medido por la temperatura ambiental supere el parámetro establecido en el bloque de ajuste de temperatura.

The screenshot shows the 'Set up your action' configuration interface for an ambient temperature alert. At the top, there is a yellow notification box stating: 'Your timezone is currently set to America/Lima. This can be modified in your account profile.' Below this, the configuration is as follows:

- If:** Heltec_Temp
- Is:** greater than
- Then:** email me
- Value:** Temperatura_Ambiental
- Unit:** value and time.
- Subject:** Alerta! el valor de Temperatura ambiental sobrepasa el parámetro establecido de: {{value}} °C
- Body:** El valor de la Temperatura ambiental sobrepasa el límite establecido en {{value}} °C con fecha {{created_at}}.

Figura 3.61 Configuración de alerta de ajuste de temperatura ambiental

A continuación, en la Figura 3.62 se ilustra la configuración para enviar una alerta al correo electrónico. Esta alerta se envía al instante que la temperatura ambiental medida por el sensor *DHT22* sea mayor o igual a 40 (°C), puesto que este valor es anormal en un invernadero [34].

The screenshot shows the 'Set up your action' interface. On the left, a sidebar indicates the current step is 'Setup' (2) out of 'Action type' (1). The main area is titled 'Set up your action' and includes a 'Help' button. A yellow notification bar states: 'Your timezone is currently set to America/Lima. This can be modified in your account profile.' The configuration steps are: 'If' is set to 'Heltec_Temp'; 'Is' is set to 'greater than or equal to' with a 'Comparison Value or Feed' of '40'; 'Then' is set to 'email me'. Below this, a dropdown for 'Heltec_Temp' is followed by a 'value and time.' button. The 'Subject' field contains the text: 'Alerta el valor de la Temperatura tiene un valor anormal que es de: {{value}} °C'. The 'Body' field contains: 'Alerta la temperatura ambiental supera los parámetros normales con un valor actual de {{value}} °C con fecha {{created_at}}'.

Figura 3.62 Configuración de alerta para la temperatura ambiental

Finalmente, en la Figura 3.63 se ilustra la configuración para enviar una alerta al correo electrónico para valores anormales en la humedad del suelo. Esta alerta se envía al instante que la humedad del suelo medida por el sensor de humedad del suelo sea menor o igual a 10 (%), puesto que este valor es anormal en un invernadero y me indica que el suelo está totalmente seco [34].

The screenshot shows the 'Set up your action' interface for soil humidity. The sidebar shows 'Setup' (2) is the active step. The main area is titled 'Set up your action' with a 'Help' button. A yellow notification bar states: 'Your timezone is currently set to America/Lima. This can be modified in your account profile.' The configuration steps are: 'If' is set to 'Heltec_HumSuelo'; 'Is' is set to 'less than or equal to' with a 'Comparison Value or Feed' of '10'; 'Then' is set to 'email me'. Below this, a dropdown for 'Heltec_HumSuelo' is followed by a 'value and time.' button. The 'Subject' field contains: 'Alerta! el suelo esta totalmente seco con un valor de: {{value}} %'. The 'Body' field contains: 'Alerta! el suelo esta seco con un valor de {{value}} %, con fecha: {{created_at}} por favor activa el sistema de irrigación.'.

Figura 3.63 Configuración de alerta para la humedad del suelo

Finalmente, en la Figura 3.64 se ilustran todas las acciones configuradas para este proyecto. Además, de la acción configurada en la Figura 3.60 se configuro otras acciones que permitirán enviar distintos correos electrónicos conforme a los parámetros

configurados en el panel de control o cuando se tenga valores anormales. Se configuró varias alertas como por ejemplo cuando los valores de temperatura ambiental superen los 40 (°C), también se configuro una alerta cuando los valores de humedad de suelo sean menores a 10 (%).

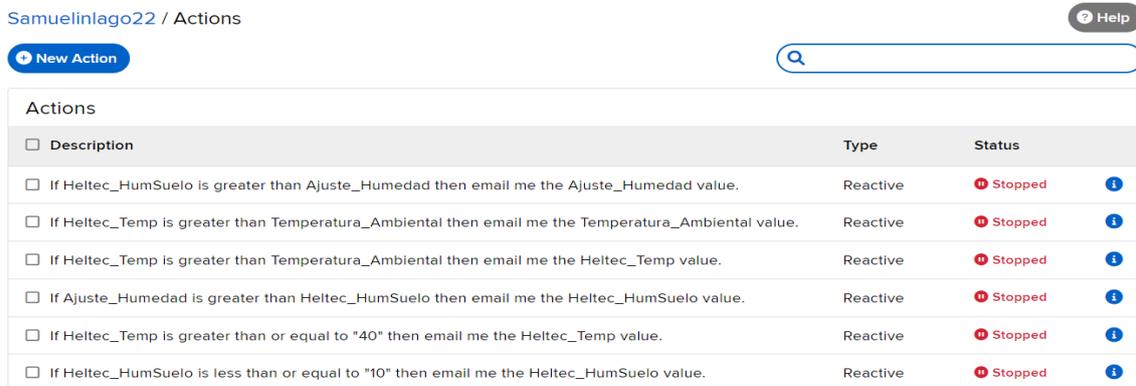


Figura 3.64 Acciones configuradas en Adafruit IO

3.4 Implementación del prototipo para control de invernadero

Implementación del código en el SoC maestro y el SoC esclavo

La implementación de los códigos para el SoC maestro y el SoC esclavo se llevó a cabo utilizando *Arduino IDE*. Para ello, primero se abrió la aplicación *Arduino IDE* en el ordenador. Luego, se cargaron los dos códigos previamente desarrollados en dos pestañas diferentes.

Para el SoC maestro, se seleccionó la opción *Tools* en la barra de menú, luego *Board*, y se eligió nuestro dispositivo, que es un *Heltec WiFi LoRa 32 (V3)*. A continuación, en *Tools*, se fue a la sección *Port* y se seleccionó *COM4*. Finalmente, se procede a cargar el código para el SoC esclavo como se puede visualizar en la Figura 3.65.

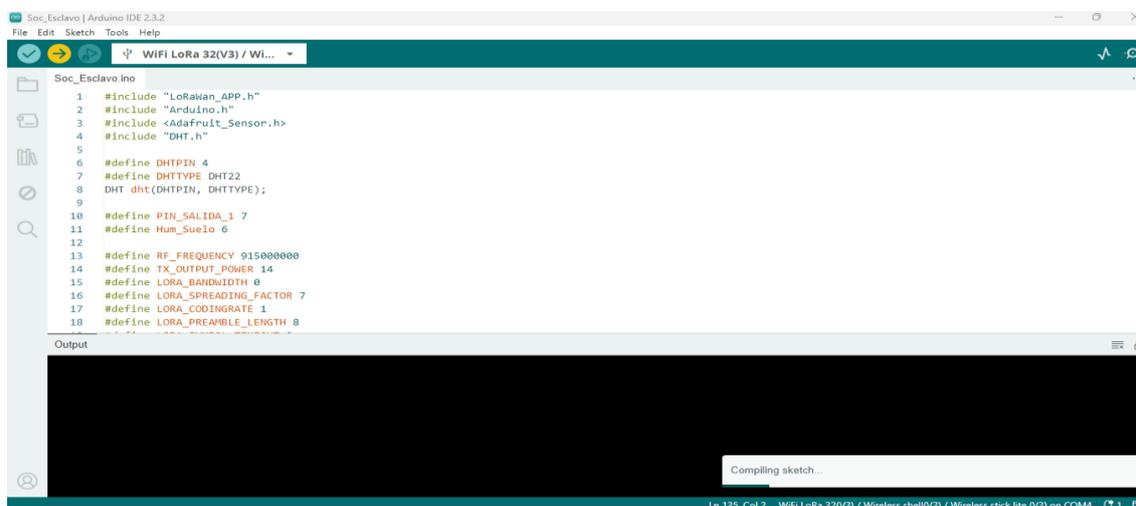
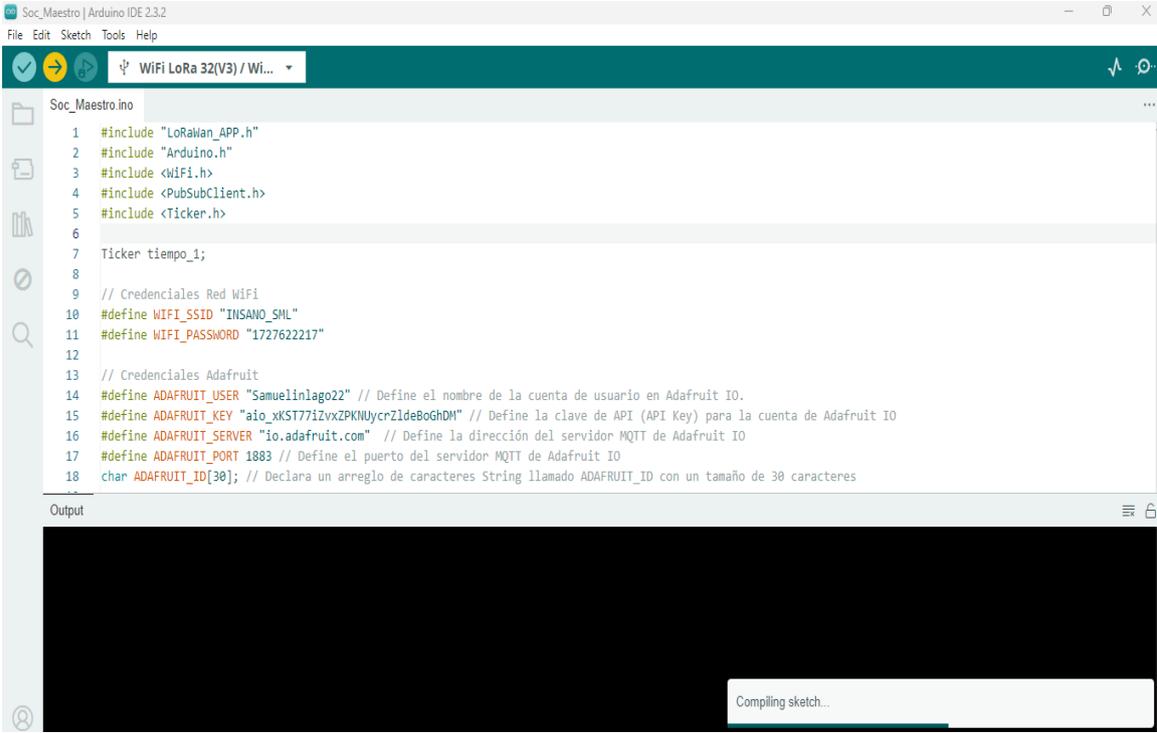


Figura 3.65 Compilación del código en el SoC esclavo

Para el SoC esclavo, se siguieron los mismos pasos, con la única diferencia de que en este caso se seleccionó el puerto *COM6*, y se procede a compilar el código como se ilustra en la Figura 3.66.



```
1 #include "LoRaWAN_APP.h"
2 #include "Arduino.h"
3 #include <WiFi.h>
4 #include <PubSubClient.h>
5 #include <Ticker.h>
6
7 Ticker tiempo_1;
8
9 // Credenciales Red WiFi
10 #define WIFI_SSID "INSANO_SML"
11 #define WIFI_PASSWORD "1727622217"
12
13 // Credenciales Adafruit
14 #define ADAFRUIT_USER "Samuelinlago22" // Define el nombre de la cuenta de usuario en Adafruit IO.
15 #define ADAFRUIT_KEY "aio_xKST77iZvxZPKNJycrZldeBoGhDM" // Define la clave de API (API Key) para la cuenta de Adafruit IO
16 #define ADAFRUIT_SERVER "io.adafruit.com" // Define la dirección del servidor MQTT de Adafruit IO
17 #define ADAFRUIT_PORT 1883 // Define el puerto del servidor MQTT de Adafruit IO
18 char ADAFRUIT_ID[30]; // Declara un arreglo de caracteres String llamado ADAFRUIT_ID con un tamaño de 30 caracteres
```

Figura 3.66 Compilación del código en el SoC maestro

Implementación del circuito electrónico

Se diseñó un esquema detallado para la conexión entre el microcontrolador esclavo y los *headers* hembra, como se muestra en la Figura 3.67. Este diseño se elaboró considerando las necesidades específicas del proyecto. Los sensores y el relé se conectan a los *headers* hembra mediante conectores macho, lo que permite su alimentación y la transmisión de datos a través de los pines *GPIO* del microcontrolador.

Además, es importante mencionar que los sensores como el *DHT22* y el sensor de humedad del suelo *FC-28* no fueron incluidos en este circuito, ya que deben colocarse en la maqueta para medir los parámetros ambientales. La conexión de los sensores y el relé se realiza mediante cables con conectores macho a los *headers* hembra ubicados en el circuito. Esta configuración no solo garantiza una operación eficiente, sino también segura, al asegurar conexiones firmes y estables. Los *headers* hembra permiten una conexión modular y organizada, facilitando tanto el mantenimiento como la expansión del sistema para futuros sensores, lo cual es crucial para la integridad y funcionalidad del proyecto.

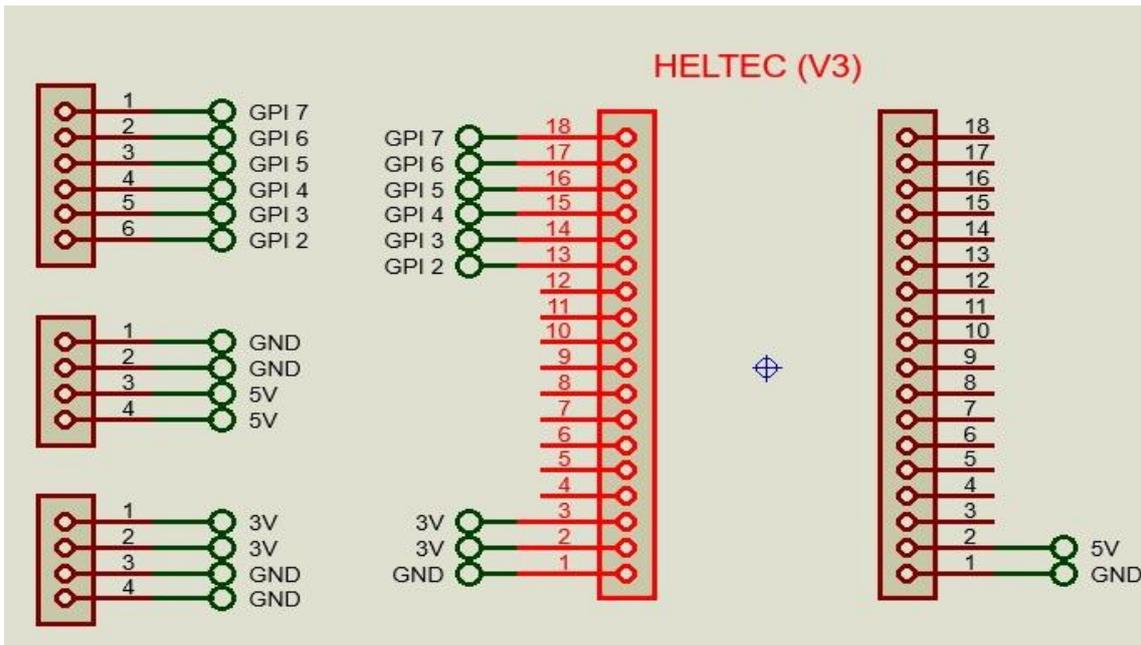


Figura 3.67 Diseño del circuito electrónico

También, para garantizar un funcionamiento óptimo de todas las conexiones electrónicas en el prototipo, es crucial diseñar y fabricar un *PCB* (*Printed Circuit Board* o placa de circuito impreso). El *PCB* facilita la interconexión eficiente entre el microcontrolador y los *headers*. Además, por medio de los *headers* se facilita la conexión con los sensores y el relé proporcionando una plataforma organizada y confiable para el montaje y la integración de componentes. Además de simplificar las conexiones, el *PCB* mejora la fiabilidad del sistema al reducir los errores de cableado.

A continuación, se elaboró el diagrama del *PCB* previo a su implementación como visualiza en la Figura 3.68.

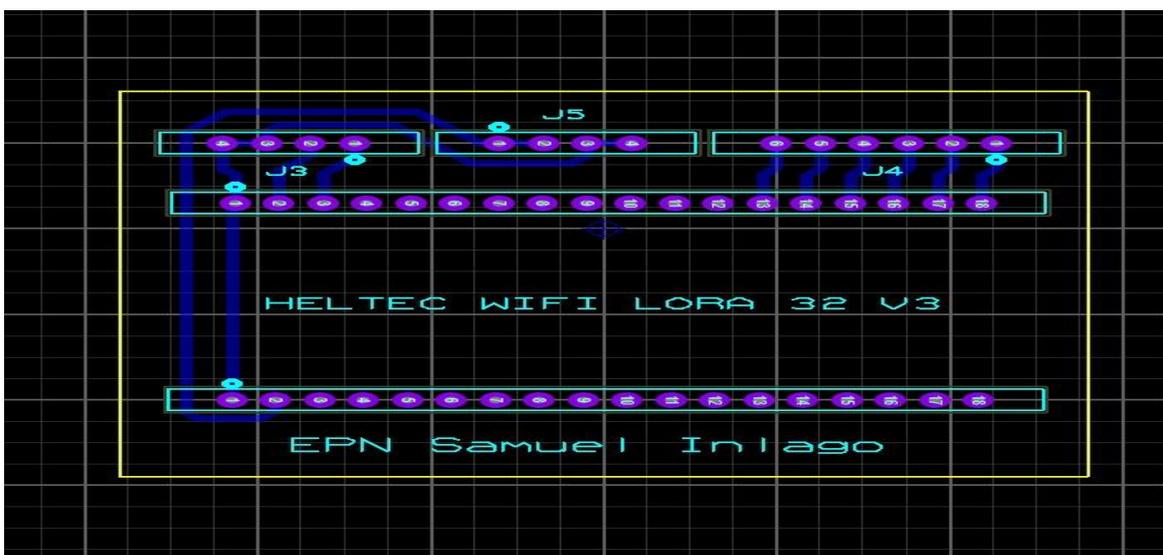


Figura 3.68 Diseño del PCB

En la Tabla 3.5, se muestra las conexiones electrónicas en el *PCB* entre los pines del SoC esclavo *Heltec WiFi LoRa 32 (V3)* y los *headers* J3, J4 y J5. Es importante mencionar que los sensores y el relé no se incluyen en el circuito electrónico del *PCB* puesto que estos van colocados dentro el prototipo de invernadero para medir parámetros de humedad del suelo, temperatura y humedad ambiental y estos se conectan por medio de cables a los *headers* del *PCB*.

Tabla 3.5 Conexiones electrónicas entre el SoC esclavo y los headers [6]

Dispositivo electrónico	Pin	Identificación	Conexión
Heltec WiFi LoRa 32 (V3)	1	GND	Conectado al <i>header</i> J3 y J5
	2	5V	Conectado al pin 1 y 2 del <i>header</i> J5
	1	GND	Conectado al pin 3 y 4 del <i>header</i> J3 y J5
	2	3V3	Conectado al pin 2 del <i>header</i> J3
	3	3V3	Conectado al pin 1 del <i>header</i> J3
	18	GPI07	Conectado al pin 1 del <i>header</i> J4
	17	GPI06	Conectado al pin 2 del <i>header</i> J4
	15	GPI04	Conectado al pin 4 del <i>header</i> J4

Luego, de completar el diseño del *PCB* y detallar la asignación de los pines en la Tabla 3.5 del SoC esclavo y los *headers*, se procedió con la fabricación del *PCB* sobre la baquelita, empleando el método del planchado. Se selecciono la baquelita con dimensiones de 6 (cm) de longitud y de ancho 5 (cm) como se ilustra en la Figura 3.69.



Figura 3.69 Baquelita

Además, se usó otros elementos como es el cloruro férrico que se visualiza en la Figura 3.70 y el circuito impreso para la elaboración del *PCB*.



Figura 3.70 Cloruro férrico

Después de utilizar todos los elementos mencionados anteriormente para desarrollar el *PCB* y obtener el resultado esperado, se procedió a realizar los orificios necesarios para el microcontrolador y los *headers*. Posteriormente, se soldaron los distintos componentes al *PCB*. Como resultado, se obtuvo la placa de circuito electrónico completamente funcional, preparada para su integración en el proyecto como se muestra en la Figura 3.71 y Figura 3.72.

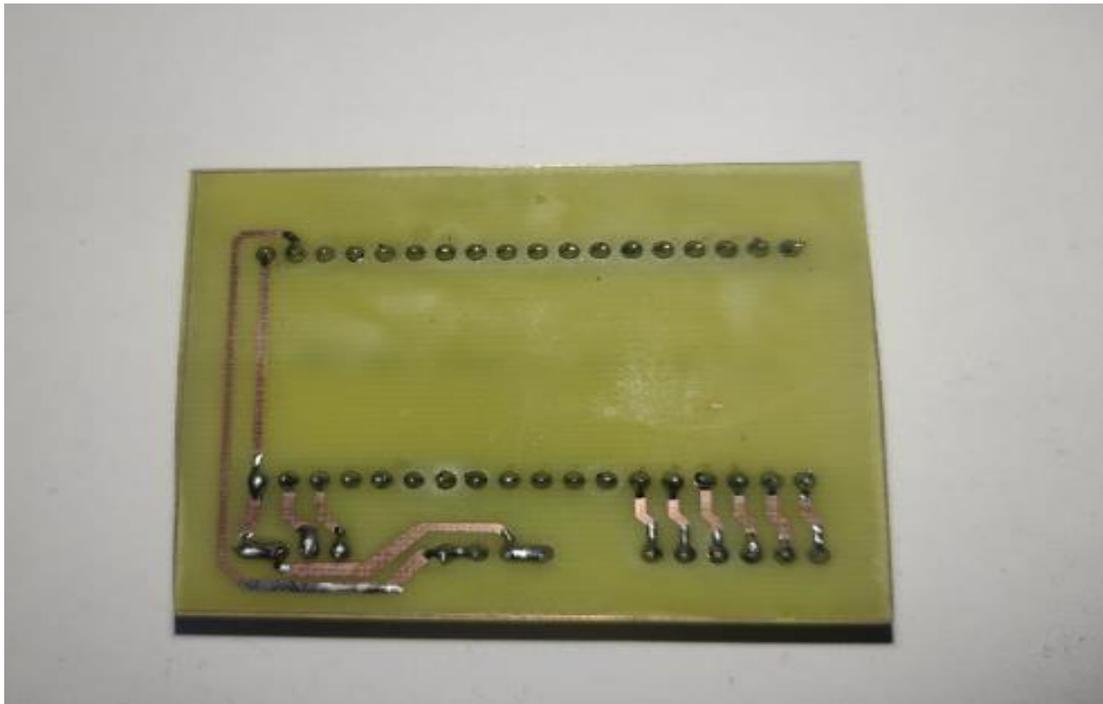


Figura 3.71 Soldadura de elementos en el PCB

En la Figura 3.72 se puede apreciar el *PCB* con los *headers* listos para conectar los sensores y el relé con cables machos.



Figura 3.72 *PCB*

Finalmente, en la Figura 3.73 se aprecia al *PCB* con el SoC esclavo colocado en su caja protección para evitar cualquier daño o manipulación de este dispositivo.

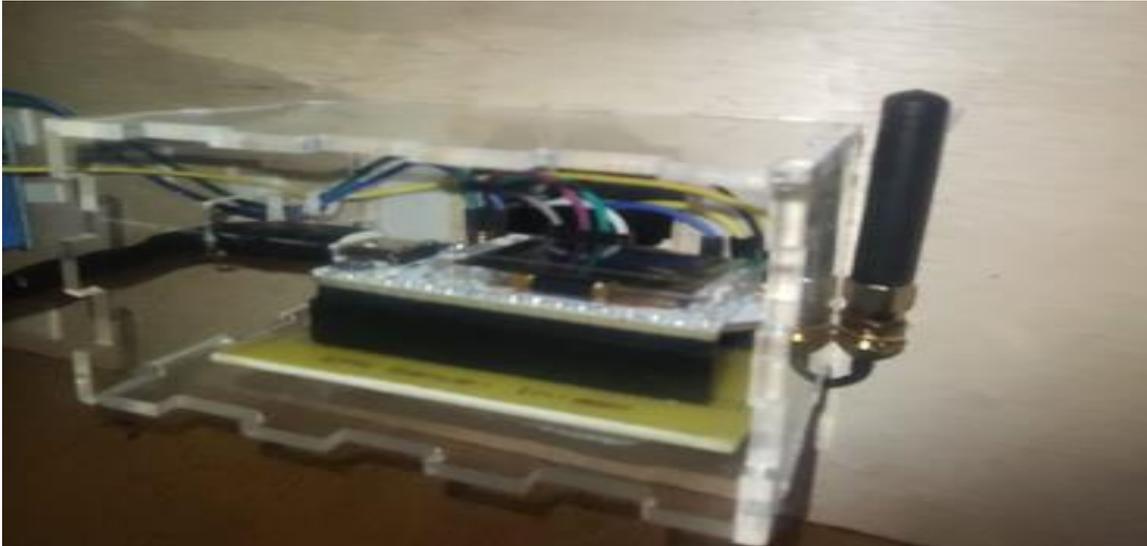


Figura 3.73 Montaje del PCB a la caja de protección

Implementación de protección al *PCB* y *SoC* esclavo

Además del desarrollo del *PCB* realizado anteriormente, es crucial implementar elementos de protección para salvaguardar tanto el *PCB* como los microcontroladores esclavo y maestro. Estos componentes electrónicos son delicados y susceptibles a daños, por lo que no deben estar expuestos sin protección adecuada.

Se diseñaron y fabricaron carcasas de protección específicas para cada microcontrolador. Para el *SoC* esclavo, se utilizó material acrílico, mientras que para el microcontrolador maestro se optó por una carcasa *MDF*. En la Figura 3.74 se aprecia el diseño de protección para el *SoC* esclavo y *PCB*

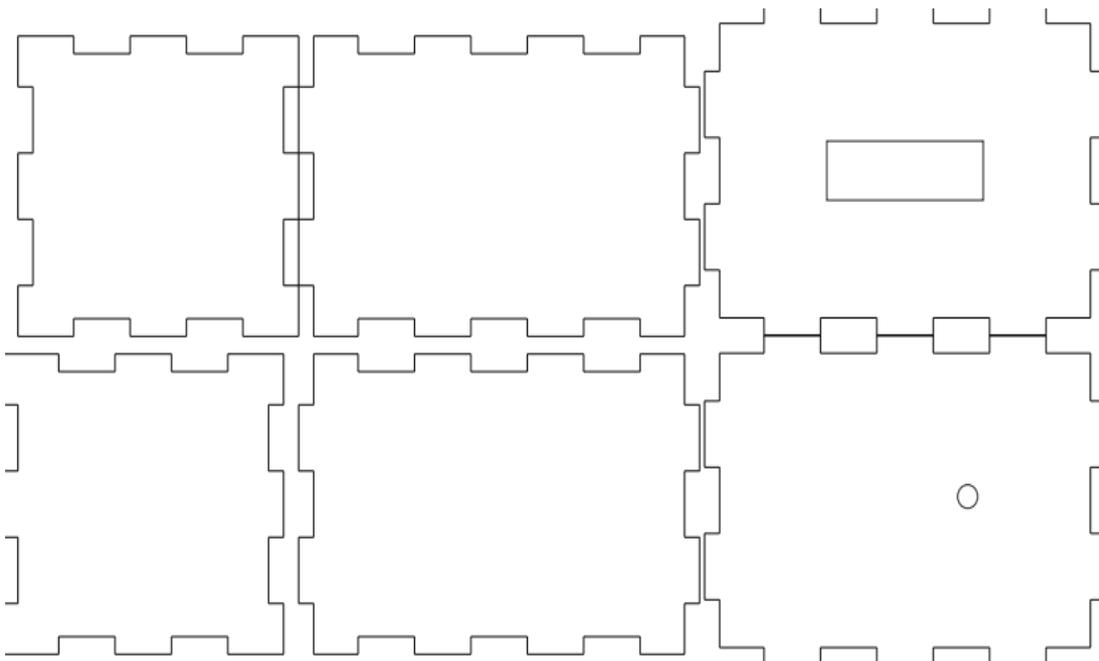


Figura 3.74 Diseño de protección para el PCB y *SoC* esclavo

Estas carcasas no solo proporcionan una barrera física contra impactos o líquidos, sino que también contribuyen a la estabilidad y durabilidad del sistema en su conjunto. A continuación, en la Figura 3.75 se aprecia el diseño para la protección del SoC maestro.

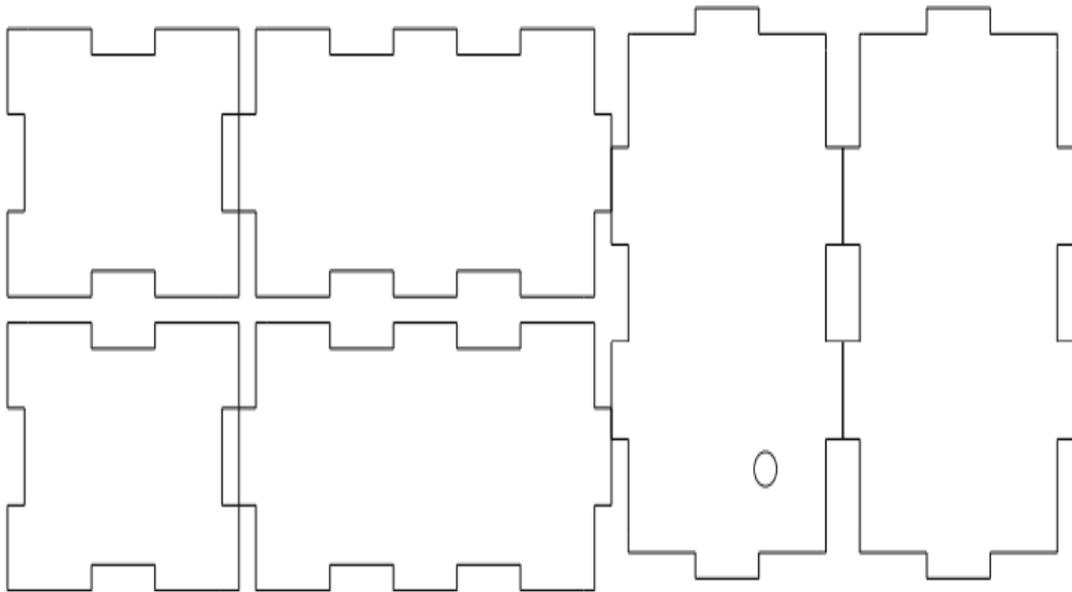


Figura 3.75 Diseño de protección para el SoC maestro

Luego de desarrollar el diseño se procedió a fabricar las dos protecciones para los dos dispositivos y el *PCB*. En la Figura 3.76 se aprecia la protección para el SoC esclavo y *PCB*.

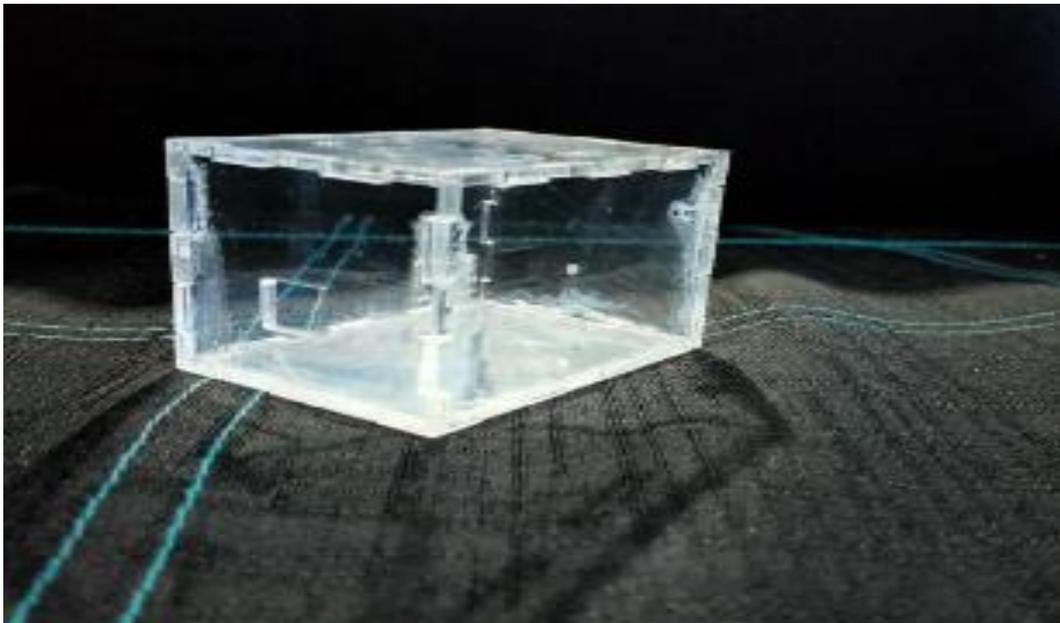


Figura 3.76 Protección para el SoC esclavo y *PCB*

A continuación, en la Figura 3.77 se puede apreciar la protección para el SoC maestro totalmente listo para su implementación.



Figura 3.77 Protección para el SoC maestro

Implementación de la maqueta

En esta sección de implementación de la maqueta para un prototipo de invernadero, se utilizaron diversos materiales para lograr una simulación efectiva. Se emplearon tablas de tríplex de pequeño tamaño y columnas de 1 (cm) de lado, así como plástico para crear y replicar la estructura de un invernadero. Las dimensiones específicas de este prototipo son 40 (cm) de longitud, 30 (cm) de ancho y 30 (cm) de altura.

El uso de estos materiales permite una construcción estable, mientras que el plástico proporciona una cubierta adecuada para simular el ambiente controlado de un invernadero. Esta maqueta no solo facilita la visualización del diseño, sino que también permite probar y ajustar los sistemas electrónicos como los sensores para este proyecto. Luego de obtener todos los elementos con sus medidas exactas como se aprecia en la Figura 3.78 se procedió a pegar todos los elementos para construir la maqueta.



Figura 3.78 Elementos para la maqueta

En la Figura 3.79 se puede apreciar el proceso para la construcción de esta maqueta implementado todos sus elementos.



Figura 3.79 Proceso de construcción de la maqueta

A continuación, en la Figura 3.80 se puede observar el resultado de la construcción de toda la maqueta para el prototipo de un invernadero y con todos sus elementos. En esta maqueta incluyen elementos como son la protección del microcontrolador esclavo en el respectivo *PCB*, el sensor *DHT22* para medir la temperatura y humedad ambientales, también incluye en el sensor de humedad del suelo *FC-28* y el respectivo relé para activar el sistema de irrigación.



Figura 3.80 Maqueta para el prototipo de invernadero

Implementación del panel de Control

Para la ejecución de esta sección, el panel de control ya ha sido configurado previamente para monitorear los datos de los sensores en *Adafruit IO*. Lo único que se debe hacer es ingresar a nuestro panel de control, denominado "Control de

Invernadero". Si todo el sistema está funcionando correctamente, el panel de control mostrará todos los datos de manera automática.

En el panel de control incluyen bloques para monitorear la temperatura ambiental, los cambios en la humedad ambiental y las variaciones en la humedad del suelo. Además, se ha configurado un bloque para el sistema de irrigación, que permite activar el relé para encender la mini bomba de agua seleccionando la opción *ON*. De igual manera, al seleccionar la opción *OFF*, el relé se apagará, desactivando la mini bomba y, en consecuencia, el sistema de irrigación. También se tiene dos bloques que tienen como función el ajuste de parámetros, denominados "Ajuste de Humedad de suelo" y "Ajuste de temperatura Ambiental".

Finalmente, en la Figura 3.81 se puede apreciar el panel de control completo con todos los bloques y sus respectivas aplicaciones. Este panel de control es de gran ayuda, ya que facilita la integración necesaria para monitorear y controlar el sistema de irrigación del prototipo de invernadero en este proyecto.

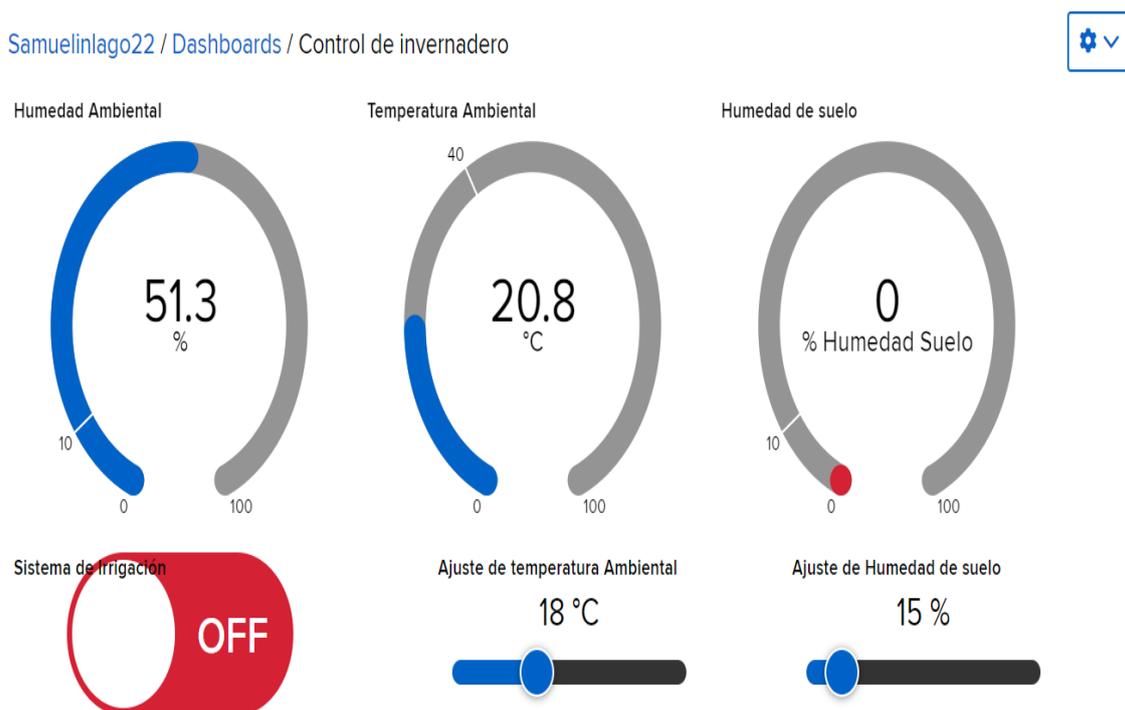


Figura 3.81 Panel de control en Adafuit IO

Implementación de alertas en *actions* en *Adafuit IO*

Para la implementación de las alertas en esta sección, el primer paso es acceder a *Adafuit IO* y luego a la sección de *actions*. A continuación, se inicia todas las alertas configuradas previamente como se aprecia en la Figura 3.82. Es importante mencionar que estas alertas se encargan de enviar notificaciones conforme a los parámetros

establecidos en el panel de control de *Adafruit IO* y cuando se tenga valores anormales del prototipo de invernadero como en la temperatura ambiental y humedad del suelo.

Samuelinlago22 / Actions ? Help

New Action

Actions	Type	Status
<input type="checkbox"/> Description		
<input type="checkbox"/> If Heltec_HumSuelo is greater than Ajuste_Humedad then email me the Ajuste_Humedad value.	Reactive	Active i
<input type="checkbox"/> If Heltec_Temp is greater than Temperatura_Ambiental then email me the Temperatura_Ambiental value.	Reactive	Active i
<input type="checkbox"/> If Heltec_Temp is greater than Temperatura_Ambiental then email me the Heltec_Temp value.	Reactive	Active i
<input type="checkbox"/> If Ajuste_Humedad is greater than Heltec_HumSuelo then email me the Heltec_HumSuelo value.	Reactive	Active i
<input type="checkbox"/> If Heltec_Temp is greater than or equal to "40" then email me the Heltec_Temp value.	Reactive	Active i
<input type="checkbox"/> If Heltec_HumSuelo is less than or equal to "10" then email me the Heltec_HumSuelo value.	Reactive	Active i

Figura 3.82 Implementación de las alertas en *actions* en *Adafruit IO*

3.5 Realizar las pruebas de funcionamiento

Después de completar las secciones de identificación de requisitos, selección de *hardware* y *software*, así como el diseño e implementación del prototipo de invernadero, se realizó las pruebas de funcionamiento del prototipo en una maqueta de invernadero. En esta etapa, se verificaron los cumplimientos, requerimientos y alcances de este proyecto.

Inicialización de todos los componentes del proyecto

En esta etapa, se procede a energizar todos los componentes del sistema. Los dispositivos *SoC*, tanto esclavo como maestro, permiten que se comuniquen entre sí a través de la tecnología *LoRa*. El *SoC* esclavo comienza a leer los datos del sensor *DHT22* y sensor de humedad del suelo, enviando esta información mediante *LoRa*. Por su parte, el dispositivo *SoC* maestro establece conexión con la red *WiFi* y se conecta a *Adafruit IO* a través del protocolo *MQTT*. Además, el dispositivo maestro recibe la información enviada por *LoRa* desde el esclavo y la transmite a los *feeds* previamente configurados en *Adafruit IO*.

En la Figura 3.83 se puede apreciar al *SoC* maestro en su caja de protección y además la conexión a las fuentes de alimentación en las siguientes figuras.



Figura 3.83 Energización del SoC esclavo, sensores y relé

En la Figura 3.84 se puede apreciar las fuentes de alimentación tanto para el SoC esclavo y para la mini bomba de agua que se activa por medio del relé que será activado o desactivado por el microcontrolador.



Figura 3.84 Suministro de energía para el SoC esclavo y la mini bomba

En la Figura 3.85 se puede apreciar que se conecta el SoC maestro a la fuente de alimentación para iniciar con su funcionamiento.



Figura 3.85 Suministro de energía para el SoC maestro

En la Figura 3.86 se ilustra la mini bomba de agua conectada a una manguera pequeña de 2 (m) de longitud, permitiendo el inicio del funcionamiento del sistema de irrigación.



Figura 3.86 Mini bomba de agua para el sistema de irrigación conectado a una manguera pequeña

A continuación, en la Figura 3.87 se puede visualizar todo el sistema del prototipo de invernadero, incluyendo el sensor de humedad del suelo y el sensor de temperatura y humedad ambiental. Además, se aprecia el sistema de irrigación, compuesto por una manguera larga con orificios, lo que permite una irrigación en el prototipo.



Figura 3.87 Prototipo de invernadero a monitorear

A continuación, se ingresa a la cuenta de *Adafruit IO* utilizando el nombre de usuario y la contraseña previamente establecidos. Una vez dentro, se accede al panel de control para verificar que la información enviada desde los sensores se esté recibiendo correctamente en los *feeds*, y que estos, a su vez, actualicen los bloques del panel de control que se ilustra en la Figura 3.88.

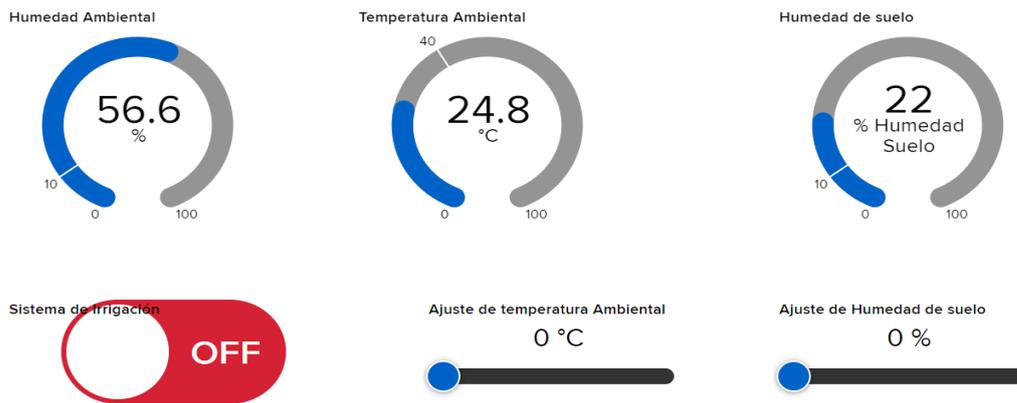


Figura 3.88 Monitoreo del prototipo de invernadero en el panel de control

Prueba de monitoreo de temperatura ambiental y humedad del suelo en el prototipo de invernadero

A continuación, se verifica tanto en la maqueta como en el panel de control las mediciones de los parámetros ambientales y del suelo. Se procede a visualizar los cambios de temperatura y humedad a lo largo del tiempo. Además, se utiliza una pistola de calor para alterar estos parámetros en el sistema.

En la Figura 3.89 se pueden visualizar los datos obtenidos después del inicio del sistema: la temperatura ambiental es de 22.2 (°C), la humedad ambiental es del 48.4 (%), y la humedad del suelo es del 20 (%).

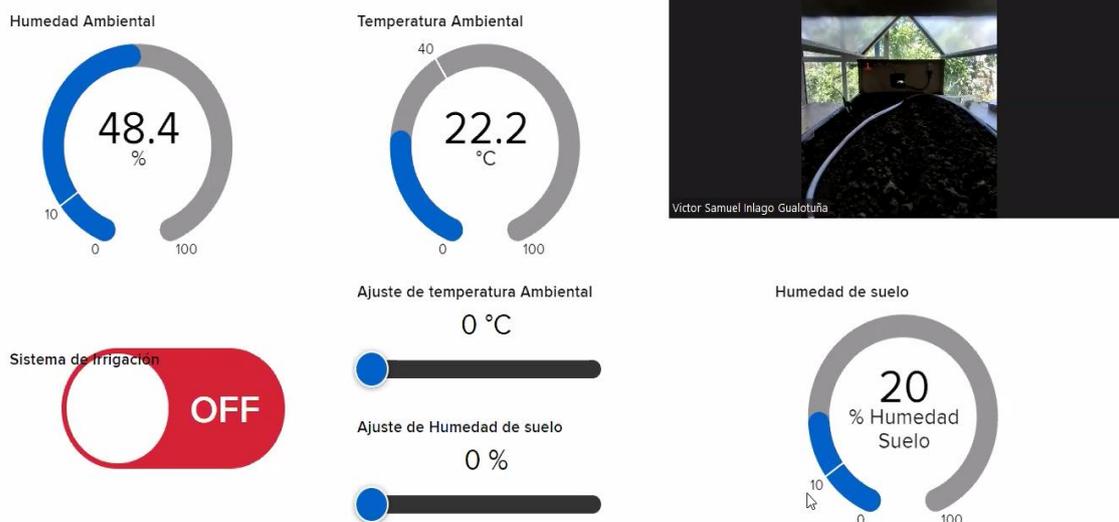


Figura 3.89 Monitoreo de parámetros ambientales sin la pistola de calor

Luego, se utiliza la pistola de calor para modificar los parámetros. Esta herramienta permite cambiar la temperatura y la humedad ambiental, aunque se requiere un corto tiempo para que los sensores registren los cambios y envíen los datos a través de *LoRa* al SoC maestro, que posteriormente los sube a *Adafruit IO* mediante *WiFi*. En la Figura 3.90, se puede apreciar un leve incremento y disminución en los valores.

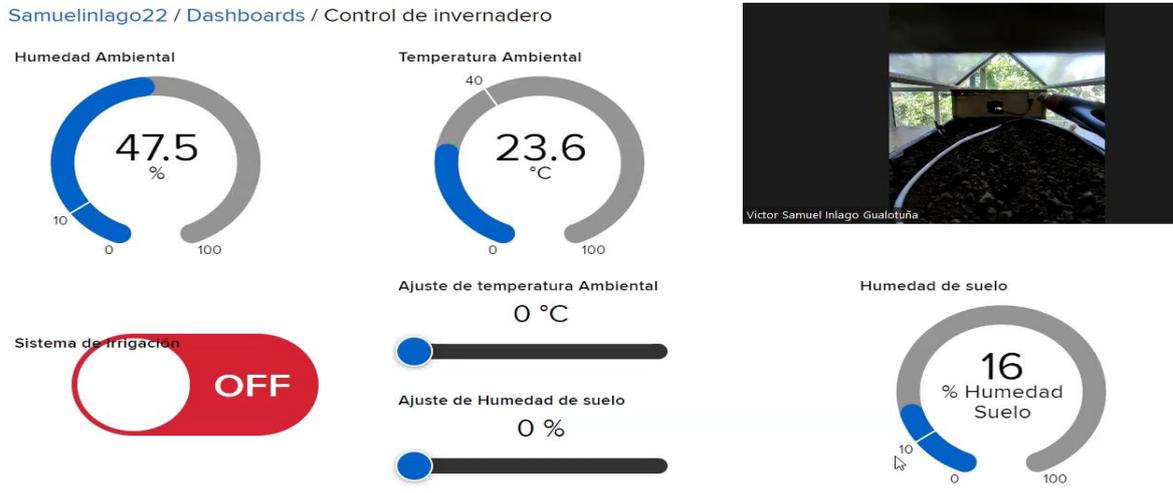


Figura 3.90 Utilización de una pistola de calor para cambiar valores de los parámetros ambientales

Finalmente, después de unos instantes, en la Figura 3.91 se puede apreciar que los cambios en la temperatura ambiental son muy visibles, alcanzando un valor de 42.2 (°C). La humedad ambiental se registra en 29 (%) y la humedad del suelo en 14 (%). Esta prueba confirma el correcto funcionamiento de los sensores en el prototipo de invernadero.

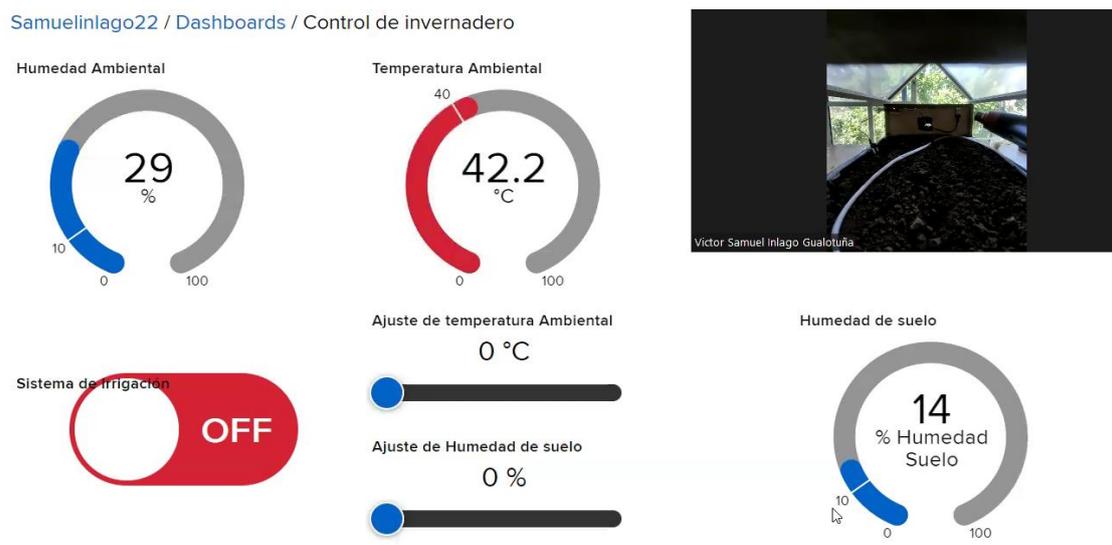


Figura 3.91 Funcionamiento de los sensores

Prueba de control del sistema de irrigación

Para realizar la prueba del sistema de irrigación, se accede al panel de control en *Adafruit IO* y se activa el bloque poniéndolo en estado *ON*. En la Figura 3.92, se puede visualizar cómo se inicia el sistema de irrigación colocando en *ON*.

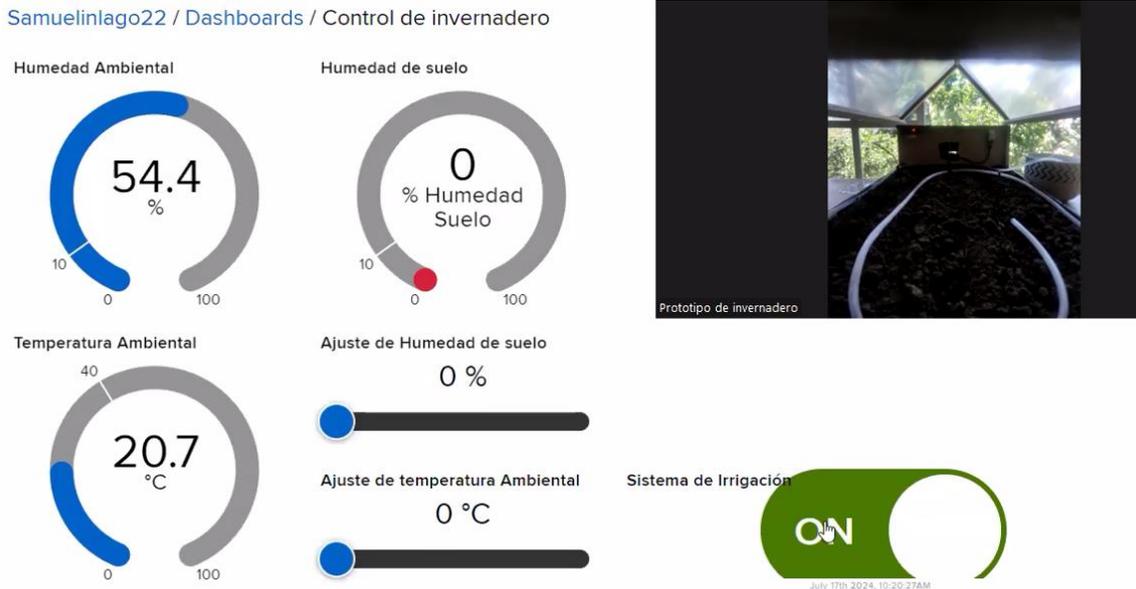


Figura 3.92 Activación del sistema de irrigación en el prototipo

A continuación, se verifica en el prototipo la activación de la mini bomba, que a su vez pone en funcionamiento el sistema de riego mediante una pequeña manguera transparente, como se ilustra en la Figura 3.93.



Figura 3.93 Verificación del sistema de irrigación en el prototipo

Luego, se procede a desactivar el sistema de irrigación colocando en *OFF* el bloque del sistema de irrigación en el panel de control. Luego, se observa que la mini bomba deja

detiene su funcionamiento, desactivando así el sistema de irrigación como se aprecia en la Figura 3.94. Con esto se verifica la prueba de funcionamiento del sistema de irrigación en el prototipo por medio de un panel de control.



Figura 3.94 Colocando en estado OFF el sistema de irrigación

Prueba del ajuste de parámetros

El primer ajuste de parámetro que se va a realizar es el ajuste en la humedad del suelo. En este momento se puede observar que la humedad del suelo se encuentra en 0 (%), lo que implica un suelo totalmente seco como se ilustra en la Figura 3.95.

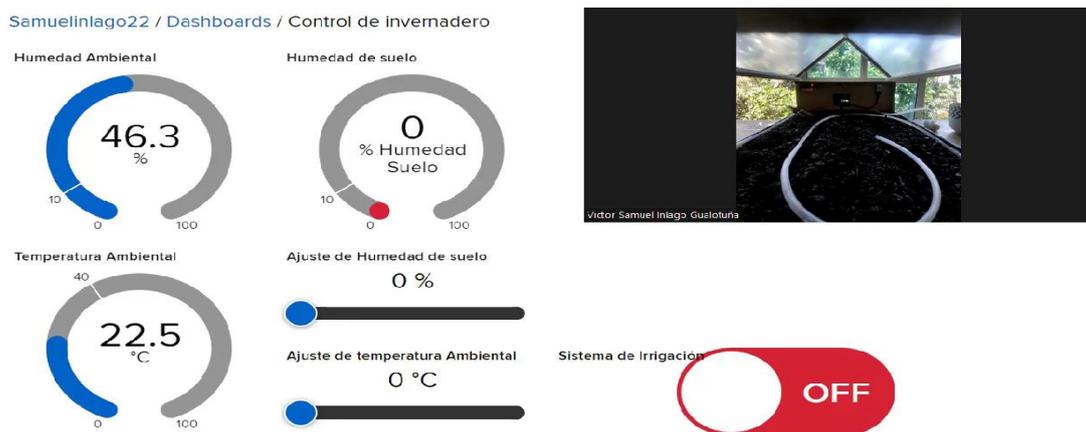


Figura 3.95 Prueba de ajuste de parámetro de humedad de suelo

A continuación, se procede a colocar en el bloque de ajuste de parámetro de humedad del suelo en 25 (%), como se ilustra en la Figura 3.96.



Figura 3.96 Parámetro del suelo ajustado a un 25 (%)

Luego, como consecuencia de esta acción, el sistema de irrigación se activa automáticamente hasta alcanzar un valor igual o superior al 25 (%) de humedad del suelo, como se aprecia en la Figura 3.97.



Figura 3.97 Encendido del sistema de irrigación automático

Finalmente, el sistema de irrigación se desactiva y deja de fluir el agua cuando el valor de humedad del suelo leído por el sensor sea mayor al 25 o igual al 25 (%). Como se puede visualizar en la Figura 3.98 y también en el panel de control.

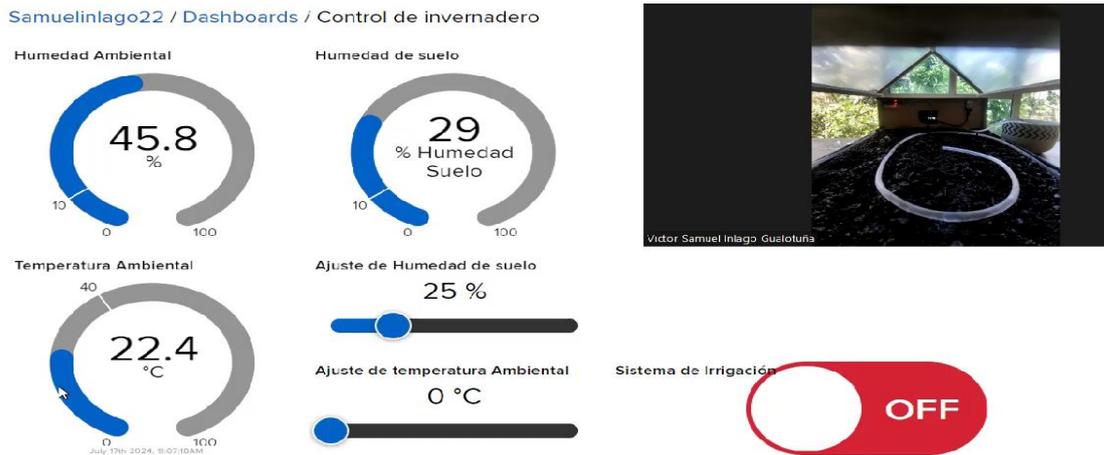


Figura 3.98 Apagado del sistema de irrigación automático

El segundo ajuste que se realizará es el de la temperatura ambiental. Es importante mencionar que el ajuste de la temperatura ambiental no se puede modificar desde los sensores en el prototipo por medio del panel de control en *Adafruit IO*, a diferencia de la humedad del suelo, que se puede ajustar mediante el sistema de irrigación.

Lo que se va a realizar es comparar el valor del bloque de ajuste de temperatura con el valor del bloque de temperatura ambiental. Si el valor de la temperatura ambiental es mayor al del parámetro establecido en el ajuste de temperatura, se procederá a enviar un correo electrónico a Gmail alertando esta acción. Asimismo, se enviará un correo con el valor de la temperatura ambiental actual.

A continuación, en la figura se puede observar que la temperatura ambiental tiene un valor de 20.2 (°C) y el ajuste de temperatura se estableció en un valor del 16 (°C) como se ilustra en la Figura 3.99.



Figura 3.99 Ajuste de temperatura ambiental

A continuación, se accede al correo electrónico *Gmail* y se verifica que llegó dos correos electrónicos desde *Adafruit IO* como se ilustra en la Figura 3.100.



Figura 3.100 Alerta en el correo electrónico

Luego, se abre el primer correo electrónico que llegó y se muestra que el valor de la temperatura ambiental sobrepasa el parámetro que se estableció en un 16 (°C), como se ilustra en la Figura 3.101.

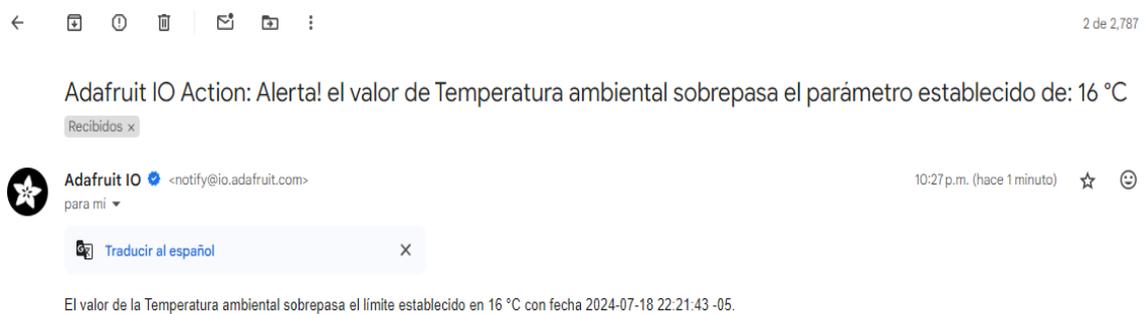


Figura 3.101 Correo electrónico de ajuste de temperatura ambiental

Finalmente, abrimos el segundo correo electrónico que indica el valor de la temperatura ambiental actual como se aprecia en la Figura 3.102. Con esto queda comprobado el funcionamiento de ajuste de parámetros.

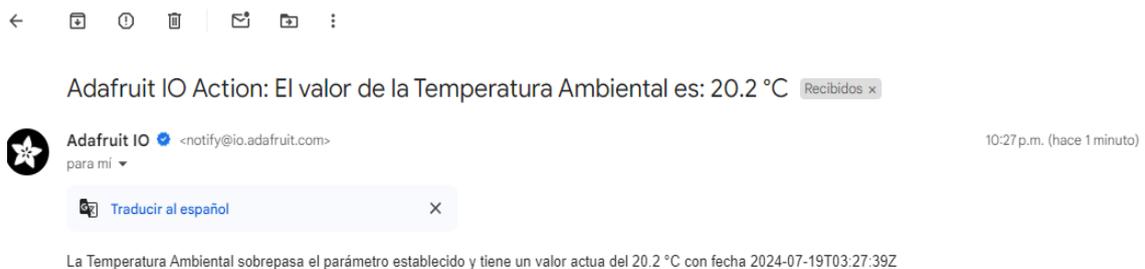


Figura 3.102 Correo electrónico con el valor de temperatura ambiental

Prueba del sistema de alerta para condiciones anormales en el prototipo de invernadero por medio de Acciones en Adafruit IO.

Para realizar esta prueba, es crucial mencionar que previamente se configuró las acciones en Adafruit IO para recibir alertas en situaciones específicas: cuando los valores de humedad del suelo sean menores o iguales al 10 (%), lo cual indica un suelo completamente seco y requiere activar el sistema de irrigación, y cuando la temperatura

ambiental supere los 40 (°C), ya que este valor es anormalmente alto y puede representar un peligro para un invernadero.

A continuación, en la Figura 3.103 se puede apreciar que la temperatura ambiental tiene un valor mayor a los 40 (°C), y la humedad del suelo tiene un valor menor al 10 (%) lo que indica que son valores anormales en el prototipo.

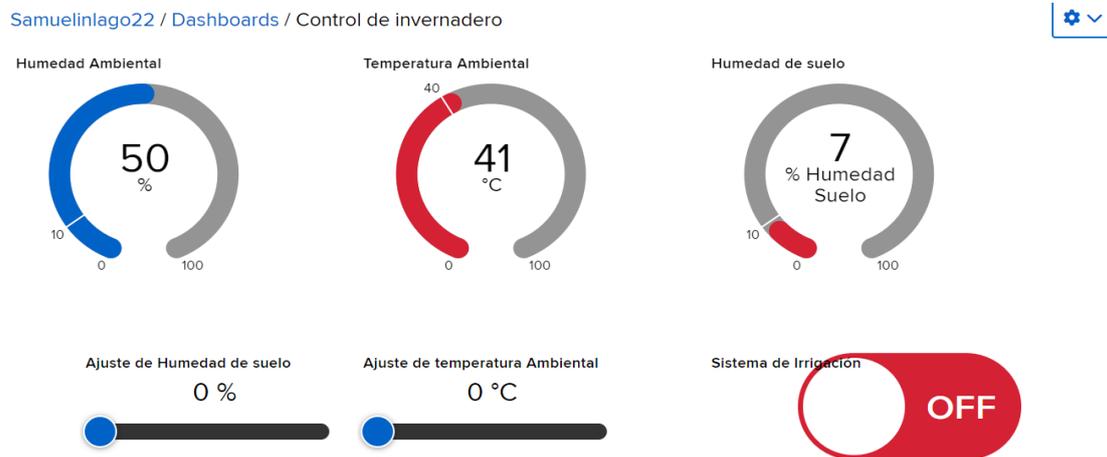


Figura 3.103 Condiciones anormales en el prototipo

A continuación, se envían automáticamente dos correos a Gmail alertando sobre estas condiciones anormales, que se visualiza en la Figura 3.104.



Figura 3.104 Correos en Gmail alertando condiciones anormales

Resumen de las pruebas de funcionamiento

Luego de realizar las pruebas de funcionamiento del prototipo con SoC para el control de invernadero mediante comunicación *LoRa*, se comprobó el monitoreo de la temperatura ambiental y la humedad del suelo en el invernadero a través del panel de control *web*. Además, se verificó el control del sistema de irrigación desde el panel de control de *Adafruit IO*, simplemente activando o desactivando el bloque correspondiente.

Asimismo, se comprobó el ajuste automático de parámetros: al establecer un valor en el bloque de ajuste de humedad del suelo, el sistema de irrigación se activó automáticamente hasta alcanzar o superar el valor establecido, momento en el cual se apagó automáticamente.

También se validó el sistema de alertas para condiciones anormales. En este caso, se verificaron alertas automáticas cuando la temperatura ambiental y la humedad del suelo presentaban valores fuera de los rangos normales. Además, se enviaron alertas al realizar ajustes en los parámetros, garantizando una supervisión continua y eficiente del invernadero.

Costo del prototipo

En la Tabla 3.6 se ilustra el costo de todos los elementos necesarios para el prototipo de control de invernadero. Los precios están basados en los listados de la página de Mercado Libre.

Tabla 3.6 Precio de todos los elementos del prototipo

Elemento	Cantidad	Precio unitario en dólares americanos	Precio total en dólares americanos
Heltec WiFi LoRa 32 (V3)	2 (u)	30	60
Sensor DHT22	1 (u)	5	5
Sensor humedad de suelo FC-28	1 (u)	4	4
Relé	1 (u)	4	4
Mini bomba 5 (V)	1 (u)	5	5
Fuente de alimentación	2 (u)	6	12
Placa PCB	1 (u)	11	11
Caja de protección para SoC esclavo	1 (u)	10	10
Caja de protección para SoC maestro	1 (u)	7	7
Maqueta	1 (u)	15	15
Mano de obra	48 (h)	10	480
		Total	613

A continuación, se proporciona el enlace del video del funcionamiento del prototipo: [Samuel Inlago Tesis Prototipo invernadero.mp4](#)

4 CONCLUSIONES

En base a los objetivos propuestos y en base a la Implementación del proyecto, se llega a las siguientes conclusiones

- La implementación del prototipo con SoC para el control de un invernadero mediante comunicación *LoRa* se realizó exitosamente utilizando las plataformas *Adafruit IO* y *Arduino IDE*. Estas plataformas fueron fundamentales para la realización del proyecto. *Adafruit IO* permitió el monitoreo de los parámetros ambientales a través de su panel de control y facilitó la conexión bidireccional con el SoC maestro mediante *WiFi*. Por su parte, *Arduino IDE* proporcionó un entorno amigable para el desarrollo del código necesario para la comunicación *LoRa*, la comunicación *WiFi* y la lectura de los datos de los sensores.
- La elección de los dos dispositivos SoC *Heltec WiFi LoRa 32 (V3)* como componentes centrales permitió establecer una comunicación bidireccional *LoRa* entre los dos microcontroladores. Además, la tecnología *WiFi* integrada en estos dispositivos facilitó que el SoC maestro se conecte a una red *WiFi* y reenvíe la información recibida por *LoRa* desde el SoC esclavo a *Adafruit IO*, posibilitando así el monitoreo remoto del prototipo. Gracias a la integración de las tecnologías *WiFi* y *LoRa* en estos dispositivos, se eliminó la necesidad de agregar módulos adicionales y cableado extra. Los SoC funcionaron de manera correcta en el entorno de *Arduino IDE*, lo que permitió el desarrollo exitoso de este proyecto *IoT*.
- Después de realizar investigaciones previas, se desarrolló e implementó una solución utilizando la tecnología *LoRa*, conocida por su bajo consumo de energía y largo alcance, entre dos dispositivos *Heltec WiFi LoRa 32 (V3)*. Además, se construyó una maqueta del invernadero, demostrando la funcionalidad y eficiencia del sistema de control diseñado. Se simuló un entorno con parámetros de humedad y temperatura ambiental, así como la humedad del suelo, validando la capacidad del sistema para monitorear y gestionar las condiciones del invernadero de manera efectiva.
- Los códigos fueron creados en el entorno abierto de *Arduino IDE*, permitiendo realizar el control del invernadero y el establecimiento de la comunicación inalámbrica mediante *LoRa* entre dos dispositivos *Heltec WiFi LoRa 32 (V3)*. Uno de los SoC se encarga de leer los datos de humedad del suelo, temperatura y humedad ambiental, enviándolos a través de *LoRa* al segundo SoC, que luego transmite la información mediante *WiFi* hacia *Adafruit IO*. Finalmente, se creó un

dashboard web en la plataforma *IoT* seleccionada, *Adafruit IO*, permitiendo un monitoreo remoto y eficiente del sistema.

- Durante la fase de implementación, todos los componentes del proyecto funcionaron exitosamente. Los *SoC* ejecutaron los códigos implementados de manera correcta, y los sensores, conectados a los pines del *SoC* esclavo y alimentados por este, realizaron lecturas de datos de manera adecuada. Además, el relé conectado a uno de los pines del *SoC* esclavo permitió el control del sistema de irrigación por medio de la mini bomba. Finalmente, la conexión electrónica en el *PCB* desarrollado facilitó la integración del proyecto, reduciendo significativamente el cableado y mejorando la organización del sistema.

5 RECOMENDACIONES

- Durante la fase de creación de códigos para los dispositivos *Heltec WiFi LoRa 32 (V3)*, se recomienda utilizar las librerías más recientes y actualizadas. Las versiones anteriores pueden presentar fallos que dificulten el reconocimiento y la conexión del equipo. Asimismo, para el sensor *DHT22*, es fundamental emplear las librerías más recientes para garantizar su correcto funcionamiento.
- Los *SoC* pueden comunicarse eficazmente a largas distancias gracias a la tecnología *LoRa*, lo cual es una ventaja significativa para aplicaciones como esta. Es crucial aprovechar esta capacidad, asegurándose de verificar las frecuencias disponibles y permitidas por la agencia de regulación de cada país. Por ejemplo, en Ecuador, la frecuencia de 915 (MHz) está designada para uso libre, facilitando el despliegue y la operación de dispositivos *LoRa* en proyectos de *IoT*.
- Se recomienda emplear una placa *PCB* para el *SoC* y utilizar conectores (*headers*) para unir los sensores. Esto hace que las conexiones sean más sencillas y disminuye la probabilidad de errores

6 REFERENCIAS BIBLIOGRÁFICAS

- [1] «¿Qué es el Internet de las cosas (IoT)?», Oracle | Cloud Applications and Cloud Platform, [En línea]. Available: <https://www.oracle.com/ar/internet-of-things/what-is-iot/>. [Último acceso: 10 Junio 2024].
- [2] «AssIUT IOT: A Remotely Accessible Testbed for Internet of Things», Home Page, [En línea]. Available: <https://doi.org/10.1109/gciot.2018.8620157>. [Último acceso: 07 Mayo 2024].
- [3] «Tecnología LoRA y LoRAWAN - Catsensors», Catsensors: Sensores e instrumentación industrial, [En línea]. Available: [https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan#:~:text=LoRa%20es%20una%20tecnolog%C3%ADa%20ideal,para%20Smart%20Cities%20\(ciudades%20inteligentes](https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan#:~:text=LoRa%20es%20una%20tecnolog%C3%ADa%20ideal,para%20Smart%20Cities%20(ciudades%20inteligentes). [Último acceso: 07 Mayo 2024].
- [4] «Regulación - Agencia de Regulación y Control de las Telecomunicaciones», Agencia de Regulación y Control de las Telecomunicaciones - Promovemos el desarrollo armónico del sector de las telecomunicaciones, radio, televisión y las TIC , mediante la administración y regulación eficiente del espectro radioeléctrico y los servicios, [En línea]. Available: <https://www.arcotel.gob.ec/regulacion/>. [Último acceso: 07 Mayo 2024].
- [5] «Qué es LoRa, cómo funciona y características principales - Venco Electrónica», Venco Electrónica, [En línea]. Available: <https://www.vencoel.com/que-es-lora-como-funciona-y-caracteristicas-principales/>. [Último acceso: 07 Mayo 2024].
- [6] «WiFi LoRa 32(V3)», Heltec Automation. [En línea]. [Último acceso: 08 05 2024].
- [7] «DHT22 Temperatura y Humedad - AV Electronics», AV Electronics, [En línea]. Available: <https://avelectronics.cc/producto/dht22-temperatura-y-humedad>. [Último acceso: 08 Mayo 2024].
- [8] «“¿Qué es un system on a chip?”», /clases/1098-ingenieria/6552-que-es-un-system-on-a-chip/, 2020. [En línea]. Available: <https://platzi.com/clases/1098-ingenieria/6552-que-es-un-system-on-a-chip/>. [Último acceso: 20 Junio 2024].
- [9] «Integración con Adafruit IO | 350 integraciones con otros servicios están listas», Apix-Drive, [En línea]. Available: <https://apix-drive.com/es/adafruit#:~:text=Adafruit%20IO%20es%20un%20servicio,Huzzah%20ESP8266%20y%20muchos%20otros>. [Último acceso: 09 Mayo 2024].

- [10] «MQTT Essentials - All Core Concepts Explained,» HiveMQ – The Most Trusted MQTT platform to Transform Your Business, [En línea]. Available: <https://www.hivemq.com/mqtt>. [Último acceso: 10 Mayo 2024].
- [11] «¿Qué es MQTT? Definición y detalles,» Paessler - The Monitoring Experts, [En línea]. Available: <https://www.paessler.com/es/it-explained/mqtt>. [Último acceso: 10 Mayo 2024].
- [12] Y. Fernández, «Qué es Arduino, cómo funciona y qué puedes hacer con uno,» Xataka - Tecnología y gadgets, móviles, informática, electrónica, 23 Septiembre 2022. [En línea]. Available: <https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno>. [Último acceso: 20 Mayo 2024].
- [13] «Guía: ¿Qué es un invernadero? Tipos y Usos :,» Hydro Environment. Inovacion Agricola en un click, [En línea]. Available: https://www.hydroenv.com.mx/catalogo/index.php?main_page=page&id=44. [Último acceso: 24 Mayo 2024].
- [14] «Qué es un invernadero y sus características principales,» El Huerto De Pepe, [En línea]. Available: <https://elhuertodepepe.com/que-es-invernadero/>. [Último acceso: 24 Mayo 2024].
- [15] V. Cherlinka, «Humedad Del Suelo: Métodos E Instrumentos De Medición,» EOS Data Analytics, 29 Diciembre 2023. [En línea]. Available: <https://eos.com/es/blog/humedad-del-suelo/>. [Último acceso: 02 Julio 2024].
- [16] «Airthings | Smart indoor Radon & Air Quality Monitors Airthings,» ¿Qué es la humedad? | Airthings, [En línea]. Available: <https://www.airthings.com/es/what-is-humidity#:~:text=La%20humedad%20es%20una%20medida,que%20el%20aire%20puede%20contener..> [Último acceso: 02 Julio 2024].
- [17] «LoRa V1.3,» LILYGO®, [En línea]. Available: <https://www.lilygo.cc/products/lora-v1-3>. [Último acceso: 11 Junio 2024].
- [18] «SX1276 Datasheet(PDF),» ALLDATASHEET.COM - Electronic Parts Datasheet Search, [En línea]. Available: <https://www.alldatasheet.com/datasheet-pdf/pdf/800239/SEMTECH/SX1276.html>. [Último acceso: 11 Junio 2024].
- [19] «SX1276,» Semtech 半导体、物联网系统和云连接 | Semtech, [En línea]. Available: <https://www.semtech.cn/products/wireless-rf/lora-connect/sx1276>. [Último acceso: 13 Junio 2024].
- [20] «Mini bomba agua 5v-9v – Electronics Ecuador,» Electronics Ecuador – Tienda de Robótica, 2022. [En línea]. Available:

<https://www.electronicsecuador.com/producto/mini-bomba-agua-5v-9v/>. [Último acceso: 14 Junio 2024].

- [21] «BOMBA DE AGUA 12V SUMERGIBLE BRUSHLESS 240L/H – Grupo Electrostore,» Grupo Electrostore – Tienda de Electrónica y Robótica, 2023. [En línea]. Available: <https://grupoelectrostore.com/shop/motores/bombas-para-agua/bomba-de-agua-12v-sumergible-brushless-240l-h/>. [Último acceso: 14 Junio 2024].
- [22] «Sensor de temperatura y humedad relativa DHT22 (AM2302),» Naylamp Mechatronics - Perú, 2023. [En línea]. Available: <https://naylampmechatronics.com/sensores-temperatura-y-humedad/58-sensor-de-temperatura-y-humedad-relativa-dht22-am2302.html>. [Último acceso: 14 Junio 2024].
- [23] «Sensor de Humedad del Suelo FC-28,» Naylamp Mechatronics - Perú, 2023. [En línea]. Available: <https://naylampmechatronics.com/sensores-temperatura-y-humedad/47-sensor-de-humedad-de-suelo-fc-28.html>. [Último acceso: 23 Junio 2024].
- [24] «Amazon.com: Adaptador de corriente de 5 V 2 A, cargador USB C para auriculares Beats Flex Beats Studio Buds, Pro True Wireless Earbuds; JBL Charge 4, JBL Pulse 4, JBL Flip 5 con cable de fuente de alimentación : Electrónica,» Amazon.com, [En línea]. Available: https://www.amazon.com/-/es/Adaptador-corriente-cargador-auriculares-alimentaci%C3%B3n/dp/B083R6PLS6/ref=sr_1_3?adgrpid=152382796222&gad_source=1&hvadid=678889256869&hvdev=c&hvlocphy=9069516&hvnetw=g&hvqmt=b&hvrnd=15856242563917449655&hvtargid=kwd-547886. [Último acceso: 09 Julio 2024].
- [25] «Amazon.com,» Amazon.com, [En línea]. Available: <https://www.amazon.com/-/es/LAOGOG-JQC-3FF-S-Z-Disparador-Interfaz-Arduino/dp/B0BRY1NP4J>. [Último acceso: 7 7 2024].
- [26] C. Yañez, «Qué es Arduino IoT Cloud | CEAC,» CEAC, 03 Noviembre 2020. [En línea]. Available: <https://www.ceac.es/blog/que-es-arduino-iot-cloud>. [Último acceso: 15 Junio 2024].
- [27] «Software,» Arduino - Home, [En línea]. Available: <https://www.arduino.cc/en/software>. [Último acceso: 15 Junio 2024].
- [28] G. IoT, «Aprende a utilizar la plataforma Adafruit IO para tus dispositivos IoT (parte 1),» Generación IoT, 16 Junio 2022. [En línea]. Available:

<https://internetdelascosas.xyz/articulo.php?id=175&titulo=Aprende-a-utilizar-la-plataforma-Adafruit-IO-para-tus-dispositivos-IoT-parte-1->. [Último acceso: 15 Junio 2024].

- [29] «Blynk, plataforma de internet de las cosas en la red. - Tecnología Humanizada,» Tecnología Humanizada, [En línea]. Available: <https://humanizationoftechnology.com/blynk-plataforma-de-internet-de-las-cosas-en-la-red/revista/2018/volumen-4-2018/11/2018/>. [Último acceso: 15 Junio 2024].
- [30] «IFTTT,» Explore Integrations - IFTTT, [En línea]. Available: <https://ifttt.com/explore>. [Último acceso: 05 Julio 2024].
- [31] «Aprendiendo Arduino,» IDE Arduino, 11 Diciembre 2016. [En línea]. Available: <https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/>. [Último acceso: 19 Junio 2024].
- [32] «Heltec Products Operation Documentation — main latest documentation,» Heltec ESP32 Series Quick Start — esp32 latest documentation, [En línea]. Available: https://docs.heltec.org/en/node/esp32/quick_start.html. [Último acceso: 19 Junio 2024].
- [33] «Heltec ESP32 Series Quick Start — esp32 latest documentation,» Heltec Products Operation Documentation — main latest documentation, 2022. [En línea]. [Último acceso: 19 Junio 2024].
- [34] B. Fernández, «▷ IFTTT: qué es, cómo funciona y para qué sirve | InboundCycle,» Agencia de Inbound Marketing - InboundCycle, 21 Febrero 2023. [En línea]. Available: <https://www.inboundcycle.com/blog-de-inbound-marketing/ifttt-que-es-como-funciona-y-para-que-sirve>. [Último acceso: 20 Junio 2024].
- [35] «CAPÍTULO 4: CONTROL DEL MEDIO AMBIENTE,» Home | Food and Agriculture Organization of the United Nations, [En línea]. Available: <https://www.fao.org/4/s8630s/s8630s06.htm#:~:text=Las%20reacciones%20biol%C3%B3gicas%20de%20importancia,o%20por%20encima%20de%2050%C2%BAC..> [Último acceso: 10 Julio 2024].

7 ANEXOS

ANEXO I. Certificado de originalidad

ANEXO II. Enlaces

ANEXO III. Conjunto de datos extensos

ANEXO I: CERTIFICADO DE ORIGINALIDAD

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 31 de julio de 2024

De mi consideración:

Yo, **LEANDRO ANTONIO PAZMIÑO ORTIZ**, en calidad de Directora del Trabajo de Integración Curricular titulado **IMPLEMENTACIÓN DE UN PROTOTIPO CON SOC PARA CONTROL DE INVERNADERO MEDIANTE COMUNICACIÓN LORA** componente **IMPLEMENTACIÓN DE UN PROTOTIPO CON SOC PARA CONTROL DE INVERNADERO MEDIANTE COMUNICACIÓN LORA Y A TRAVÉS DE UNA INTERFAZ WEB** elaborado por la estudiante **VICTOR SAMUEL INLAGO GUALOTUÑA** de la carrera en Tecnología Superior en Redes y Telecomunicaciones, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 9%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

Atentamente,

Leandro Pazmiño Ortiz

Docente

Escuela de Formación de Tecnólogos

ANEXO II: ENLACES

A continuación, se encuentra el enlace del video de la simulación del proyecto:
[Samuel Inlago Tesis Prototipo invernadero.mp4](#)



Anexo II.I Código QR de la implementación y pruebas del prototipo de invernadero mediante comunicación *LoRa*.

ANEXO III: CÓDIGOS FUENTE

Código fuente del SoC esclavo

```
#include "LoRaWan_APP.h"

#include "Arduino.h"

#include <Adafruit_Sensor.h>

#include "DHT.h"

#define DHTPIN 4 // Se define el pin al que esta conectado el sensor DHT22

#define DHTTYPE DHT22 // Se define el tipo de sensor DHT que se está utilizando en
este caso es un DHT22

DHT dht(DHTPIN, DHTTYPE); //Se crea un objeto DHT llamado dht utilizando el pin
definido en DHTPIN y el tipo de sensor DHTTYPE

#define PIN_SALIDA_1 7 //Se define el pin de salida 1 como el pin 7 utilizado para
controlar el relé

#define Hum_Suelo 6 //Se define el pin Hum_Suelo como el pun 6 utilizado para el
sensor de humedad del suelo

#define RF_FREQUENCY 915000000

#define TX_OUTPUT_POWER 14

#define LORA_BANDWIDTH 0

#define LORA_SPREADING_FACTOR 7

#define LORA_CODINGRATE 1

#define LORA_PREAMBLE_LENGTH 8

#define LORA_SYMBOL_TIMEOUT 0

#define LORA_FIX_LENGTH_PAYLOAD_ON false

#define LORA_IQ_INVERSION_ON false

#define RX_TIMEOUT_VALUE 1000

#define BUFFER_SIZE 30
```

```

char txpacket[BUFFER_SIZE];

char rxpacket[BUFFER_SIZE];

static RadioEvents_t RadioEvents;

int16_t txNumber;

int16_t rssi, rxSize;

bool lora_idle = true;

String StrDataResp, StrDataIn;

float ValHum, ValTem = 0;

int HumSueloVal = 0;

int StrDataInVal = 0; // Variable para almacenar el valor convertido de StrDataIn

void OnTxDone(void);

void OnTxTimeout(void);

void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr);

void setup() {

    pinMode(PIN_SALIDA_1, OUTPUT); // Configura PIN_SALIDA_1 como salida
    digitalWrite(PIN_SALIDA_1, LOW); // Establece PIN_SALIDA_1 en cero
    Serial.begin(115200); // Inicia la comunicación serial a 115200 baudios
    Mcu.begin(); // Inicia la configuración del microcontrolador

    txNumber = 0; // Inicializa txNumber en 0
    rssi = 0; // Inicializa rssi en 0

    RadioEvents.RxDone = OnRxDone; // Asigna la función OnRxDone al evento
    RxDone

    RadioEvents.TxDone = OnTxDone; // Asigna la función OnTxDone al evento TxDone

    RadioEvents.TxTimeout = OnTxTimeout; // Asigna la función OnTxTimeout al evento
    TxTimeout

```

```

Radio.Init(&RadioEvents); // Inicializa la radio con los eventos definidos

Radio.SetChannel(RF_FREQUENCY); // Establece el canal de frecuencia establecido
en 915 (MHz)

Radio.SetRxConfig(MODEM_LORA, LORA_BANDWIDTH,
LORA_SPREADING_FACTOR,
LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
0, true, 0, 0, LORA_IQ_INVERSION_ON, true); // Configura los parámetros
de recepción LoRa

dht.begin(); // Inicializa el sensor DHT22
}

void loop() {

if (lora_idle) { // Si lora_idle es verdadero

lora_idle = false; // Establece lora_idle en falso

Serial.println("into RX mode"); // Imprime en la consola "into RX mode"

Radio.Rx(0); // Pone la radio en modo RX (recepción)

}

Radio.IrqProcess(); // Procesa las interrupciones de la radio

}

void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr) {

rssi = rssi; // se asigna rssi a sí mismo sin necesidad de cambio

rxSize = size; // Asigna size a rxSize

memcpy(rxpacket, payload, size); // Copia el payload recibido a rxpacket

rxpacket[size] = '\0'; // Termina rxpacket con el carácter nulo '\0'

Radio.Sleep(); // Pone la radio en modo Sleep

```

```

StrDataIn = String(rxpacket); // Convierte rxpacket a un objeto String StrDataIn

Serial.print("StrDataIn: "); // Imprime "StrDataIn: "

Serial.println(StrDataIn); // Imprime StrDataIn en la consola

// Verificar si el mensaje es "ON" o "OFF"

if (StrDataIn == "OFF") { // Si StrDataIn es igual a "OFF"

    digitalWrite(PIN_SALIDA_1, LOW); // Establece PIN_SALIDA_1 en cero

} else if (StrDataIn == "ON") { // Si StrDataIn es igual a "ON"

    digitalWrite(PIN_SALIDA_1, HIGH); // Establece PIN_SALIDA_1 en uno o
encendido

}

ValHum = dht.readHumidity(); // Lee la humedad del sensor DHT22

ValTem = dht.readTemperature(); // Lee la temperatura del sensor DHT22

if (isnan(ValHum) || isnan(ValTem)) { // Si ValHum o ValTem no son un número válido

    ValHum = 0; // Establece ValHum en 0

    ValTem = 0; // Establece ValTem en 0

    Serial.println(F("Error de lectura del sensor DHT22!")); // Imprime un mensaje de
error

}

int sensorValue = analogRead(Hum_Suelo); // Lee el valor del sensor de humedad de
suelo del sensor FC-28

HumSueloVal = map(sensorValue, 0, 4095, 99, 0); // Mapea el valor del sensor a un
rango de 0 a 99

// Convertir StrDataIn a un entero para comparar

StrDataInVal = StrDataIn.toInt();

// Verificar y controlar el PIN_SALIDA_1 según la comparación de valores

if (HumSueloVal < StrDataInVal) {

    digitalWrite(PIN_SALIDA_1, HIGH); // Enciende PIN_SALIDA_1

```

```

while (HumSueloVal < StrDataInVal) {

    sensorValue = analogRead(Hum_Suelo); // Lee el valor del sensor de nuevo

    HumSueloVal = map(sensorValue, 0, 4095, 99, 0); // Actualiza el valor mapeado

    delay(1000); // Espera 1 segundo antes de la próxima lectura

}

digitalWrite(PIN_SALIDA_1, LOW); // Apaga PIN_SALIDA_1 cuando la condición
ya no se cumple

}

Serial.print("% Humedad: "); // Imprime el valor de Humedad ambiental

Serial.print(ValHum);

Serial.print("% Temperatura: "); // Imprime el valor de Temperatura

Serial.print(ValTem);

Serial.print("°C Humedad Suelo: "); // Imprime el valor de humedad de suelo

Serial.print(HumSueloVal);

Serial.println();

    StrDataResp = String("DataRespuesta") + String("@") + String(ValHum, 2) +
String("@") + String(ValTem, 2) + String("@") + String(HumSueloVal); // Concatena los
datos en StrDataResp para enviarlos a través de LoRa

    int str_len = StrDataResp.length() + 1; // Calcula la longitud de StrDataResp más uno

    char char_TramaSend[str_len]; // Crea un buffer para la trama de datos a enviar

    StrDataResp.toCharArray(char_TramaSend, str_len); // Convierte la cadena de
datos de respuesta a un array de caracteres

    sprintf(txpacket, char_TramaSend); // Copia la trama de datos a enviar en el buffer
txpacket

    Serial.printf("\r\nEnviando Respuesta \"%s\" , longitud %d\r\n", txpacket,
strlen(txpacket)); // Imprime la trama de datos a enviar y su longitud en el monitor serial

    Radio.Send((uint8_t *)txpacket, strlen(txpacket)); // Envía la trama de datos a través
de la radio LoRa

```

```

    lora_idle = false; // Cambia el estado de LoRa a activo
}

void OnTxDone(void) {

    Serial.println("TX done....."); // Imprime un mensaje indicando que la transmisión ha
finalizado

    lora_idle = true; // Cambia el estado de LoRa a inactivo

    Radio.Rx(0); // Pone la radio en modo de recepción
}

void OnTxTimeout(void) {

    Radio.Sleep(); // Pone la radio en modo de reposo

    Serial.println("TX Timeout....."); // Imprime un mensaje indicando que la transmisión
ha superado el tiempo de espera

    lora_idle = true; // Cambia el estado de LoRa a inactivo
}

```

Código fuente SoC maestro

```

#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <WiFi.h>
#include <PubSubClient.h>
#include <Ticker.h>

Ticker tiempo_1;

// Credenciales Red WiFi
#define WIFI_SSID "Moises_Fullnet"
#define WIFI_PASSWORD "1727622217Sa$"

// Credenciales Adafruit
#define ADAFRUIT_USER "Samuelinlago22" // Define el nombre de la cuenta de
usuario en Adafruit IO.
#define ADAFRUIT_KEY "aio_xKST77iZvxZPKNUycrZIdeBoGhDM" // Define la clave
de API (API Key) para la cuenta de Adafruit IO
#define ADAFRUIT_SERVER "io.adafruit.com" // Define la dirección del servidor
MQTT de Adafruit IO
#define ADAFRUIT_PORT 1883 // Define el puerto del servidor MQTT de Adafruit IO
char ADAFRUIT_ID[30]; // Declara un arreglo de caracteres String llamado
ADAFRUIT_ID con un tamaño de 30 caracteres

```

```

// Publicar
#define ADAFRUIT_FEED_Humd ADAFRUIT_USER "/feeds/Heltec_Humd" // Define
la dirección del feed MQTT para publicar datos de humedad
#define ADAFRUIT_FEED_Temp ADAFRUIT_USER "/feeds/Heltec_Temp" // Define
la dirección del feed MQTT para publicar datos de temperatura
#define ADAFRUIT_FEED_HumSuelo ADAFRUIT_USER "/feeds/Heltec_HumSuelo"
// Define la dirección del feed MQTT para publicar datos de humedad del suelo

// Suscripción
#define ADAFRUIT_DATA_IN ADAFRUIT_USER "/feeds/dataOn_Off" // Define la
dirección del feed MQTT para suscribirse y recibir datos, se usa para recibir msg de on
y off desde Adafruit IO
#define ADAFRUIT_ADJUST_HUMIDITY ADAFRUIT_USER
"/feeds/Ajuste_Humedad" // Define la dirección del feed MQTT para suscribirse y
recibir datos, se usa para recibir msg de ajuste de humedad.

// Declaración de parámetros LoRa
#define RF_FREQUENCY 915000000
#define TX_OUTPUT_POWER 5
#define LORA_BANDWIDTH 0
#define LORA_SPREADING_FACTOR 7
#define LORA_CODINGRATE 1
#define LORA_PREAMBLE_LENGTH 8
#define LORA_SYMBOL_TIMEOUT 0
#define LORA_FIX_LENGTH_PAYLOAD_ON false
#define LORA_IQ_INVERSION_ON false
#define RX_TIMEOUT_VALUE 1000
#define BUFFER_SIZE 30

char txpacket[BUFFER_SIZE]; // Arreglo de caracteres para almacenar el paquete de
transmisión
char rxpacket[BUFFER_SIZE]; // Arreglo de caracteres para almacenar el paquete de
recepción

//Declaración de variables globales y funciones
float ValorFloat;
bool lora_idle = true;
bool BandSendData = false;
String StrTramaSend;
String StrDataLoraln, StrIndx;
float Val_Humd, Val_Temp, Val_HumSuelo;

static RadioEvents_t RadioEvents;
void OnTxDone(void);
void OnTxTimeout(void);
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr);
String getValue(String data, char separator, int index);

WiFiClient espClient;

```

```

PubSubClient client(espClient);
void setup_WiFi();
void callback(char* topic, byte* payload, unsigned int length);
void reconnect();
void mqtt_publish(String feed, float val);
void get_MQTT_ID();
void funcion_1();

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

  get_MQTT_ID();
  setup_WiFi();
  client.setServer(ADAFRUIT_SERVER, ADAFRUIT_PORT);
  client.setCallback(callback);

  Mcu.begin();
  RadioEvents.TxDone = OnTxDone;
  RadioEvents.TxTimeout = OnTxTimeout;
  RadioEvents.RxDone = OnRxDone;

  Radio.Init(&RadioEvents);
  Radio.SetChannel(RF_FREQUENCY);
  Radio.SetTxConfig(MODEM_LORA, TX_OUTPUT_POWER, 0,
LORA_BANDWIDTH,
                    LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                    LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                    true, 0, 0, LORA_IQ_INVERSION_ON, 3000);

  Radio.SetRxConfig(MODEM_LORA, LORA_BANDWIDTH,
LORA_SPREADING_FACTOR,
                    LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                    LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
                    0, true, 0, 0, LORA_IQ_INVERSION_ON, true);

  tiempo_1.attach(10, funcion_1);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  if (BandSendData && lora_idle) {
    BandSendData = false;
    // ACA SE REALIZA LA CONVERSION DE String a CharArray que se requiere de
    enviar en ese formato
    int str_len = StrTramaSend.length() + 1;
    char char_TramaSend[str_len];

```

```

    StrTramaSend.toCharArray(char_TramaSend, str_len);

    sprintf(txpacket, char_TramaSend);

    Serial.printf("\n\nEnviando Paquete \"%s\" , longitud %d\n\n", txpacket,
strlen(txpacket));
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    lora_idle = false;
    ValorFloat++;
}
Radio.IrqProcess(); // Procesa las interrupciones de LoRa, manejando eventos
como transmisiones y recepciones completadas
}

void funcion_1() { //se ejecuta periódicamente para indicar que hay datos listos para
enviar por LoRa
    BandSendData = true; // Indica que hay datos listos para enviar por LoRa
}

void OnTxDone(void) { //se ejecuta cuando se completa una transmisión LoRa
    Serial.println("TX done.....");
    lora_idle = true;
    Radio.Rx(0);
}

void OnTxTimeout(void) { //se ejecuta cuando la transmisión LoRa excede el tiempo
de espera
    Radio.Sleep();
    Serial.println("TX Timeout.....");
    lora_idle = true;
}

void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr) { //maneja los
datos recibidos por LoRa
    memcpy(rxpacket, payload, size);
    rxpacket[size] = '\0';
    Serial.printf("\n\nPaquete Recibido \"%s\" con RSSI %d , LONGITUD %d\n\n",
rxpacket, rssi, size);

    StrDataLoraln = String(rxpacket); // Convierte el buffer rxpacket a un objeto String y
lo almacena en StrDataLoraln
    // Extrae valores específicos de la cadena StrDataLoraln usando el separador '@' y
los índices correspondientes
    StrIdx = getValue(StrDataLoraln, '@', 0);
    Val_Humd = (getValue(StrDataLoraln, '@', 1)).toFloat();
    Val_Temp = (getValue(StrDataLoraln, '@', 2)).toFloat();
    Val_HumSuelo = (getValue(StrDataLoraln, '@', 3)).toFloat();

    Serial.print("Indice: "); Serial.println(StrIdx);
    Serial.print(" Val_Humd: "); Serial.print(Val_Humd);

```

```

Serial.print(" Val_Temp: "); Serial.print(Val_Temp);
Serial.print(" Val_HumSuelo: "); Serial.println(Val_HumSuelo);
// Publica los valores extraídos en los feeds de Adafruit IO correspondientes.
mqtt_publish(ADAFRUIT_FEED_Humd, Val_Humd);
delay(10);
mqtt_publish(ADAFRUIT_FEED_Temp, Val_Temp);
delay(10);
mqtt_publish(ADAFRUIT_FEED_HumSuelo, Val_HumSuelo);
delay(10);
}

String getValue(String data, char separator, int index) { // Función para obtener un
valor específico de una cadena basada en un índice y un separador
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) { // Bucle para recorrer la
cadena hasta que se encuentre el valor deseado o se llegue al final
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

// Función para Publicar por MQTT
void mqtt_publish(String feed, float val) {
    String value = String(val, 2);
    if (client.connected()) {
        client.publish(feed.c_str(), value.c_str());
        Serial.println("Publicando al tópico: " + String(feed) + " | mensaje: " + value);
    }
}

//*****
//***  CONEXIÓN WIFI  ***
//*****

void setup_WiFi() {
    delay(10);
    // Nos conectamos a nuestra red WiFi
    Serial.println();
    Serial.print("Conectando a ");
    Serial.println(String(WIFI_SSID));

    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");

```

```

}

Serial.println("");
Serial.println("Conectado a red WiFi");
Serial.println("Dirección IP: ");
Serial.println(WiFi.localIP());
}
// Función para capturar datos por MQTT
void callback(char *topic, byte *payload, unsigned int length) { // maneja la recepción
de mensajes MQTT, procesa y almacena los datos recibidos
  String mensaje = ""; // Declara una cadena vacía para almacenar el mensaje MQTT
recibido
  String str_topic(topic); // Convierte el tema MQTT (char*) en un objeto String llamado
str_topic

  for (uint16_t i = 0; i < length; i++) {
    mensaje += (char)payload[i]; // Concatena cada byte del payload en mensaje
como caracteres
  }

  mensaje.trim(); // Elimina cualquier espacio en blanco al inicio o final de la cadena
mensaje

  Serial.println("Tópico: " + str_topic); // Imprime en consola el tema del mensaje
recibido
  Serial.println("Mensaje: " + mensaje); // Imprime en consola el contenido del
mensaje MQTT

  if (str_topic == ADAFRUIT_DATA_IN || str_topic ==
ADAFRUIT_ADJUST_HUMIDITY) { // verifica si el tema del mensaje recibido por
MQTT es uno de los dos temas específicos ADAFRUIT_DATA_IN ó
ADAFRUIT_ADJUST_HUMIDITY
    StrTramaSend = mensaje; // Se guarda el mensaje en la variable global
SteTramaSend para enviarlo vía LoRa
    BandSendData = true; // Se indica que hay datos listos para enviar por
comunicación LoRa
  }
}
// Capturar el ChipID para Id de MQTT
void get_MQTT_ID() {
  uint64_t chipid = ESP.getEfuseMac();
  sprintf(ADAFRUIT_ID, sizeof(ADAFRUIT_ID), "%llu", chipid);
}

void reconnect() {
  while (!client.connected()) { // Mientras el cliente MQTT no esté conectado, se
ejecuta este bucle
    if (client.connect(ADAFRUIT_ID, ADAFRUIT_USER, ADAFRUIT_KEY)) { //
Intenta conectar al servidor MQTT de Adafruit con las credenciales y el ID del
dispositivo

```

```

    Serial.println("MQTT conectado!"); // Si la conexión es exitosa, imprime que
MQTT está conectado
    client.subscribe(ADAFRUIT_DATA_IN); // Se suscribe al tópico especificado
para recibir datos de ON y OFF
    client.subscribe(ADAFRUIT_ADJUST_HUMIDITY); // Se suscribe al tópico
especificado para recibir datos de Ajuste de Humedad
    Serial.println("Suscrito a los tópicos: " + String(ADAFRUIT_DATA_IN) + " y " +
String(ADAFRUIT_ADJUST_HUMIDITY)); // Imprime el tópico al que se ha suscrito
    } else {
        Serial.print("Falló :( con error -> "); // Si la conexión falla, imprime un mensaje
de error
        Serial.print(client.state()); // Imprime el estado actual del cliente MQTT
        Serial.println(" Intentamos de nuevo en 5 segundos"); // Indica que se intentará
nuevamente la conexión en 5 segundos
        delay(5000); // Espera 5 segundos antes de intentar reconectar
    }
}
}
}

```