ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE UN PROTOTIPO CON SOC PARA CONTROL DE NIVEL DE AGUA EN UNA CISTERNA MEDIANTE COMUNICACIÓN LORA

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

MELISSA ANAHÍ TIGSELEMA PACHECO

melissa.tigselema@epn.edu.ec

DIRECTOR: LEANDRO ANTONIO PAZMIÑO ORTIZ

leandro.pazmino@epn.edu.ec

DMQ, JULIO 2024

CERTIFICACIONES

Yo, Melissa Anahí Tigselema Pacheco 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.

Melissa Anahí Tigselema Pacheco

melissa.tigselema@epn.edu.ec

melissa.tigselema@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Melissa Anahí Tigselema Pacheco, bajo mi supervisión.

Leandro Antonio 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.

Melissa Anahí Tigselema Pacheco

C.I. 1723338131

DEDICATORIA

Principalmente a Dios, ya que sin él no estaría en donde estoy, por brindarme salud y me ha ayudado en los momentos en los que me quería rendir.

Con todo mi cariño a mis padres que me han motivado y me han convencido de salir de mi zona de confort cada día.

A mi querido hermano Alex que siempre me ha apoyado y me animado a perseverar ante las dificultades que se me han presentado. Y ha tenido fe en mí.

A mi hermano Daniel, que atraviesa su adolescencia con una perspectiva única sobre la vida.

AGRADECIMIENTO

A mis padres y hermanos que me por su paciencia inagotable y sus consejos.

A los Ingenieros, que me han desafiado a superarme y a dar mi máximo potencial. A mis compañeros de clase que me han brindado su ayuda y sus conocimientos. A mis amigos por los buenos momentos compartidos, por cada gesto amable que han tenido conmigo: Adriana Chicaiza, Cynthia Carvajal, Samuel Inlago, Vanessa Zurita.

A la Escuela Politécnica Nacional por abrirme las puertas otorgándome saberes y destrezas para en un futuro enfrentarme a los desafíos que me esperan fuera de esta hermosa Institución.

ÍNDICE DE CONTENIDOS

RESUMEN

El proyecto a realizar es sobre un sistema de medición de nivel de agua. Esto tiene un gran impacto en zonas donde el agua es escasa y se requiere juntar agua en los meses que llueve mucho y por lo cual es de gran ayuda tener una cisterna. El proyecto está conformado por dos dispositivos uno maestro conectado a *Wi-Fi* y a la interfaz web y el dispositivo esclavo es el que está conectado a los sensores y actuadores. Además, en la plataforma web se encuentran el ajuste de parámetros y los gráficos en donde se muestran nivel de agua y distancia.

En la sección inicial del proyecto, se definió claramente los objetivos a cumplir, se realizó una revisión profunda para establecer un marco teórico sólido y se delimitó el alcance del proyecto.

En la segunda sección se realizó la metodología en donde se detalla cómo se van a implementar cada uno de los objetivos propuestos.

En la tercera sección se presenta la implementación del proyecto de forma integral, detallando la elección de *hardware*, *software*, diseño y ejecución, asegurando que estos cumplan con los criterios del alcance del sistema.

Para la cuarta sección se probó el correcto funcionamiento con la maqueta creada, simulando una cisterna y realizando las pruebas pertinentes para verificación de alarmas.

Al finalizar el escrito se encuentra las conclusiones estás están redactadas de acuerdo con los objetivos planteados. De igual manera, las recomendaciones se dan en base a las dificultades presentadas durante la realización del proyecto. Luego están los anexos, en los cuales se encuentran el video de funcionamiento y el código utilizado para la implementación.

PALABRAS CLAVE: *Wi-Fi LoRa ESP32 Heltec (V2), Sensor HC-SR04, Adafruit IO*, Monitoreo remoto, Interfaz web

ABSTRACT

The project to be carried out is about a water level measurement system. This has a great impact in areas where water is scarce and it is required to collect water in the months of heavy rainfall and therefore it is helpful to have a cistern. The project consists of two devices, a master device connected to Wi-Fi and the web interface and the slave device is the one connected to the sensors and actuators. In addition, the web platform contains the parameter settings and the graphs where water level and distance are displayed.

In the initial section of the project, the objectives to be met were clearly defined, a thorough review was carried out to establish a solid theoretical framework and the scope of the project was delimited.

In the second section, the methodology was developed, detailing how each of the proposed objectives will be implemented.

The third section presents the implementation of the project in a comprehensive manner, detailing the choice of hardware, software, design and execution, ensuring that these meet the criteria of the scope of the system.

For the fourth section, the correct operation was tested with the created model, simulating a cistern and performing the pertinent tests for alarm verification.

At the end of the paper, the conclusions are written according to the objectives set. Similarly, the recommendations are given based on the difficulties encountered during the project. Then there are the annexes, which contain the video of operation and the code used for the implementation.

KEYWORDS: *Wi-Fi LoRa ESP32 Heltec (V2), HC-SR04 Sensor, Adafruit IO, Remote Monitoring, Web Interface*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

En varias provincias del Ecuador no se cuenta con un servicio de agua potable, 9 de las 24 provincias tienen el índice más bajo en la medición nacional entre ellas figuran 6 provincias de la región costa y 3 de la sierra. La baja cobertura del servicio básico es consecuencia de la ineficiencia administrativa, la lejanía entre los centros poblados y la insuficiencia de recursos [1]. Una solución a esta problemática se podría resolver mediante la realización de una cisterna en los poblados más distantes. Para que tengan una reserva en tiempo de sequía.

El prototipo de medición de nivel de agua con tecnología *LoRa* presenta una solución altamente conveniente para zonas remotas, gracias a su bajo costo de implementación, bajo consumo de energía y amplio alcance. Este innovador sistema permite monitorear el nivel de agua de manera remota y eficiente, asegurando que se active el llenado automático una vez que llegue al nivel mínimo configurado. De esta manera, se garantiza un suministro constante de agua, incluso durante periodos de escasez.

La configuración emplea dos System-on-a-Chip (*SoC*) diferenciados: uno para la conexión *Wi-Fi* y la interfaz web, y otro para la interacción con los actuadores y el sensor HC-SR04. Ambos *SoC* se comunican bidireccionalmente mediante tecnología *LoRa*, permitiendo una interacción fluida.

El sensor HC-SR04 permite enviar sondas y por medio de estas logra identificar en qué lugar se encuentra algún objeto. En este caso envía una señal y mide la distancia. Luego con una operación matemática se puede sacar el nivel de agua mediante la separación que tiene el agua del sensor. Y el actuador con la mini bomba de agua se encargan de el llenado de la cisterna de acuerdo con los parámetros pertinentes.

Por otro lado, el *SoC* que se conecta a una red *Wi-Fi* debe tener en el código el nombre de la red con su correspondiente contraseña para acceder a internet. Además, contará con las credenciales de *Adafruti IO* proporcionas por la interfaz una vez creada la cuenta.

La construcción de la maqueta demandó un balde con un orificio de salida de agua, replicando el vaciado manual que realizan las personas en sus hogares. Además, se requirió un suministro de agua para simular el llenado de la cisterna. Este proyecto tiene la finalidad de tener un suministro de agua.

1.1 Objetivo general

Implementar un prototipo con *SoC* para control de nivel de agua en una cisterna mediante comunicación *LoRa*

1.2 Objetivos específicos

Definir los requisitos técnicos y funcionales para el control del nivel de agua.

Seleccionar el *hardware* y *software* acorde a los requerimientos establecidos.

Diseñar el prototipo del sistema de control de nivel de agua.

Implementar el prototipo del sistema de control del nivel de agua.

Realizar pruebas de funcionamiento del prototipo.

1.3 Alcance

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

Monitoreo del nivel de agua de la cisterna.

Control de la apertura y cierre del sistema para llenar la cisterna.

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 la cisterna.

1.4 Marco Teórico

Comunicación *LoRa*

Con el continuo avance tecnológico, en el año 2012 en el continente europeo se encontró con una modulación que tenía muchos beneficios entre los cuales se destacan la tolerancia al ruido, modulación de amplio espectro y frecuencias de operación que permiten comunicaciones a larga distancia. Además, cuenta con una técnica de modulación utilizada por los equipos para alcanzar una vasta cobertura con un consumo de energía reducido [2].

Por otra parte, agiliza el intercambio de información con una velocidad reducida entre dispositivos. Que van desde los 0.3 (Kbps) a 50 (Kbps). No obstante, hay gran variedad de dispositivos conforme al área en donde se desea trabajar [2].

Internet de las cosas *(IoT)*

Se define como Internet de las cosas a un conjunto de dispositivos conectados por internet que envían información de los datos, los cuales se recolectan mediante sensores, *software* u objetos mecánicos. Estos objetos que con tan solo estar entrelazados por medio de internet pueden interactuar entre sí, sin necesitar la supervisión de un ser humano. Esto es gracias al gran ancho de banda que se dispone hoy en día. Ya que estas tecnologías requieren una red estable. Existe una variada gama de usos en el ámbito de la *IoT* como por ejemplo controlar la calefacción de un inmueble, monitorear un área ganadera, etc. [3].

Wi-Fi LoRa ESP32 Heltec (V2)

Este es un poderoso *SoC*, cuenta con *Wi-Fi* por lo cual permite conectarse a Internet y por medio de este obtener actualizaciones y efectuar aplicaciones basadas en la Nube. Igualmente, concede varias posibilidades de comunicación y de control remoto. El estándar de Internet que maneja es 802,11 b/n/g. Además, cuenta con Bluetooth, por lo que permite conexión inalámbrica con un reducido alcance.

Por otro lado, lo que destaca de este dispositivo es que dispone de un módulo *LoRa* el cual es utilizado para comunicación a largas distancias con un consumo de energía pequeño. Por lo que es una gran ventaja ya que con otras tecnologías esto no sería posible, así mismo, esta tecnología fue creada para proyectos que utilicen *IoT* [4].

Mini bomba de agua

Bomba de agua, diseñada para operar en el rango de 2.5 y 6 (V_{DC}), se muestra como una solución ágil y segura al momento de fabricar microcircuitos. Con una capacidad de hasta 2 litros por minuto, esta bomba es óptima para realizar una extensa variedad de tareas y proyectos que requieran un caudal modesto.

Equipada con un motor interno de 0.3 (A) y una estructura de termoplástico resistente a la corrosión, garantiza durabilidad y resistencia en diversos entornos. Además, su diseño completamente sumergible facilita su instalación: basta con conectar una pequeña manguera a la salida de la bomba, ajustar el voltaje según las necesidades y sumergirla por completo [5].

Sensor

Un sensor es un dispositivo que detecta o percibe cambios en el entorno, como variaciones de movimiento, calor, luz y otros elementos. La información que recibe se convierte en una señal eléctrica. Esta señal es procesada por circuitos que pueden desencadenar una acción específica en un sistema o máquina.

Estos sistemas electrónicos operan basados en los mismos fundamentos que los circuitos eléctricos, por lo que es esencial poder regular el paso de la corriente eléctrica. En detalle, un sensor toma los estímulos del entorno y los convierte en señales eléctricas. Posteriormente, las señales se transmiten mediante una interfaz que las convierte en código binario, para después ser tratados por una computadora [6].

Actuadores eléctricos

Es un elemento electromagnético que actúa como botón automático, esto quiere decir que es controlado por un circuito. Este dispositivo, conocido como relé, emplea una bobina, electroimán para monitorear algunos contactos eléctricos. En el momento, en que el circuito de control envía una señal, activa el electroimán, lo que permite que a través de la bobina cruce la corriente. Esto crea un campo magnético que interactúa con los contactos del relé, provocando su apertura o cierre. Así, el relé puede abrir o cerrar los circuitos eléctricos que tiene conectados de manera independiente. Este proceso permite controlar un circuito de salida, usando un circuito de control que requiere una pequeña fuente de voltaje para accionar al relé [7].

Arduino IDE

El *software Arduino IDE* ofrece un ambiente de desarrollo completo que permite programar dispositivos *Arduino* utilizando uno o varios lenguajes de programación. Incluye varios componentes esenciales: un editor de código donde se escribe el programa, un compilador que traduce el código a un lenguaje que la computadora pueda entender, un depurador para encontrar y corregir errores, y un constructor de interfaz gráfica (GUI) para diseñar visualmente las interfaces de usuario.

El *IDE* de *Arduino* proporciona funcionalidades para transferir y ejecutar el programa en la memoria flash del dispositivo, lo que agiliza el progreso y la ejecución de proyectos en la plataforma [8].

Adafruit IO

La plataforma *Adafruit IO*, dirigida por empresa *Adafruit Industries*, facilita la recopilación, el resguardo y la presentación inmediata de datos generados por dispositivos *IoT* en la nube. Ofrece una plataforma web fácil de usar para el control de dispositivos. Es compatible con algunos microcontroladores los cuales son *Raspberry Pi, Arduino y ESP8266.*

La utilización de esta plataforma es sencilla por lo cual, a la hora de realizar un proyecto, es accesible y completa. Para utilizar esta plataforma se debe tener en cuenta las pestañas principales las cuales son los *dashboard* y los *feeds.* Los *dashboards* es similar

a un panel de trabajo donde se coloca los bloques o gráficos que componen nuestro proyecto. Los *feeds* actúan como intermediarios para la transmisión de datos entre el sensor y la plataforma [9].

2 METODOLOGÍA

Principalmente se contemplaron todas las funcionalidades que debe tener el prototipo, por lo que se realizó un amplio estudio de lo que se requiere. Asegurando que se cumplan todos los requisitos con la debida atención. Por lo que, desde el inicio del desarrollo, se enfocó en el área de tener una interfaz gráfica en donde se puedan recopilar y visualizar los datos teniendo en cuenta que el tipo de comunicación utilizando sea *LoRa.*

Como siguiente paso, elección de *software* y el *hardware*. Estos deben ser acorde a las características que demanda el prototipo, los *SoCs* deben ser compatibles con el rango de frecuencias en la región en la que se van a implementar y el sensor debe ser resistente. En el *software* en cambio la plataforma a usar debe conectarse al módulo *Wi-Fi LoRa ESP32 Heltec (V2)* esto es de suma relevancia porque en esta, se cargará el código para que el prototipo realice las funciones especificadas en el plan de trabajo.

El diseño del prototipo inició por el desarrollo de códigos para controlar el nivel de agua y establecer comunicación inalámbrica mediante *LoRa.* Estos códigos se diseñarán para garantizar un monitoreo preciso del nivel de agua y la distancia a la que debe llegar para que no haya desperdicio de la misma teniendo así una transmisión confiable de datos. Una vez completados y probados los códigos, se implementarán en el *SoC* y se verificará su funcionamiento en un entorno físico. Posteriormente, se creará un *dashboard* web en *Adafruit IO para* proporcionar a los usuarios una interfaz para visualizar y gestionar los datos del sistema.

Una vez seleccionado el *hardware* adecuado, elegido el *software* pertinente y completado el desarrollo de los códigos necesarios para el prototipo, se procederá a llevar a cabo la implementación del sistema. Este proceso implica la integración de las diferentes partes previamente desarrolladas, uniendo los elementos de *hardware* y *software* de manera coherente y funcional. Además, se creará un prototipo que simula el despliegue del sistema de control del nivel de agua. Este prototipo servirá como herramienta para realizar pruebas y ajustes, garantizando así su correcto funcionamiento y eficacia en la gestión del nivel de agua.

Tras completar la fase de implementación del prototipo, se harán pruebas de funcionamiento para confirmar su buen rendimiento. Los resultados de estas pruebas

serán utilizados para corregir posibles errores, asegurando así que el prototipo satisfaga los requisitos establecidos.

3 RESULTADOS

En la fase de implementación del proyecto se desarrollará un prototipo para medir el nivel de agua utilizando un *SoC* y comunicación *LoRa*. Además, con un relé se podrá controlar el apagado y encendido de la mini bomba ya sea para llenar la cisterna o para apagarla en caso de que ya, se haya abastecido.

De igual manera, se implementará un *dashboard* para que el usuario pueda visualizar los datos obtenidos por lo sensores a través de internet por medio de la plataforma *Adafruit IO*.

3.1 Definición de requisitos técnicos y funcionales para el nivel de control de agua

A continuación, se presentan requisitos necesarios para implementar el prototipo

Dispositivo *máster*

Este dispositivo debe contar con acceso a Internet y también con comunicación *LoRa* ya que se requiere que el *SoC*, maneje ambas tecnologías para hacer un poco más sencillo el proceso de implementación. Debe trabajar en las frecuencias libres del país donde se desarrollará, en este caso, la frecuencia 916 (MHz) para que no haya interferencias.

Igualmente, el código para cargar al *SoC* se hará utilizando la plataforma *Arduino IDE* el cual su lenguaje de programación es C++. En este se deberá tomar en cuenta las librerías que necesita el código para que el funcionamiento sea el adecuado.

Dispositivo *slave*

Este es un equipo el cual estará conectado a los sensores, relé y bomba de agua. Debe utilizar la tecnología *LoRa* para transmitir datos de manera eficiente y a larga distancia al dispositivo máster. Además, a este equipo se deben conectar los sensores que detectan el nivel de agua en tiempo real, el relé que controla el encendido y apagado de una mini bomba según las lecturas, la bomba se activa automáticamente en función a los parámetros configurados. La comunicación *LoRa* garantiza una transmisión de datos fiable, incluyendo niveles de agua, alertas lo cual es ideal para áreas remotas o de difícil acceso.

Este equipo debe ser completamente compatible con el dispositivo *máster*, centralizando la recepción y análisis de datos en la plataforma *Adafruit IO*.

Sistema de comunicación con *LoRa*

El prototipo debe emplear comunicación *LoRa* ya que se debe transmitir los datos proporcionados por los sensores al dispositivo *máster* y viceversa para controlar el funcionamiento de la mini bomba. Este diseño es ideal, ya que no debe tener interferencias para no alterar los valores enviados por los sensores. Otro factor importante es el largo alcance que tiene ya que este permite de 10 a 20 (Km) la conexión. Con un limitado consumo de energía.

LoRa se presenta como una opción óptima para configurar redes de *IoT* que requieran sensores operativos sin acceso a corriente eléctrica convencional por lo cual es un gran beneficio [10].

Implementación de código

Para crear un programa eficiente en *Arduino IDE* que integre todos los elementos y funciones sin interferir entre sí, es crucial seguir buenas prácticas como definir claramente los requisitos, dividir el programa en módulos o funciones separadas, etc. También es importante utilizar temporizadores para sincronizar funciones y manejar interrupciones de manera eficiente, minimizando el procesamiento dentro de las rutinas de interrupción. Realizar pruebas exhaustivas de cada módulo antes de integrarlos.

Adicionalmente, documentar y comentar el código de manera clara facilita su comprensión y mantenimiento. Aplicando estos principios, se logra un programa en *Arduino IDE* que funcione correctamente, sea fácil de mantener y evite interferencias entre funciones.

Sistema de monitoreo

Para este prototipo, el sistema de monitoreo se debe diseñar para detectar condiciones anormales, como el estado de la cisterna cuando esta se encuentre vacía, y tomar acciones correspondientes con un código integrado. En tales circunstancias, se enviaría una orden para activar el llenado de la cisterna hasta alcanzar una altura predeterminada.

Por otro lado, si la cisterna se encuentra llena, el sistema cerraría automáticamente la mini bomba para evitar desperdicio de agua o cualquier otro problema asociado. Esta funcionalidad garantiza un control eficiente del nivel de agua y un uso óptimo de los recursos disponibles.

Diseño de la interfaz web

Al diseñar la interfaz web, es fundamental elegir una plataforma que ofrezca una interfaz gráfica y funcional, permitiendo a los usuarios interactuar de manera eficiente con el contenido. Además de su aspecto estético y facilidad de uso, es esencial integrar funcionalidades prácticas que mejoren la experiencia del usuario. La interfaz web ofrece la posibilidad de gestionar el funcionamiento de la mini bomba de forma remota, permitiendo a los usuarios encender y apagar la bomba cuando lo requieran.

Esta característica no solo añade valor al sitio web al ofrecer control remoto conveniente, sino que también puede mejorar la eficiencia operativa y la comodidad para los usuarios al brindarles control directo sobre el sistema desde cualquier lugar con acceso a internet.

Construcción de circuito electrónico

Para garantizar el éxito de nuestro circuito, es fundamental contar con un diseño sólido y verificado. Simuladores como Proteus nos permiten validar el funcionamiento del circuito en un entorno virtual antes de su construcción física, ahorrando tiempo y recursos. Gracias a las simulaciones, podemos anticipar y corregir posibles errores de diseño, optimizando así el proceso de desarrollo.

Una vez finalizada la etapa de diseño electrónico, se procede a la fabricación de la placa de circuito impreso (*PCB*). Este proceso implica la transformación del esquema eléctrico en una representación física, donde los componentes electrónicos se ubican de manera precisa sobre una lámina conductora y se conectan mediante pistas metálicas.

3.2 Selección del *hardware* y *software*

En Ecuador, es posible encontrar una variedad de componentes electrónicos adecuados para la implementación de este proyecto. Ya sea para el *SoC*, dispositivos que recolecten información como lo son los sensores. Para la ejecución de este esquema, se llevó a cabo un análisis técnico exhaustivo para evaluar los componentes individuales de este prototipo.

Determinación del *SoC*

Para determinar el dispositivo que será de *SoC* se tomó en cuenta que las características indispensables son: que cuente con conexión a *Wi-Fi,* comunicación por medio de *LoRa,* capacidad de integración de sensores. Debe tener un tamaño compacto. Los modelos que gozan de algunas de estas cualidades anteriormente mencionadas son: *Wi-Fi LoRa ESP32 Heltec (V2), Wireless Stick Lite(V3), ESP32-WROOM-32.*

El modelo *Wi-Fi LoRa ESP32 Heltec (V2)* es un módulo que permite integrar 3 tecnologías *Wi-Fi, Bluetooth*, *LoRa*. Esta última tecnología, enfocada en la comunicación a distancia, opera en las frecuencias entre 863 y 928 (MHz). Al contar con el *controlador SX1276*, resulta ideal para establecer conexiones de hasta 3.6 (km) de alcance. Adicionalmente esta placa trabaja con el estándar de *Wi-Fi 802.11 b/g/n* en la banda de frecuencia de 2.4 a 2.5 (GHz). Como se ilustra en la Figura 3.1 se puede ver la distribución de pines [11].

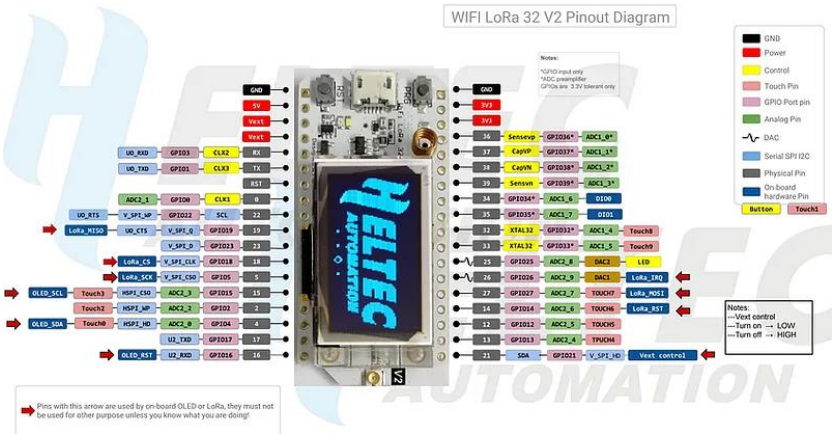


Figura 3.1 Wi-Fi LoRa ESP32 Heltec (V2) [12]

El módulo *Wireless Stick Lite (V3)* posee un microprocesador *ESP32-S3FN8* el cual ofrece alto rendimiento con doble núcleo a 240 (MHz). Conectividad versátil por tener *Wi-Fi, LoRa* y Bluetooth para cubrir un amplio abanico de aplicaciones en el *IoT*. Además, con una interfaz *USB-C* la cual tiene protecciones completas y regulador de voltaje. Gestión de batería de litio ya sea para protección y detección de energía. *Chip CP2102* el cual facilita la descarga de programas y depuración. En la siguiente Figura 3.2 se muestra los pines que posee este dispositivo [13].

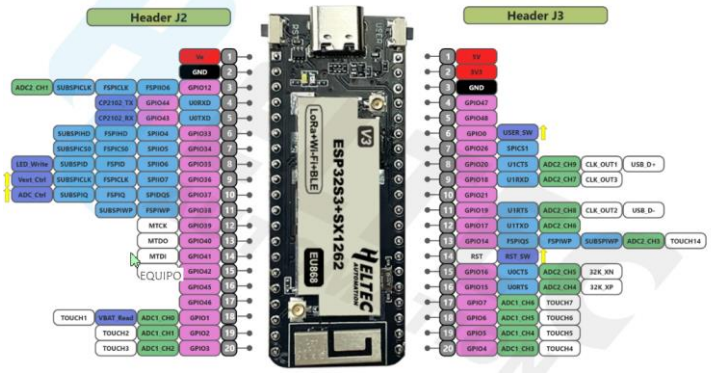


Figura 3.2 *Wireless stick lite (*V3) [14]

El siguiente módulo es el *ESP32-WROOM-32* es un dispositivo que dispone de las siguientes características: posee un chip *ESP32-D0WDQ6* el cual es esencial ya que

debe adaptarse todos entornos que maneja. De igual manera, trabaja con el estándar de *Wi-Fi 802.11b/g/n* en el rango de frecuencias de la 2412 a la 2484 (MHz). Cuenta con la tecnología *Bluetooth* en la versión 4.2 *BR/EDR y LE*. Como se muestra en la Figura 3.3 se requiere de un voltaje de alimentación de 3.3 a 5 (V) [15].

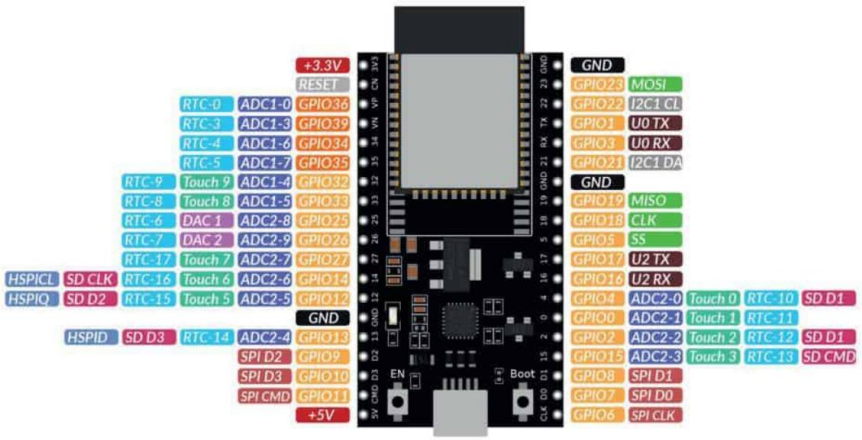


Figura 3.3 ESP32-WROOM-32 [15]

Teniendo en cuenta las características de los *SoCs* anteriormente mencionados y verificado las exigencias del prototipo. Se realizó una comparación de las especificaciones de los modelos en función a los objetivos de desarrollo del proyecto como se aprecia en la Tabla 3.1.

Tabla 3.1 Comparación de dispositivos [11] [13] [15] [16]

Componente	***Wi-Fi LoRa ESP32 Heltec (V2)***	***Wireless stick lite (V3)***	***ESP32-WROOM-32***
Estándar de *Wi-Fi*	*802.11 b/g/n*	802.11 b/g/n	*Wi-Fi 802.11b/g/n*
Voltaje de trabajo (V)	5	3.7 a 5	3,3 a 5
Chip *LoRa*	SX1276	SX1262	No Integrado
Dimensiones (mm)	51 x 25 x 7	58.08 x 22.6 x 8.2	18 x 25.5 x 3.10
Potencia (dBm)	20	21	14
Sensibilidad (dBm)	-148	-134	-69 a -98
Costo en dólares	38.00	32.50	10.00

En base a la información recopilada de la Tabla 3.1 se decidió que la mejor opción es el *SoC Wi-Fi LoRa ESP32 Heltec (V2)* debido a sus capacidades y el tamaño de este. Un factor adicional que determinó la elección de este dispositivo fue su disponibilidad en el país. Dado que tanto el módulo *ESP32-WROOM-32* no cuenta con conectividad *LoRa*

y al no encontrar el dispositivo *Wireless Stick Lite (V3)*. La mejor opción es *Wi-Fi LoRa ESP32 Heltec (V2).*

Determinación del sensor de nivel de agua

Sensor HC-SR04 ultrasónico, este dispositivo detecta y calcula a que distancia se encuentra los objetos ya que emite un pulso ultrasónico el cual mide el tiempo del eco que envía. Tiene un bajo impacto energético. Además, su voltaje de alimentación es de 5 (V). Es capaz de medir distancias entre 2 (cm) hasta los 4.5 (m). Ángulo de medición 15 grados con una frecuencia de sonda de 40 (KHz). Su duración del pulso ultrasónico es de 10 (μS). Lapso de pulso de salida va de 100 a 25000 (μS). El presente dispositivo se observa en la Figura 3.4 [17].

Figura 3.4 Sensor HC-SR04 [17]

Sensor JSN-SR04T ultrasónico, resistente al agua. Rango de medición de 20 (cm) a 6 (m). Su voltaje de alimentación va desde 3 a 5 (V) con una corriente de 8 (mA). Tiene un ángulo de medición de 75 grados y una frecuencia de trabajo de 40 (KHz). Ideal para entornos hostiles donde la exposición a la intemperie es un factor importante. El modelo de este sensor se encuentra en la Figura 3.5 [18].

Figura 3.5 Sensor JSN-SR04T [18]

Considerando las propiedades de los sensores previamente descritos y tras evaluar las demandas del prototipo, se llevó a cabo una comparación de las características técnicas de los modelos presentados en la Tabla 3.2 en relación con las metas de desarrollo del proyecto.

Tabla 3.2 Comparación de características de sensores [17] [18]

Componente	HC-SR04	JSN-SR04T
Tensión de Alimentación (V)	5	3 a 5
Frecuencia de sonda (KHz)	40	40
Rango de medición de (cm) a (m)	2 a 4.5	20 a 6
Ángulo de medición en °	15	45
Resistente al Agua	No	Si
Costo en dólares	4.60	21.00

Al seleccionar el sensor adecuado, se consideraron diversos factores importantes: costo, rango de medición, tamaño y facilidad de implementación. Tras evaluar estas características, se determinó que el sensor HC-SR04 es la opción más idónea.

Selección del relé

Módulo relé este es capaz de gestionar dispositivos que operen hasta con 250 (V) y 10 (A). Cada canal está aislado eléctricamente mediante un optoacoplador, e incluye un indicador *LED* para visualizar su estado. Su diseño práctico lo hace ideal para trabajar con *Arduino*. Su funcionamiento lógico, donde la salida normalmente abierta (NO) se activa al recibir un "0" lógico (0V) y se desactiva con un "1" lógico (5V). El dispositivo mencionado se ilustra en la Figura 3.6 [19] [20].

Figura 3.6 Módulo relé

Las características se detallan a continuación en la Tabla 3.3.

Tabla 3.3 Características del módulo relé [19] [20]

Componente	Modulo relé
Voltaje de Funcionamiento (V)	5
Tiempo de acción (ms)	De 5 a 10
Indicadores de activación	Led
Dimensiones (mm)	51 x 31
N° de canales	1
Costo en dólares americanos	3.6

El módulo de relé de un canal es una opción ideal para controlar una mini bomba de agua de tamaño reducido debido a su operación segura y eficiente, control preciso del flujo de agua, y facilidad de integración en sistemas compactos. Además, su versatilidad lo hace adecuado para diversas aplicaciones, y su alta fiabilidad reduce la necesidad de mantenimiento.

Determinación de la mini bomba

La mini bomba de agua está diseñada para mover líquidos dentro y fuera de recipientes, siendo compatible con diferentes dispositivos y operando a un voltaje de 2.5 a 6 (V). Construida en plástico, su tamaño compacto es de 43 (mm) x 23 (mm) como se muestra en la Figura 3.7. Esta bomba puede elevar una columna de agua hasta 40 (cm). El grosor externo del conducto de salida es de 7.5 (mm), en contraste con el grosor interno de 5 (mm) [21].

Figura 3.7 Mini bomba de agua [21]

Algunas de las características más relevantes de este dispositivo son las siguientes mencionadas en la Tabla 3.4 .

Tabla 3.4 Características de mini bomba de agua [21]

Componente	Mini bomba
Voltaje de operación (V)	2.5 a 6
Corriente de funcionamiento (mA)	130 a 220
Material	Plástico
Tamaño (mm)	43 x 23
Grosor externo de tubo de salida (mm)	7.5
Grosor interno de tubo de salida (mm)	5
Costo en dólares	3.04

Determinación de fuentes de alimentación

Batería de litio o también llamada (LiPo). Son baterías recargables, compuestas a veces por varias celdas, son ideales para múltiples funciones que demanden corrientes superiores a 1 (A), combinando un peso ligero con tamaño reducido como se lográ visualizar en la Figura 3.8 . Tiene una vida útil de 2 a 3 años, disponen de diferentes tamaños. Las baterías recargables no solo se distinguen por su capacidad para ser reutilizadas, sino que también conservan su eficacia a lo largo de múltiples ciclos de carga y descarga [22].

Figura 3.8 Batería de litio (LiPo) *[22]*

Fuente de alimentación con conector micro *USB*. Este versátil adaptador ofrece 5 (V) a 1 (A) de potencia a través de un puerto micro *USB*, ideal para recargar baterías y suministrar energía a una gran variedad de dispositivos electrónicos, alimentar placas de desarrollo siempre y cuando cuenten con un puerto micro *USB* como se indica en la Figura 3.9 [23] [24].

Figura 3.9 Fuente de alimentación con conector micro *USB* [23] [24]

De manera resumida, los detalles técnicos de las fuentes de alimentación se pueden identificar en la Tabla 3.5.

Tabla 3.5 Comparación batería de litio y fuente de alimentación [22] [23] [24] [25]

Componentes	Batería de Litio	Fuente de alimentación
Voltaje de Trabajo (V)	3.7	5
Corriente de Salida (mA)	1200	1000
Conector micro *USB*	No	Si
Dimensiones (cm)	5.08 x 3.35 x 0.59	5 x 5 x 4
Peso (kg)	0.022	0.2
Costo en dólares	5.00	6.00

Para la implementación del prototipo, se optó por una fuente de alimentación con conector micro *USB* por varias razones fundamentales. En primer lugar, este tipo de conector es ampliamente compatible con muchos dispositivos y entornos, lo que simplifica la integración del prototipo con otros equipos y sistemas existentes. Además, el uso de un conector micro *USB* permite una conexión estable para asegurar un suministro de energía constante durante el funcionamiento del proyecto.

Esto no solo mejora la estabilidad operativa, sino que también facilita el mantenimiento periódico al proporcionar una forma estándar y conveniente de realizar ajustes o actualizaciones necesarias en el prototipo. Esta elección estratégica no solo optimiza el rendimiento del proyecto, sino que también asegura que pueda adaptarse y evolucionar eficientemente con futuras mejoras o cambios en los requisitos técnicos.

3.3 Diseño del prototipo

Selección de la plataforma

Las plataformas que se adaptan a los requerimientos del proyecto las cuales son 2 principalmente: *Arduino IoT Cloud, Adafruit IO*.

Arduino IoT Cloud simplifica la creación de esquemas de programación con configuraciones automáticas y rápidas. Esto acelera el proceso para que los desarrolladores conviertan un tablero de control en un dispositivo funcional en apenas unos minutos. La plataforma también ofrece múltiples formas de interacción, como *API REST HTTP, MQTT,* utilidades de línea de comandos, *JavaScript,* etc. Requiere el uso de placas pertenecientes a la familia *MKR*, que facilitan la creación de nodos para *IoT* con diversas opciones de conectividad y soporte para *hardware* de terceros y sistemas basados en la nube con interfaz gráfica como se indica en la Figura 3.10 [26].

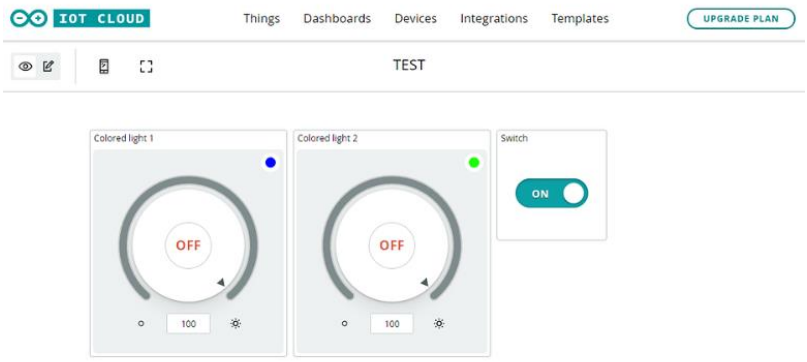


Figura 3.10 *Dashboard Arduino IoT Cloud* [27]

La versión gratuita de *Adafruit IO* limita el envío de datos a 30 registros por minuto y almacena la información durante un máximo de 30 días. Permite activar alertas o disparadores cada 15 minutos y crear hasta 5 paneles de control, cada uno con 10 elementos o fuentes de datos. Al iniciar sesión, se muestra un resumen de los paneles activos y se accede a la información personal y la clave única asignada por la plataforma. Asimismo, su interfaz web es la siguiente Figura 3.11.

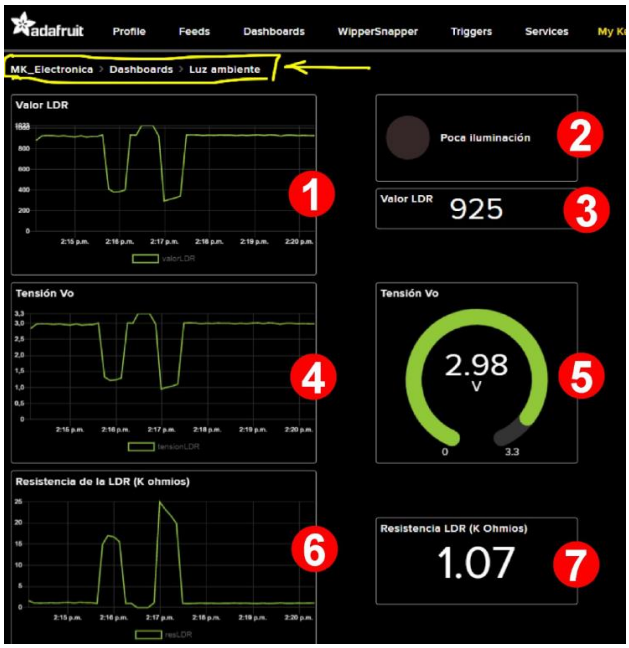


Figura 3.11 *Dashboard* de *Adafruit IO* [9]

Igualmente se podrá mirar en los *feeds* las variables configuradas y en los *dashboard* los equivalentes en un panel de control. Asimismo, en los *dashboard,* se coloca los bloques o *"blocks"* que forman parte del proyecto [9]. En la siguiente Tabla 3.6 se detallan algunas características que tienen cada una de las plataformas.

Tabla 3.6 Comparación de plataformas [9] [26]

Características	***Arduino IoT Cloud***	***Adafruit IO***
Numero de dispositivos	2	2
Número de variables para plan gratuito	20	10
Integración página web y móvil	Si	No
Editor de programación	Si	No

Tras analizar las opciones propuestas, se optó por *Adafruit IO* debido a su facilidad de uso y compatibilidad con el módulo *Wi-Fi LoRa ESP32 Heltec* (V2). Esta plataforma no solo integra otras plataformas, sino que también ofrece diversos métodos de comunicación entre dispositivos, lo cual resulta altamente beneficioso y funcional para el proyecto.

Diseño del diagrama general del sistema

El diseño del prototipo del sistema se basa en la integración de múltiples dispositivos clave para un funcionamiento coordinado, como se ilustra en la Figura 3.12. En primer lugar, se tiene al dispositivo esclavo que es un *Wi-Fi LoRa ESP32 Heltec (V2),* que desempeña un papel esencial al estar conectado tanto a un relé como a un sensor ultrasónico HC-SR04. Este sensor ultrasónico realiza mediciones continuas del nivel de agua y la distancia, proporcionando datos precisos que son enviados al dispositivo maestro que de igual manera es un *Wi-Fi LoRa ESP32 Heltec (V2).*

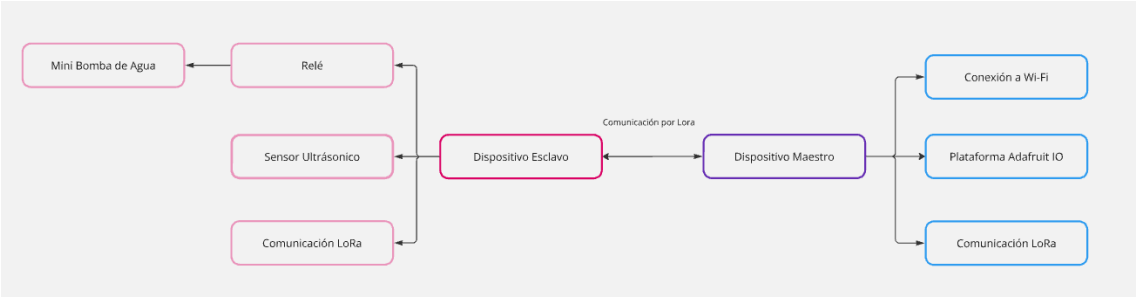


Figura 3.12 Esquema del prototipo de medición de nivel de agua

El dispositivo maestro, por su parte, está equipado con conexión *Wi-Fi* y se conecta a la plataforma *Adafruit IO*. Esta plataforma actúa como un *hub* central donde se visualizan y registran en tiempo real de las variables críticas como la distancia y el nivel de agua. Esta visualización permite un monitoreo remoto y en tiempo real del estado del sistema desde cualquier ubicación con acceso a Internet.

Desde *Adafruit IO* también se envían comandos al dispositivo esclavo para control del relé y envió de parámetros. Estos comandos determinan el encendido y apagado del relé, el cual a su vez activa o desactiva una mini bomba de agua según las condiciones medidas por el sensor ultrasónico como se puede ver en la Figura 3.13. Esta integración permite una gestión automatizada del suministro de agua.

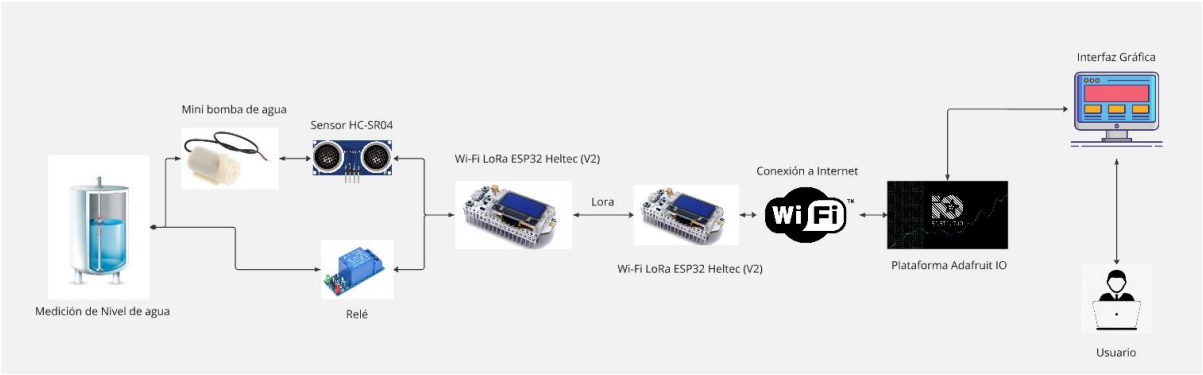


Figura 3.13 Componentes que integran el sistema

Elaboración de códigos

Previamente a la elaboración de código primero se realizó la configuración de *Arduino IDE* y la conexión a la placa *Wi-Fi LoRa ESP32 Heltec (V2)* a continuación se explica las acciones efectuadas.

Configuración del *SoC* en *Arduino IDE*

Primero, se accede a la pestaña "Preferencias" y en la sección "Gestor de URLs adicionales de tarjetas", se debe agregar el URL correspondiente al módulo *Wi-Fi LoRa ESP32 Heltec (V2)*. Luego, se hace clic en "OK", tal como se muestra en la Figura 3.14.

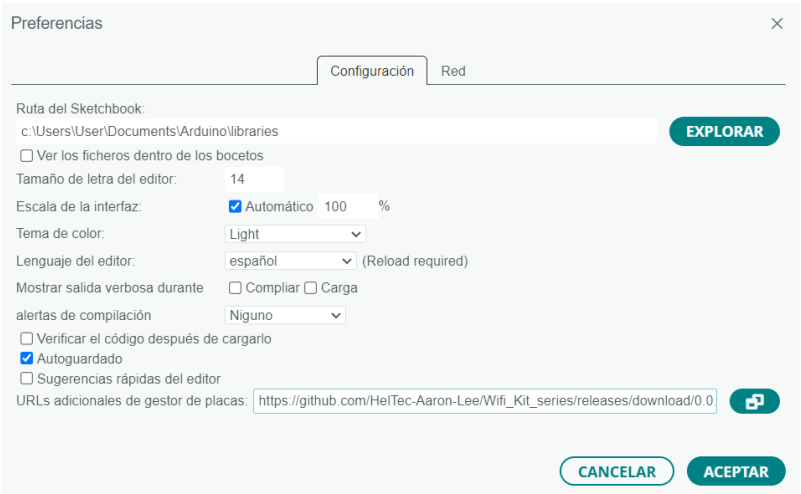


Figura 3.14 Implementación de SoC

Después, se navega a la pestaña "Herramientas", se selecciona la opción "Placa", y luego el "Gestor de Placas", como se muestra en la Figura 3.15.Se debe buscar “Heltec" y seleccionar la opción correspondiente, como se ilustra en la Figura 3.16. Finalmente, se hace clic en "Instalar" para completar el proceso.

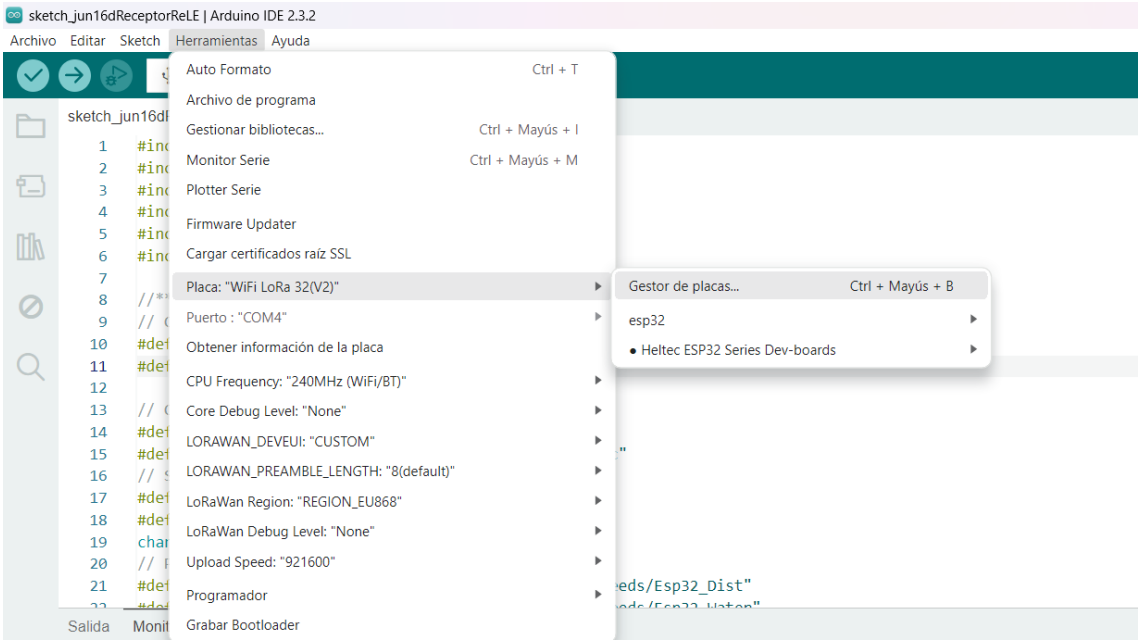


Figura 3.15 Ir a gestor de placas

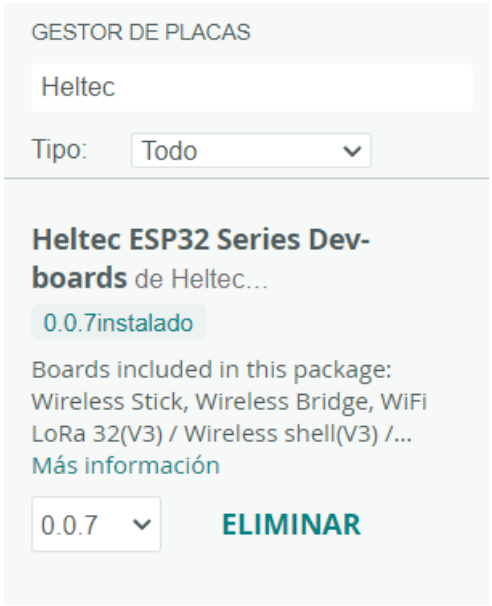


Figura 3.16 Versión más actual

Para conectar el dispositivo se debe regresar a la pestaña "Herramientas" y al seleccionar la opción de "Placa", se deberían mostrar las librerías disponibles para la integración de proyectos con las placas Heltec como se muestra en la Figura 3.17.

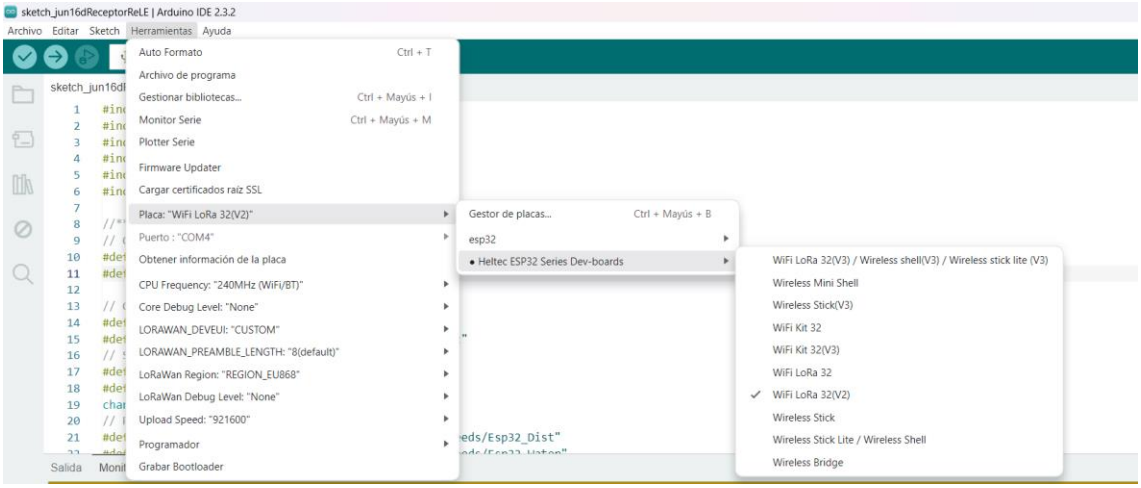


Figura 3.17 Librería instalada

Desarrollo del código para el dispositivo maestro

El código incluye las bibliotecas fundamentales para un dispositivo que combina comunicación *LoRaWAN y MQTT* sobre *Wi-Fi*. Entre ellas, *Arduino.h* incluye funciones para la entrada/salida de datos, el control de tiempo, y la manipulación de pines, facilitando la programación y control de *hardware*. *LoRaWan_APP.h* está diseñada para aplicaciones que requieran comunicación *LoRaWan*.

La biblioteca *WiFi.h* gestiona la conectividad *Wi-Fi* en dispositivos *ESP32*. *PubSubClient.h* facilita la comunicación del protocolo *MQTT* ya sea suscripción y publicación de datos. La biblioteca *Adafruit_Sensor.h* se utiliza para interactuar con sensores y *<Ticker.h>* proporciona funcionalidad de temporización para tareas periódicas. La declaración de las librerías en el código se ilustra en Figura 3.18.

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

Figura 3.18 Definición de bibliotecas

Para conectar a internet, se requiere definir el nombre de la red *Wi-Fi* y su contraseña correspondiente. Este paso resulta fundamental para que el dispositivo pueda acceder a la red y obtener los recursos necesarios. En este caso se puede ver en la Figura 3.19 las credenciales de la red.

```
// Credenciales Red WiFi
#define WIFI_SSID "Alfanet_WTigselema"
#define WIFI_PASSWORD "AMD301812#73"
```

Figura 3.19 Credenciales de Wi-Fi

Para tener conexión con *Adafruit IO* y la nube primero se debe crear una cuenta en esta plataforma y al ingresar en la esquina superior derecha de la interfaz, se encuentran las credenciales que se debe agregar en el código como el nombre de usuario y clave. Además, se añade la dirección a la que el dispositivo se conectará y se declara el puerto que ocupará. Las credenciales de *Adafruit IO* se pueden observar en la Figura 3.20.

```
// Credenciales Adafruit
#define ADAFRUIT_USER "MelIoT"
#define ADAFRUIT_KEY "aio_FstB83A1TplfxDuGGDt8Em3rTsSc"
#define ADAFRUIT_SERVER "io.adafruit.com"
#define ADAFRUIT_PORT 1883
char ADAFRUIT_ID[30];
```

Figura 3.20 Credenciales para Adafruit IO

Después se define las constantes a publicar en este caso es “Distancia”, “Nivel de agua”, “*Relay*”, “Máximo Nivel de agua” y “Mínimo Nivel de agua” esto es para poder identificar en los *feeds* los datos recibidos o enviados desde la plataforma de *Adafruit IO*. En la Figura 3.21 se muestra el código de las variables.

```
// Publicar
#define ADAFRUIT_FEED_Esp32_Dist          ADAFRUIT_USER "/feeds/Esp32_Dist"
#define ADAFRUIT_FEED_Esp32_Water         ADAFRUIT_USER "/feeds/Esp32_Water"
#define ADAFRUIT_FEED_RELAY               ADAFRUIT_USER "/feeds/relay"
#define ADAFRUIT_FEED_MAX_WATER_LEVEL     ADAFRUIT_USER "/feeds/max_water_level"
#define ADAFRUIT_FEED_MIN_WATER_LEVEL     ADAFRUIT_USER "/feeds/min_water_level"
// Suscripción
#define ADAFRUIT_DATA_IN   ADAFRUIT_USER "/feeds/data_in"
```

Figura 3.21 Publicación de variables

A continuación, para este caso específico, se establece una frecuencia de transmisión de 916 (MHz), junto con una potencia de salida adecuada para la transmisión de datos. Además, se define un ancho de banda *LoRa* de 125 (KHz). El factor de expansión *LoRa*, por su parte, permite aumentar la robustez de la señal mediante la repetición de esta.

```
#define RF_FREQUENCY                                916000000 // Hz
#define TX_OUTPUT_POWER                             14        // dBm
#define LORA_BANDWIDTH                              0         // [0: 125 kHz,
                                                              //  1: 250 kHz,
                                                              //  2: 500 kHz,
                                                              //  3: Reserved]
#define LORA_SPREADING_FACTOR                       7         // [SF7..SF12]
#define LORA_CODINGRATE                             1         // [1: 4/5,
                                                              //  2: 4/6,
                                                              //  3: 4/7,
                                                              //  4: 4/8]
#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT                         0         // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON                  false
#define LORA_IQ_INVERSION_ON                        false

#define RX_TIMEOUT_VALUE                            1000
#define BUFFER_SIZE                                 30 // Defina el tamaño de la carga
```

Figura 3.22 Definición de parámetros para comunicación LoRa

Cuenta con una tasa de codificación para *LoRa* y un preámbulo, que es una cadena de símbolos que facilita la sincronización con el receptor. También tiene un tiempo de espera específico, definido en milisegundos, y una longitud de carga fija. La inversión IQ está desactivada (valor false), lo que indica que este método no se está utilizando. Al finalizar, se encuentra con un *buffer* que permite transmitir un máximo de 30 datos por paquete, toda la configuración de los parámetros para *LoRa* se encuentra en la Figura 3.22.

Posteriormente, se declaran las variables en este caso están los *buffer*s que son de tipo carácter, los cuales son para almacenar datos de transmisión y recepción. *Radio Events* sirve para incluir y almacenar los punteros de las funciones que manejarán los diferentes eventos para la comunicación por *LoRa.* Otro parámetro son los *rssi, rxSize* que se presentan en el código son para almacenar la intensidad de la señal recibida en conjunto con el tamaño de datos que obtiene.

```
//*******************************************************
//Declaración de Variables
char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];

static RadioEvents_t RadioEvents;
int16_t rssi, rxSize;

String StrDataLoraIn, StrIndx;
float Val_Dist, Val_Water;
float Esp32_Dist, Esp32_Water;
int pastMaxWaterLevel, pastMinWaterLevel;
int relay, maxWaterLevel, minWaterLevel;
bool relayState = false;
bool lora_idle = true;

Ticker tiempo_1; // Declaración global de la variable Ticker
WiFiClient espClient;
PubSubClient client(espClient);

//*******************************************************
```

Figura 3.23 Declaración de variables

Como se muestra en la Figura 3.23 se implementó una cadena de datos para el control los cuales son *StrDataLoRaIn* y *StrIndx.* También variables de tipo *Float* en estas se guardan las variables que recolectan los datos los cuales son números decimales en este grupo entrar las siguientes: *Val_Dist, Val_Water, Esp32_Dist y Esp32_Water*. Las variables de tipo *Int* estas guardan los datos que son números enteros en esta entra los valores de *pastMaxWaterLevel, pastMinWaterLevel,releley, maxWaterLevey y minWaterLevel.* Y las variables de tipo *Bool* las cuales solo puede tomar valores de Verdadero o Falso y estas son el *relayState* y *LoRa_idle.* La variable *ticker tiempo_1* se utiliza para establecer un temporizador periódico. Las ultimas variables son para conexión a Internet y para que *MQTT* la utilice para comunicación.

El código contiene una serie de declaraciones de funciones que organizan y definen la lógica del programa en el *SoC*. Estas funciones incluyen *getValue,* que divide una cadena basada en un delimitador y devuelve el valor en un índice específico, y funcion_1, que publica datos en *Adafruit IO* cada 5 segundos. Otras funciones, como *sendRelayState, sendMaxWaterLevel, y sendMinWaterLevel*, se encargan de enviar información específica a través de *LoRa*, mientras que *setup_wifi* configura la conexión *Wi-Fi* necesaria para la comunicación *MQTT*.

```
//*******************************************************
//Declaración de Funciones
String getValue(String data, char separator, int index);
void funcion_1();
void sendRelayState();
void sendMaxWaterLevel(String maxWaterLevel);
void sendMinWaterLevel(String minWaterLevel);
void setup_wifi();
void callback(char* topic, byte* payload, unsigned int length);
void reconnect();
void mqtt_publish(String feed, float val);
void mqtt_publish(String feed, String val);
void get_MQTT_ID();

//*******************************************************
```

Figura 3.24 Declaración de funciones

Además, como se ilustra en la Figura 3.24 las funciones como *callback* y *reconnect* manejan la comunicación *MQTT*, procesando mensajes recibidos y reconectando al servidor *MQTT* si la conexión se pierde. La función *MQTT_publish* publica datos en los *feeds* de *Adafruit IO*, tanto en formato de cadena como numérico, y *get_MQTT_ID* genera un identificador único para el dispositivo. Estas funciones permiten al *Wi-Fi LoRa ESP32 Heltec (V2)* gestionar la comunicación y la interacción con la red, facilitando el intercambio de datos y comandos entre dispositivos y servidores.

La función *setup* se requiere para configurar la comunicación serial, conexión *Wi-Fi*, cliente *MQTT*, eventos de radio y módulo *LoRa*, temporizador para convocar la función_1 cada 5 (s). Como se puede visualizar en la Figura 3.25.

```
void setup()
{
    Serial.begin(115200);
    //****
    get_MQTT_ID();
    setup_wifi();
    client.setServer(ADAFRUIT_SERVER, ADAFRUIT_PORT);
    client.setCallback(callback);
    //****
    Mcu.begin();

    rssi = 0;

    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(5, funcion_1); // Correcta llamada a attach
}
```

Figura 3.25 Setup

La función *Loop* a diferencia de la función *Setup* la cual se ejecuta solo cuando se inicializa el programa, la función *Loop* se repite de manera cíclica mientras el *SoC* este encendido o hasta que se interrumpa manualmente el programa. Como se muestra en la Figura 3.26 en esta se encuentra alojada el algoritmo para la reconexión con el cliente *MQTT*. Luego se tiene la función para verificar si el dispositivo *LoRa* no este inactivo.

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

    if (lora_idle)
    {
        lora_idle = false;
        Serial.println("into RX mode");
        Radio.Rx(0);
    }
    Radio.IrqProcess();
}
//*******************************************************
```

Figura 3.26 Función *void loop*

La función funcion_1 realiza varias tareas clave para mantener actualizado el sistema. Primero, publica los valores actuales de distancia (Esp32_Dist) y nivel de agua (Esp32_Water) en los *feeds* correspondientes de *Adafruit IO.* Además, actualiza el estado del relé en el *feed ADAFRUIT_FEED_RELAY* basándose en si *relay* es igual a 1 o no. Para los niveles máximo y mínimo de agua, solo publica los datos si han cambiado desde la última vez, evitando actualizaciones innecesarias. Como se aprecia en la Figura 3.27.

```
void funcion_1()
{
    mqtt_publish(ADAFRUIT_FEED_Esp32_Dist, Esp32_Dist);
    mqtt_publish(ADAFRUIT_FEED_Esp32_Water, Esp32_Water);
    mqtt_publish(ADAFRUIT_FEED_RELAY, relay==1 ?"ON":"OFF");
    if(pastMaxWaterLevel != maxWaterLevel){
    mqtt_publish(ADAFRUIT_FEED_MAX_WATER_LEVEL, maxWaterLevel);
    }
    if(pastMinWaterLevel != minWaterLevel){
    mqtt_publish(ADAFRUIT_FEED_MIN_WATER_LEVEL, minWaterLevel);
    }
    pastMaxWaterLevel = maxWaterLevel;
    pastMinWaterLevel = minWaterLevel;
    sendRelayState();
}
```

Figura 3.27 Funcion_1

Luego, se actualiza las variables de referencia *pastMaxWaterLevel* y *pastMinWaterLevel* con los valores actuales para que las próximas comparaciones detecten cualquier cambio en los niveles de agua. Finalmente, llama a *sendRelayState* para enviar el estado del relé a través de *LoRa*, lo que permite que otros dispositivos en la red *LoRa* reciban esta información. Esta función asegura que tanto los *feeds* de *Adafruit IO* como la comunicación *LoRa* se mantengan actualizados con la información más reciente.

Por otra parte, la función *OnTxDone* se ejecuta cada vez que se completa la transmisión de datos y la función *OnTxTimeout* se activa siempre que el tiempo de espera de la transmisión de datos se agota, como se ve en la Figura 3.28.

```
void OnTxDone(void) {
    Serial.println("TX done......");
    lora_idle = true;
    Radio.Rx(0);  // Habilitar el modo receptor después de la transmisión
}

void OnTxTimeout(void) {
    Radio.Sleep();
    Serial.println("TX Timeout......");
    lora_idle = true;
    Radio.Rx(0);  // Activar el modo receptor después del tiempo de espera.
}
```

Figura 3.28 Manejo de evento de transmisión

La función *OnRxDone* se ejecuta cuando el módulo *LoRa* recibe un paquete de datos. Primero, almacena el contenido del paquete en un *buffer* y lo convierte a una cadena de texto para su procesamiento. Después, desactiva el módulo *LoRa* para ahorrar energía, e imprime en el monitor serial el contenido del paquete, indicando la intensidad de la señal recibida (RSSI) y el tamaño del paquete como se puede visualizar en la Figura 3.29.

```
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)
{
    rssi = rssi;
    rxSize = size;
    memcpy(rxpacket, payload, size);
    rxpacket[size] = '\0';
    Radio.Sleep();
    Serial.printf("\r\nPaquete Recibido \"%s\" con RSSI %d , LONGITUD %d\r\n", rxpacket, rssi, rxSize);
    StrDataLoraIn = String(rxpacket);

    StrIndx = getValue(StrDataLoraIn, '@', 0);
    Val_Dist = (getValue(StrDataLoraIn, '@', 1)).toFloat();
    Val_Water = (getValue(StrDataLoraIn, '@', 2)).toFloat();
    relay = (getValue(StrDataLoraIn, '@', 3)).toInt();
    maxWaterLevel = (getValue(StrDataLoraIn, '@', 4)).toInt();
    minWaterLevel = (getValue(StrDataLoraIn, '@', 5)).toInt();

    if (StrIndx == "ArdEsp32")
    {
        Esp32_Dist = Val_Dist;
        Esp32_Water = Val_Water;
        relayState = relay == 1;
    }

    Serial.print(" Index: "); Serial.println(StrIndx);
    Serial.print(" Val_Dist: "); Serial.println(Val_Dist);
    Serial.print(" Val_Water: "); Serial.println(Val_Water);
    Serial.print(" Relay: "); Serial.println(relay);

    lora_idle = true;
}
```

Figura 3.29 Manejo de evento de recepción

A continuación, la función extrae y procesa varios valores del paquete. Usa la función *getValue* para dividir la cadena en partes basadas en el delimitador '@', y convierte estas partes en valores numéricos apropiados, como flotantes y enteros. Estos valores incluyen datos sobre la distancia, el nivel de agua, y el estado del relé, que se asignan a variables específicas del programa.

Finalmente, la función actualiza variables globales si el índice recibido es *"ArdEsp32"*, y ajusta el estado del relé según el valor recibido. Los valores extraídos se imprimen en el monitor serial para depuración. La función marca el módulo *LoRa* como inactivo, indicando que está listo para recibir el siguiente paquete de datos.

La función *getValue* por otro lado extrae un valor de una cadena (data) que está separada por un carácter específico (separator). Recorre la cadena y cuenta cuántos separadores encuentra hasta llegar al índice deseado (index). Si encuentra el valor en el índice especificado, devuelve la subcadena correspondiente; si no, devuelve una cadena vacía. Como se ilustra en la Figura 3.30.

```
// Se utiliza para dividir una cadena de texto en partes usando un separador y devolver una parte específica basada en el índice.
String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        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]) : "";
}
```

Figura 3.30 División de cadena de datos

Las funciones *MQTT_publish* en el código se encargan de publicar mensajes en un servidor *MQTT*, manejando tanto valores flotantes como cadenas de texto. La primera función convierte un valor flotante en una cadena con dos decimales para asegurar un formato consistente y luego publica este valor en el tópico especificado, siempre que el cliente *MQTT* esté conectado. También imprime el tópico y el mensaje en el monitor serial para facilitar la depuración.

La segunda función pública directamente una cadena de texto en el tópico, con una verificación previa para asegurar que el cliente *MQTT* esté conectado. Al igual que la primera función, también imprime en el monitor serial el tópico y el mensaje que se están enviando, lo que ayuda a verificar y depurar el proceso de publicación de mensajes. Como se observa en la Figura 3.31.

```
void mqtt_publish(String feed, float val)
{
    String value = String(val, 2);  // Formatear a dos decimales
    if (client.connected()) {
        client.publish(feed.c_str(), value.c_str());
        Serial.println("Publicando al tópico: " + String(feed) + " | mensaje: " + value);
    }
}

void mqtt_publish(String feed, String val)
{
    if (client.connected()) {
        client.publish(feed.c_str(), val.c_str());
        Serial.println("Publicando al tópico: " + String(feed) + " | mensaje: " + val);
    }
}
```

Figura 3.31 Pública en MQTT

Las funciones que están diseñadas para enviar datos por medio de comunicación *LoRa* es la función *sendRelayState()* se encarga de enviar el estado actual del relé, formando un mensaje que indica si está activado (*relayState* igual a true) o desactivado (*relayState* igual a false). Luego, transmite este mensaje a través de la interfaz *LoRa* y muestra el estado actual del relé en el puerto serial.

```
// Función para enviar el estado del relé por LoRa
void sendRelayState()
{
    snprintf(txpacket, BUFFER_SIZE, "relay@%d", relayState ? 1 : 0);
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    Serial.println("Enviando estado del relé: " + String(relayState));
}

void sendMaxWaterLevel(String maxWaterLevel)
{
    snprintf(txpacket, BUFFER_SIZE, "max_water_level@%s", maxWaterLevel);
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    Serial.println("Enviando nivel máximo del agua: " + maxWaterLevel);
}

void sendMinWaterLevel(String minWaterLevel)
{
    snprintf(txpacket, BUFFER_SIZE, "min_water_level@%s", minWaterLevel);
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    Serial.println("Enviando nivel mínimo del agua: " + minWaterLevel);
}
```

Figura 3.32 Datos enviados por LoRa

Por otro lado, como se puede observar en la Figura 3.32 las funciones *sendMaxWaterLevel()* y *sendMinWaterLevel()* tienen una estructura similar: construyen

mensajes que contienen los niveles máximo y mínimo de agua especificados como argumentos. Estos mensajes se envían a través del *SoC* por medio de *LoRa* y se imprime el nivel correspondiente en el puerto serial para confirmar la transmisión exitosa. Estas funciones son fundamentales para el control de dispositivos mediante comunicación *LoRa*, facilitando la gestión y visualización de datos críticos como los niveles de agua y el estado del relé.

La función *setup_wifi()* está diseñada para conectar el dispositivo maestro a una red *Wi-Fi* utilizando el nombre de red (SSID) y la contraseña proporcionados. Comienza con un breve retraso antes de iniciar el proceso de conexión, mostrando en el puerto serial el nombre de la red a la que intenta conectarse. Durante el proceso, utiliza un bucle para esperar hasta que la conexión sea establecida (WL_CONNECTED), mostrando puntos para indicar progreso. En la Figura 3.33 se puede apreciar de mejor manera como es la programación.

```
//*******************************************************
//  CONEXION A INTERNET
void setup_wifi()
{
    delay(10);

    // Conexión a 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());
}
//*******************************************************
```

Figura 3.33 Conexión a internet

Una vez conectado, imprime mensajes indicando el éxito de la conexión y muestra la dirección IP asignada al dispositivo por la red *Wi-Fi*. Esta función facilita la comunicación el intercambio de datos a través de redes *Wi-Fi*, asegurando una conexión estable y confiable.

La función *callback* en cambio procesa mensajes recibidos a través de *MQTT*, como se ilustra en la Figura 3.34 convirtiendo el tópico y el mensaje a *String* y mostrándolos en el puerto serial. Si el tópico coincide con *ADAFRUIT_FEED_RELAY*, ajusta el estado del relé (*relayState)* según si el mensaje es *"ON" o "OFF"* y llama a *sendRelayState ()* para enviar el estado actual.

```
// Función para capturar data por MQTT
void callback(char *topic, byte *payload, unsigned int length)
{
    String mensaje = "";
    String str_topic(topic);

    for (uint16_t i = 0; i < length; i++) {
        mensaje += (char)payload[i];
    }

    mensaje.trim();

    Serial.println("Tópico Ada: " + str_topic);
    Serial.println("Mensaje Ada: " + mensaje);

    if (str_topic == ADAFRUIT_FEED_RELAY) {
        if (mensaje == "ON") {
            relayState = true;
        } else if (mensaje == "OFF") {
            relayState = false;
        }
        sendRelayState();
    }

    if (str_topic == ADAFRUIT_FEED_MAX_WATER_LEVEL) {
        sendMaxWaterLevel(mensaje);
    }
    if (str_topic == ADAFRUIT_FEED_MIN_WATER_LEVEL) {
        sendMinWaterLevel(mensaje);
    }
}
```

Figura 3.34 Acciones basadas en el tópico

Si el tópico es *ADAFRUIT_FEED_MAX_WATER_LEVEL* o *ADAFRUIT_FEED_MIN_WATER_LEVEL*, llama a *sendMaxWaterLevel* (mensaje) y *sendMinWaterLevel* (mensaje) respectivamente, para transmitir los niveles máximo y mínimo del agua.

La función *get_MQTT_ID ()* genera un ID único para el dispositivo utilizando el chip ID del *ESP32* y lo almacena en la variable *ADAFRUIT_ID*. La función *reconnect()* se encarga de reconectar al cliente *MQTT* si no está conectado, usando el ID generado, nombre de usuario y clave. Si la conexión es exitosa, imprime un mensaje de confirmación y se suscribe a varios tópicos específicos *(ADAFRUIT_DATA_IN, ADAFRUIT_FEED_RELAY, ADAFRUIT_FEED_MAX_WATER_LEVEL, ADAFRUIT_FEED_MIN_WATER_LEVEL)*. Si la conexión falla, imprime el estado del cliente y reintenta la conexión después de 5 segundos. Como se puede observar en la Figura 3.35.

```
// Capturar el ChipID para Id de MQTT
void get_MQTT_ID()
{
    uint64_t chipid = ESP.getEfuseMac();
    snprintf(ADAFRUIT_ID, sizeof(ADAFRUIT_ID), "%llu", chipid);
}

void reconnect()
{
    while (!client.connected())
    {
        if (client.connect(ADAFRUIT_ID, ADAFRUIT_USER, ADAFRUIT_KEY))
        {
            Serial.println("MQTT conectado!");
            client.subscribe(ADAFRUIT_DATA_IN);
            client.subscribe(ADAFRUIT_FEED_RELAY);
            client.subscribe(ADAFRUIT_FEED_MAX_WATER_LEVEL);
            client.subscribe(ADAFRUIT_FEED_MIN_WATER_LEVEL);
            Serial.println("Suscrito a los tópicos: " + String(ADAFRUIT_DATA_IN) + ", " + String(ADAFRUIT_FEED_RELAY) +", "+ String(ADAFRUIT_FEED_MAX_WATER_LEVEL)+", "+ String(ADAFRU
        }
        else
        {
            Serial.print("falló :( con error -> ");
            Serial.print(client.state());
            Serial.println(" Intentamos de nuevo en 5 segundos");
            delay(5000);
        }
    }
}
```

Figura 3.35 Captura el id del dispositivo

Desarrollo del código para el dispositivo esclavo

Primero se tiene la librería de *LoRaWan_APP.h* para la comunicación con el dispositivo maestro por medio de *LoRa*. A continuación, se tiene la librería de *Arduino.h* la cual sirve para la interacción del lenguaje de programación C++ y el *hardware*. A su vez, la librería *NewPing.h* tiene el propósito del control del sensor ultrasónico HC-SR04. Las librerías declaradas se ilustran en la Figura 3.36.

```
//Librerias
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <NewPing.h>
```

Figura 3.36 Librerías del dispositivo esclavo

Se definen tres pines específicos para conectar componentes externos a la placa del *SoC*. El *TRIG_PIN* (pin 12) está destinado a emitir una señal para activar el sensor ultrasónico HC-SR04, mientras que el *ECHO_PIN* (pin 13) recibe el eco de la señal reflejada para medir la distancia. El *RELAY_PIN* (pin 2) controla un relé que puede activar o desactivar dispositivos externos. Estas definiciones facilitan el manejo de *hardware* al utilizar nombres descriptivos en lugar de números de pines, mejorando la claridad del código. Los pines declarados se muestran en la Figura 3.37.

```
// Definiciones de los pines conectados para el sensor HC-SR04 y Relé
#define TRIG_PIN 12  // Pin digital para el TRIG
#define ECHO_PIN 13  // Pin digital para el ECHO
#define RELAY_PIN 2 // Pin para el relé
```

Figura 3.37 Declaración de pines

Seguido se define las constantes. Como la velocidad del sonido esto permite la calibración para el sensor HC-SR04. Configuración del módulo *LoRa*: la frecuencia RF está establecida en 916 (MHz), con una potencia de emisión de 14 (dBm) y un ancho de banda de 125 (KHz). Utiliza un factor de expansión SF7 y una tasa de codificación de 4/5, con una longitud de preámbulo 8. Como se ilustra en la Figura 3.38.

```
//*******************************************************************
// Definicion de Constantes
#define SOUND_SPEED 0.034  // Velocidad del sonido en cm/uS

#define RF_FREQUENCY 916000000 // Hz

#define TX_OUTPUT_POWER 14        // dBm

#define LORA_BANDWIDTH 0         // [0: 125 kHz,
                                 //  1: 250 kHz,
                                 //  2: 500 kHz,
                                 //  3: Reserved]
#define LORA_SPREADING_FACTOR 7  // [SF7..SF12]
#define LORA_CODINGRATE 1        // [1: 4/5,
                                 //  2: 4/6,
                                 //  3: 4/7,
                                 //  4: 4/8]
#define LORA_PREAMBLE_LENGTH 8   // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT 0    // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON false
#define LORA_IQ_INVERSION_ON false

#define RX_TIMEOUT_VALUE 1000
#define BUFFER_SIZE 30 // Define el tamaño del payload

//*******************************************************************
```

Figura 3.38 Definición de constantes

El tiempo de espera de símbolo está desactivado, así como la longitud fija del *payload* y la inversión de IQ. El tiempo de espera para recepción es de 1000 (ms) y el tamaño del *buffer* es de 30 (bytes). Estos parámetros proporcionan una configuración equilibrada para la comunicación *LoRa*.

Se declaran diversas variables y funciones que permiten gestionar la comunicación y el control de *hardware* en un sistema. Las variables como *txpacket* y *rxpacket* son *buffers* que almacenan los datos transmitidos y recibidos, mientras que *StrDataLoRaIn y StrIndx*

manejan cadenas relacionadas con los datos de *LoRa*. Además, se definen variables para configurar niveles de agua *(maxWaterLevel y minWaterLevel)*, el estado del relé, y para almacenar valores de distancia y agua. Como se indica en la Figura 3.39 la variable *LoRa_idle* indica si el módulo *LoRa* está inactivo, lo cual es crucial para manejar la recepción y transmisión de datos de manera eficiente.

```
//*****************************************************************
//Variables y Funciones
char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];
String StrDataLoraIn, StrIndx;

double txNumber;

bool lora_idle = true;

// filtros
int maxWaterLevel = 85;
int minWaterLevel = 5;
int relay = 0;
float Val_Dist, Val_Water = 0;

static RadioEvents_t RadioEvents;
void OnTxDone(void);
void OnTxTimeout(void);
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr);
void turnOnRelay();
void turnOffRelay();
//*****************************************************************
```

Figura 3.39 Variables y funciones

Se establece un conjunto de funciones para gestionar los eventos de comunicación y controlar el *hardware*. Las funciones *OnTxDone, OnTxTimeout, y OnRxDone* se emplean para gestionar los sucesos vinculados a la transferencia de datos a través del módulo *LoRa,* asegurando que los datos se procesen adecuadamente y se manejen errores. También se declaran funciones para encender y apagar el relé *(turnOnRelay y turnOffRelay)*, permitiendo controlar los dispositivos en función de los datos recibidos.

Estas declaraciones preparan el programa para una integración efectiva con el módulo *LoRa* y otros componentes de *hardware*. Al definir claramente las variables y funciones, el código facilita la gestión de datos y el control de dispositivos, lo que permite una operación robusta y flexible.

La función *setup* es la encargada de ejecutarse una sola vez al iniciar el programa y se encarga de tareas esenciales como establecer la comunicación serial a una velocidad de 115200 (baudios), inicializar el *SoC* y configurar un contador de transmisiones para registrar el número de datos enviados. Como se muestra en la Figura 3.40 se configura el pin de salida del relé y que este inicie en estado apagado.

Luego se programó los eventos para la recepción de paquetes, estos se ejecutan cuando se completa la transmisión y cuando falla la transmisión debido al tiempo de espera. A continuación de igual manera se inicializa los eventos configurados a la frecuencia requerida. Y se configuran los parámetros para transmisión y recepción para la comunicación *LoRa.*

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

    txNumber = 0;

    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, LOW); // Inicializa el relé apagado


    RadioEvents.RxDone = OnRxDone;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;


    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);

    // Inicialización de los pines del sensor HC-SR04
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);

    Radio.Rx(0);  // Habilitar el modo receptor al inicio
}
```

Figura 3.40 Función *setup*

Por otra parte, se inicializan los pines para el sensor HC-SR04 el pin *trigger* para salida y el pin *echo* para la entrada. Además, habilita el modo receptor para el inicio del programa.

```
void loop() {
    if (lora_idle == true) {
        Radio.Rx(0);  // Habilitar continuamente el modo receptor
    }
    Radio.IrqProcess();
}
```

Figura 3.41 Función *void loop*

La función *loop* verifica si *LoRa_idle* es true y, de ser así, habilita el modo de recepción tal y como se muestra en la Figura 3.41 se continua en el módulo de radio con *Radio.Rx(0)*. Luego, siempre llama a *Radio.IrqProcess* para procesar las interrupciones del módulo de radio en cada iteración del bucle.

```
void turnOnRelay(){
  relay=1;
  digitalWrite(RELAY_PIN, HIGH); // Encender el relé
  Serial.println("Relé encendido");
}

void turnOffRelay(){
  relay=0;
  digitalWrite(RELAY_PIN, LOW); // Apagar el relé
  Serial.println("Relé apagado");
}
```

Figura 3.42 Control del relé

Como se puede ver en la Figura 3.42 las funciones *turnOnRelay* y *turnOffRelay* controlan al relé. La función *turnOnRelay* enciende el relé, estableciendo la variable *relay* en 1, poniendo en alto el pin correspondiente (*RELAY_PIN*) y enviando un mensaje al monitor serial indicando que el relé está encendido. La función *turnOffRelay* apaga el relé, estableciendo la variable *relay* en 0, poniendo en bajo el pin correspondiente y enviando un mensaje al monitor serial indicando que el relé está apagado.

Por otro lado, la función *OnRxDone* se encarga de procesar paquetes de datos recibidos a través del módulo de radio. Como se aprecia en la Figura 3.43 al recibir un paquete, la función copia los datos del paquete en un *buffer*, agrega un carácter nulo al final para tratarlo como una cadena de texto, y coloca el módulo de radio en modo de sueño para conservar energía. Luego, imprime información sobre el paquete recibido y convierte los datos en una cadena de texto para su posterior análisis.

Una vez convertidos los datos a una cadena, la función extrae los comandos y eventos relevantes. Dependiendo del tipo de evento, puede encender o apagar el relé, o ajustar los niveles máximos y mínimos de agua. Estos comandos se procesan de acuerdo con los valores extraídos del paquete recibido, ajustando el comportamiento del sistema según la información contenida en el paquete.

```
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)
{
    memcpy(rxpacket, payload, size);
    rxpacket[size] = '\0';
    Radio.Sleep();
    Serial.printf("\r\nPaquete Recibido \"%s\" con RSSI %d , LONGITUD %d\r\n", payload, rssi, size);

    int command = 0;
    String event;

    StrDataLoraIn = String(rxpacket);

    event = getValue(StrDataLoraIn, '@', 0);
    command = (getValue(StrDataLoraIn, '@', 1)).toFloat();

    if(event == "relay"){
      if (command == 1) {
          turnOnRelay();
      } else if (command == 0) {
          turnOffRelay();
      }
    }
    if(event == "max_water_level"){
      maxWaterLevel = command;
    }
    if(event == "min_water_level"){
      minWaterLevel = command;
    }
```

```
    // Medición de la distancia utilizando el HC-SR04
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    long duration = pulseIn(ECHO_PIN, HIGH);
    Val_Dist = float(duration / 2.0) * SOUND_SPEED; // Convertir duración en cm
    // Calcular el nivel de agua basado en la distancia
    if (Val_Dist <= 2) {
        Val_Water = 100.0; // Nivel de agua lleno
    } else if (Val_Dist >= 16) {
        Val_Water = 0.0; // Nivel de agua vacío
    } else {
        // Interpolación lineal para calcular el nivel de agua entre 2 cm y 16 cm
        Val_Water = 100.0 - ((Val_Dist - 2) / (16 - 2) * 100.0);
    }

    if (isnan(Val_Dist) || isnan(Val_Water)) {
        Val_Dist = 0;
        Val_Water = 0;
        Serial.println(F("Error de lectura del sensor HC-SR04!"));
    }

    if(Val_Water >= maxWaterLevel){
      turnOffRelay();
    }else if(Val_Water <= minWaterLevel){
      turnOnRelay();
    };
```

```
    Serial.print("Val_Dist: ");
    Serial.print(Val_Dist);
    Serial.print(" cm  Val_Water: ");
    Serial.print(Val_Water);
    Serial.println(" %");

    Serial.printf("Relay status: %d\t MaxWaterLevel: %d\t MinWaterLevel: %d", relay, maxWaterLevel, minWaterLevel);


    // Crear el paquete con el formato requerido por el receptor
    sprintf(txpacket, "ArdEsp32@%.2f@%.2f,@%d@%d@%d", Val_Dist, Val_Water,relay,maxWaterLevel,minWaterLevel);
    Serial.printf("\r\nEnviando Paquete \"%s\" , longitud %d\r\n", txpacket, strlen(txpacket));
    Radio.Send((uint8_t *)txpacket, strlen(txpacket)); // Enviar el paquete
    lora_idle = false;
}
```

Figura 3.43 Función *OnRxDone*

Después de procesar los comandos, la función mide la distancia usando un sensor ultrasónico HC-SR04 para evaluar el nivel de llenado del contenedor. La distancia medida se convierte en un porcentaje de nivel de agua mediante interpolación lineal. Si la distancia o el nivel de agua son inválidos, se ajustan a cero y se reporta un error. La función utiliza esta información para controlar el estado del relé, encendiéndolo o apagándolo según los niveles de agua calculados y los umbrales establecidos.

Finalmente, la función crea un paquete de datos con la información actual, incluyendo la distancia medida, el nivel de agua, el estado del relé y los niveles máximos y mínimos de agua. Imprime detalles sobre el paquete que se enviará para fines de depuración y usa el módulo de radio para enviarlo. Al finalizar, marca el módulo de radio como no inactivo para prepararlo para futuras comunicaciones.

```
void OnTxDone(void) {
    Serial.println("TX done......");
    lora_idle = true;
    Radio.Rx(0);  // Habilitar el modo receptor después de la transmisión
}

void OnTxTimeout(void) {
    Radio.Sleep();
    Serial.println("TX Timeout......");
    lora_idle = true;
    Radio.Rx(0);  // Habilitar el modo receptor después del tiempo de espera
}
```

Figura 3.44 Eventos de emisión y recepción

Las funciones *OnTxDone* y *OnTxTimeout* manejan eventos de transmisión en dispositivos *LoRa*WAN. *OnTxDone* se ejecuta cuando la transmisión es exitosa, imprimiendo un mensaje en el monitor serial, marcando el dispositivo como inactivo (*LoRa*_idle = *true*) y poniendo el dispositivo en modo receptor *Radio.Rx(0)*. *OnTxTimeout* se ejecuta cuando la transmisión falla por un tiempo de espera, pone el dispositivo en modo de suspensión *Radio.Sleep()*, imprime un mensaje de error en el

monitor serial, marca el dispositivo como inactivo y también lo pone en modo receptor como se puede visualizar en la Figura 3.44. Ambas funciones aseguran que el dispositivo esté listo para recibir datos después de sus respectivos eventos.

Implementación de la interfaz web

Para desarrollar la interfaz web, el primer paso es registrarse en *Adafruit IO* tal y como se muestra en la Figura 3.45. A fin de proceder con el registro, es necesario acceder a la sección correspondiente en el sitio web de *Adafruit IO* y suministrar los datos requeridos. Estos incluyen nombre completo, una dirección de correo electrónico activa, un nombre de usuario exclusivo dentro de la plataforma y una contraseña robusta. Este proceso asegurará que la cuenta se configuro correctamente para acceder a todas las funcionalidades que *Adafruit IO* ofrece.

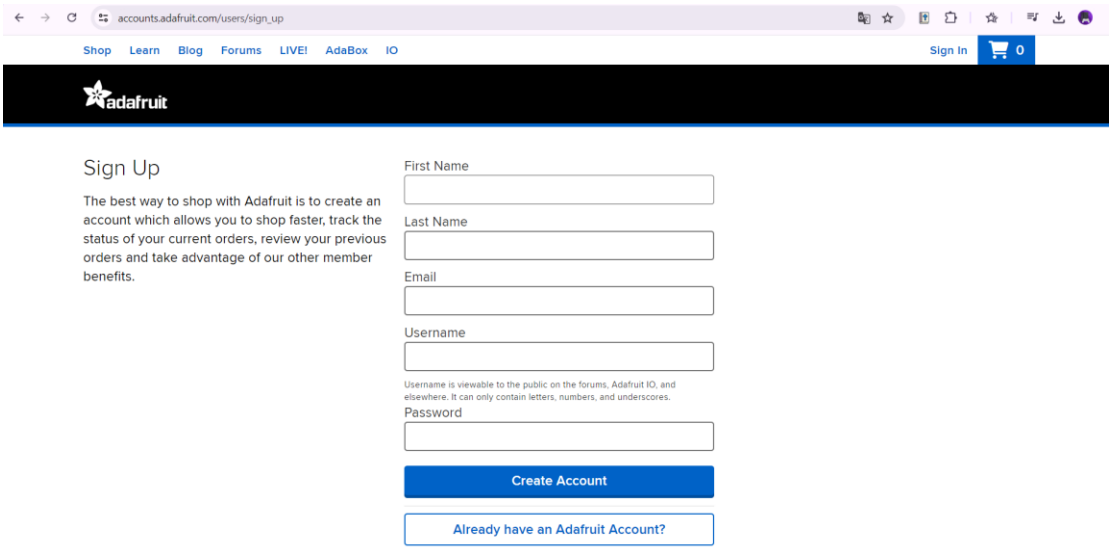


Figura 3.45 Registro en la plataforma *Adafruit IO*

El siguiente paso es iniciar sesión. Dirigiéndose a la página de ingreso de *Adafruit IO*. Cómo se puede observar en la Figura 3.46. En esta sección, se encontrará campos designados para ingresar la dirección de correo electrónico y la contraseña que se registró durante el proceso de inscripción.

Es necesario introducir la dirección de correo electrónico en el espacio destinado y luego escribir la contraseña en el siguiente campo. Hay que asegurarse de verificar que la información sea correcta para evitar problemas de autenticación. Finalizado el ingreso

de información, se procede a dar clic en el botón correspondiente para acceder a la cuenta.

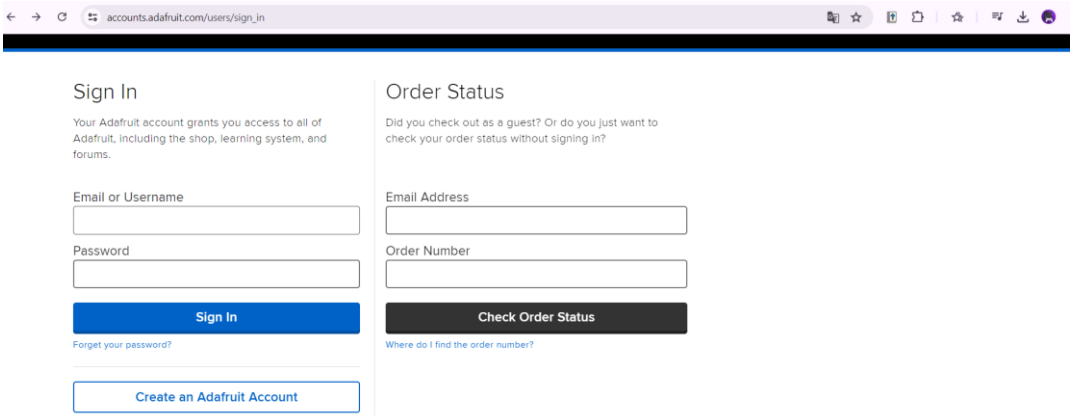


Figura 3.46 Ingreso a la plataforma *Adafruit IO*

Antes de la creación del *dashboard* primero se debe agregar las variables, primero se accede a la pestaña *IO* y luego se selecciona *feeds.* En esta sección, se encontrará un ícono con la opción "*New Feed*". Al hacer clic en este ícono, se podrá configurar el nombre del *feed* y añadir una breve descripción que explique su propósito tal y como se muestra en la Figura 3.47.

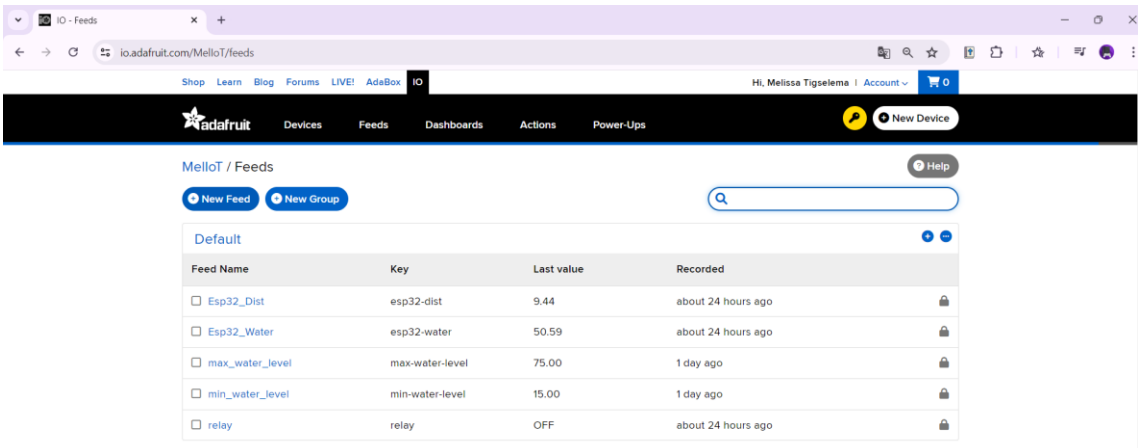


Figura 3.47 Creación de *feeds*

Para crear el *dashboard*, hay que dirigirse a la opción "*New Dashboard*" en el menú y seleccionar este apartado para comenzar. Se debe asignar un nombre adecuado al proyecto, asegurándose de que refleje claramente el propósito y objetivos. Este proceso permite personalizar y organizar la información de manera que sea fácil tal y como se muestra en la Figura 3.48.

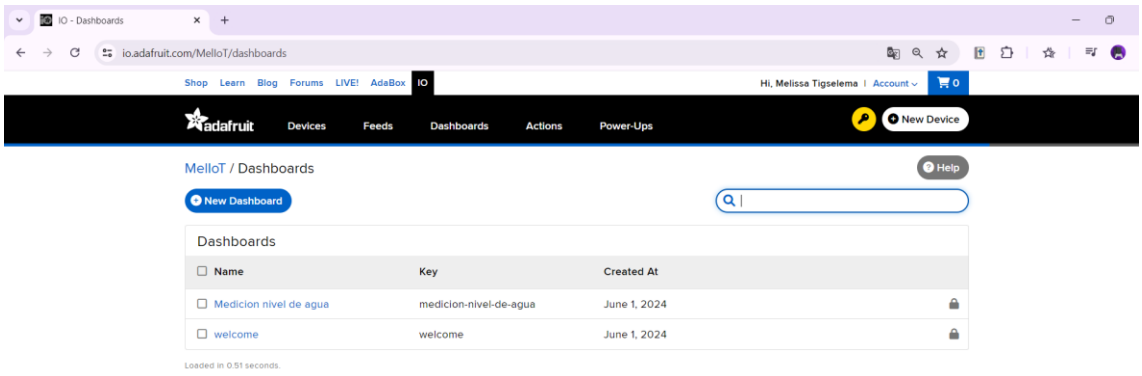


Figura 3.48 Creación de *dashboard*

Por otro lado, para crear los bloques donde se visualizarán las variables configuradas, se debe hacer clic en el ícono de la rueda de configuración. Y seleccionar la opción *"Create New Block"*. Así como se puede ver en la figura Figura 3.49.

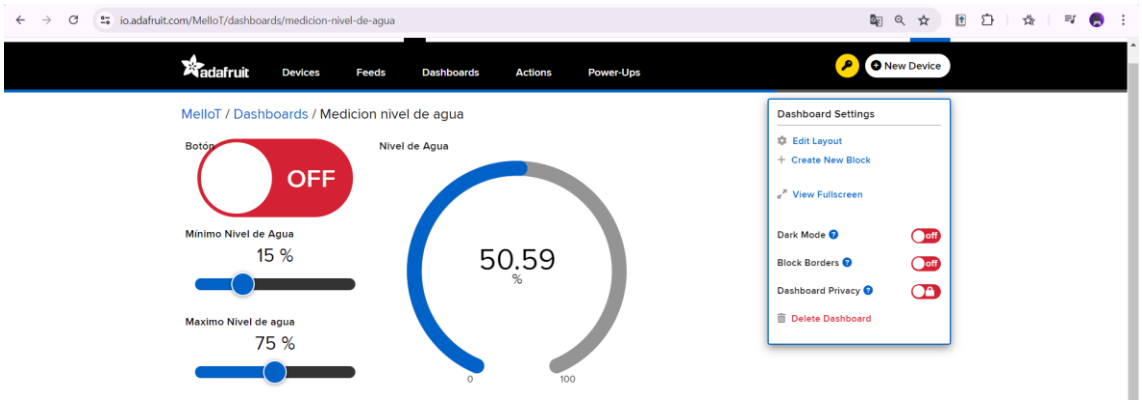


Figura 3.49 Creación de *block*

Después de eso, se despliega una lista de bloques disponibles que pueden ser utilizados. La elección del bloque adecuado depende del tipo de gráfico que se desea mostrar en el *dashboard* en este caso seleccioné el block número cuatro ya que con ese se puede evidenciar la distancia recorrida, como se puede apreciar en la Figura 3.50.

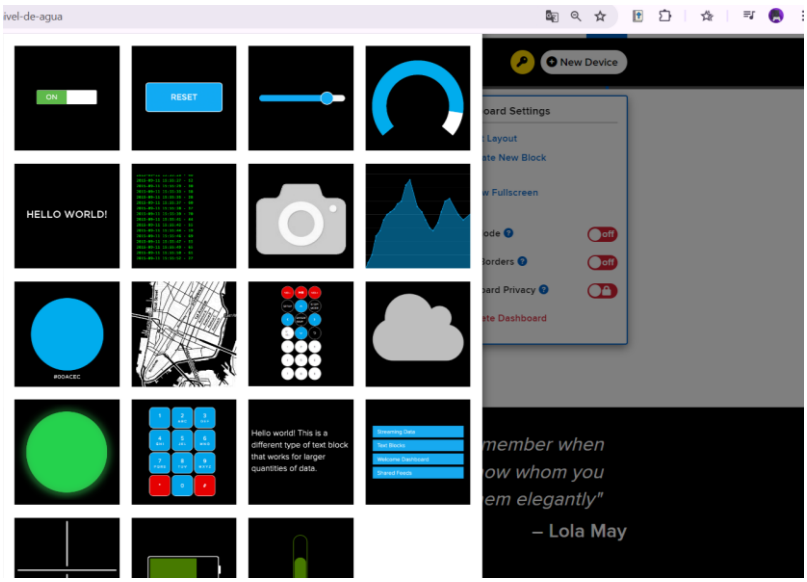


Figura 3.50 Tipos de *blocks*

Una vez seleccionado el *block*, aparecerá un cuadro de diálogo que mostrará los distintos *feeds* configurados, los cuales se deben enlazar con los *blocks* deseados. En este caso se escogió *Esp32_Dist* que es la variable que se va a configurar en ese *block*, como se ve en la Figura 3.51.

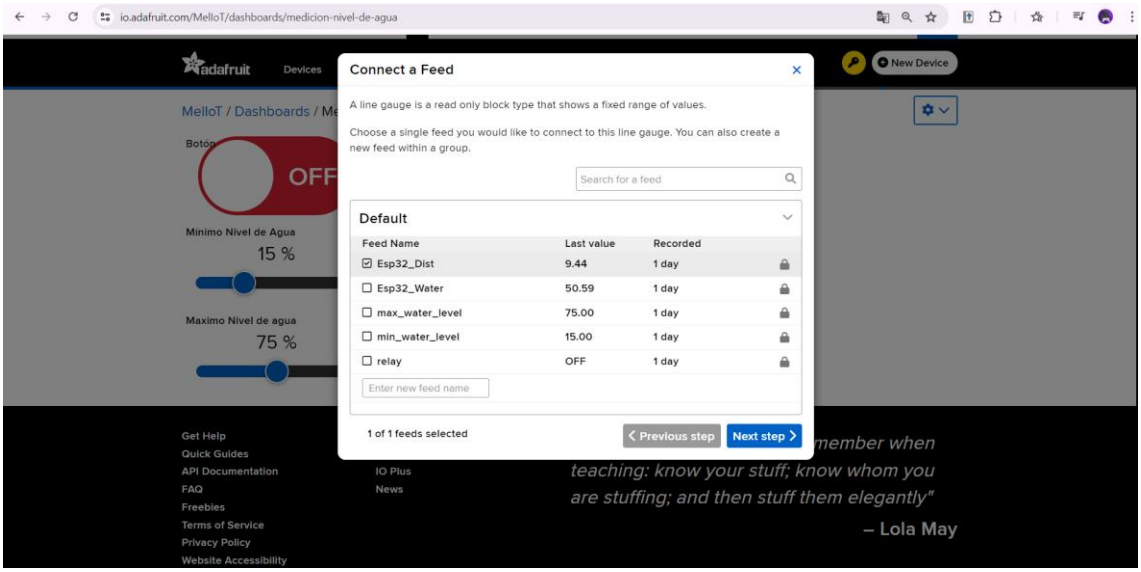


Figura 3.51 Elección de *feed* para el *block*

Una vez seleccionado el *block* y el *feed*, es crucial configurar su nombre y ajustar los parámetros específicos. En este contexto, se calibra el bloque estableciendo el valor mínimo en 0 y el máximo en 100 (cm) esto son los valores que se configuraron para su funcionamiento al momento de medir nivel de agua, lo cual refleja la distancia que el sensor mide respecto al agua y se visualiza en este *block* es de gran importancia ya que

se puede ver que tan cerca esta del sensor el agua. Como se ilustra en la Figura 3.52 después de actualizar estos valores, se guarda la configuración en el *dashboard*.

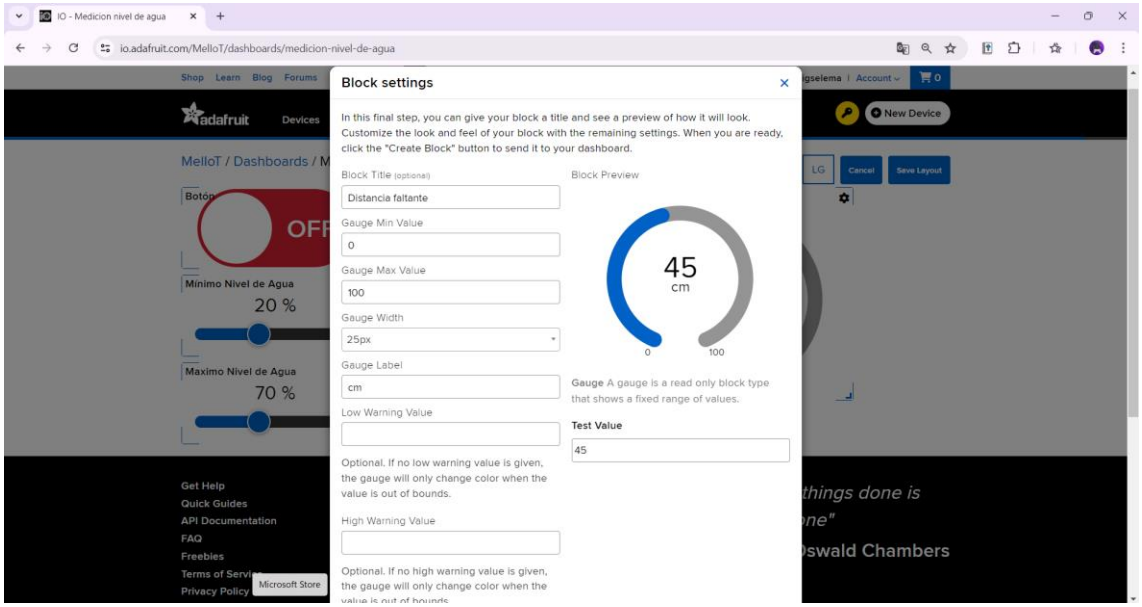


Figura 3.52 Configuración de parámetros *block* distancia

Por consiguiente, como se visualiza en la Figura 3.53 es como se crea el *block* de a Distancia faltante para el *dashboard* en el proyecto de medición de nivel de agua.

Figura 3.53 *Block* distancia

De igual manera para el siguiente *block* se debe dar click en la rueda, elegir "*create new block*", se desplegará varios tipos de gráfico como se visualiza en la Figura 3.50. Para este *block* se escogió la cuarta imagen que permite percibir de una mejora manera los datos recibidos. Se abre el cuadro de dialogo y en este se presenta el *feed* para entrelazar con el *block* es el *Esp32_Water* la cual se ilustra en la Figura 3.54.

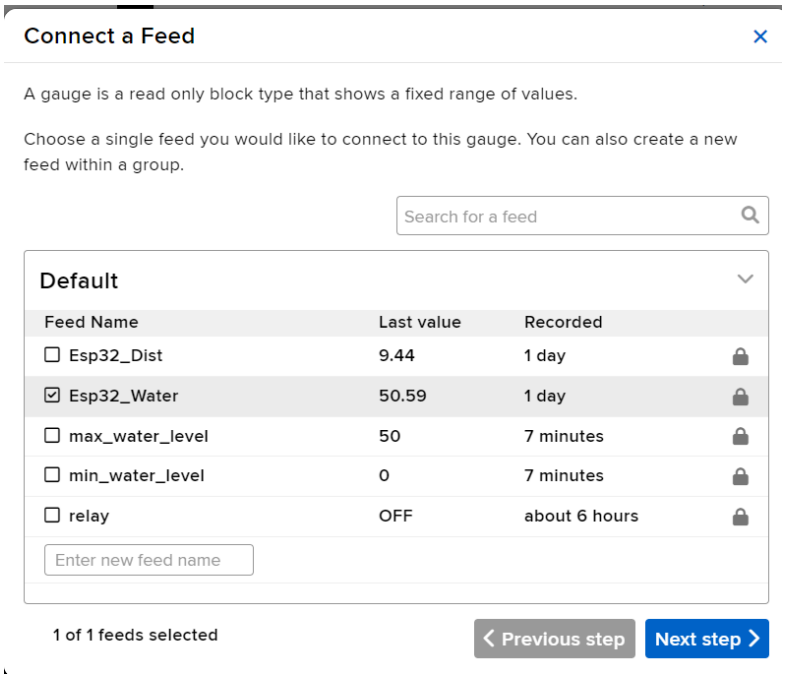


Figura 3.54 Elección de *feed esp32_water*

Una vez seleccionado el *block* se realiza la configuración de parámetros en este caso se le configuro el nombre el cual es nivel de agua el mínimo y máximo valor al que debe llegar este *block* que es de 0 a 100 este es medido en porcentaje tal y como se muestra en la Figura 3.55. Luego se procede a guardar el *block*.

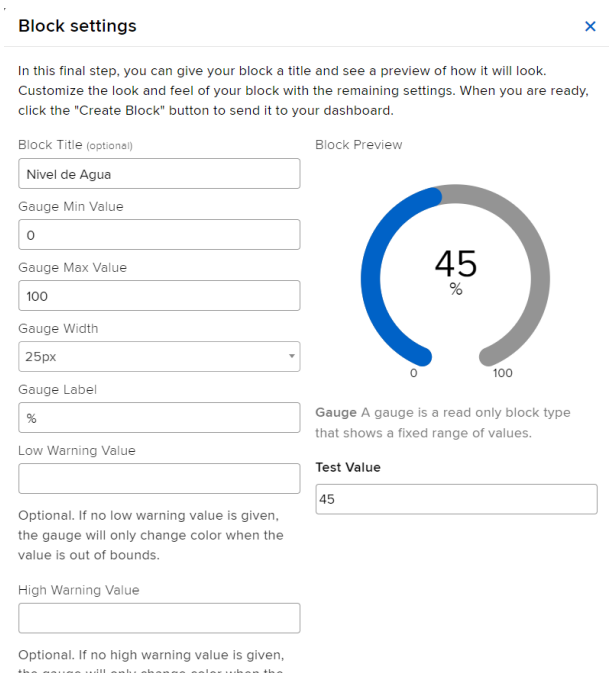


Figura 3.55 Configuración de parámetro de nivel de agua

Luego se realiza una verificación del *block* para ver si funciona de la forma adecuada este se puede visualizar en la Figura 3.56.

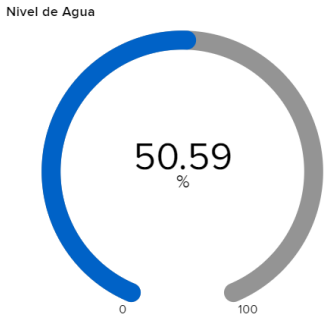


Figura 3.56 *Block* de nivel de agua

El siguiente *block* para configurar es el del relé, se escoge el tipo de grafico a visualizar en este caso se escogió el *block* número uno presentado en la Figura 3.50 debido a que se requiere que tome dos valores ya sea "*ON*" para el prendido de la mini bomba o "*OFF*" para el apagado. Después se entrelaza con el *feed* de *relay* el cual se observa en la Figura 3.57.

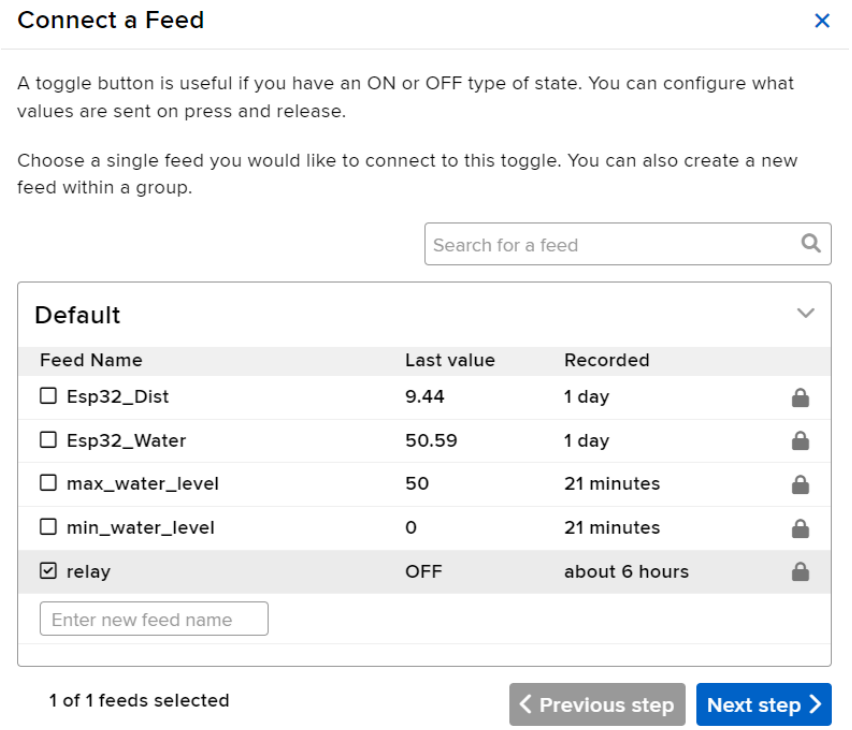


Figura 3.57 Elección de *feed relay*

Tras seleccionar el bloque a utilizar, se le asigna un nombre descriptivo, en este caso "Botón". Posteriormente, se configuran sus parámetros, tal como se muestra en la Figura 3.58, donde se establecen los valores "*ON*" y "*OFF*" respectivamente.

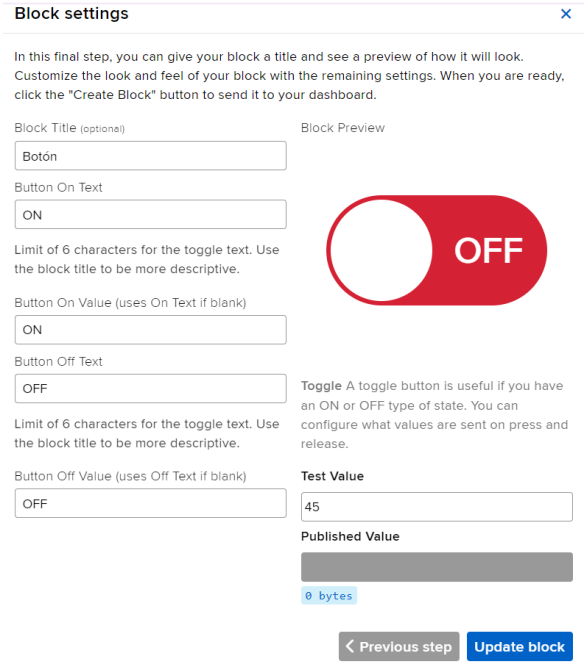


Figura 3.58 Ajuste de parámetro del botón

Posteriormente, en la página principal del *dashboard*, se visualiza el bloque del botón, el cual se encuentra ubicado en una posición estratégica para facilitar su interacción con los usuarios. Este botón como se ilustra en la Figura 3.59, al ser cliqueado, desencadena una serie de acciones ya sea prendido o apagado.

Figura 3.59 *Block* del botón

En la configuración de los siguientes bloques, se seleccionó el gráfico número 3 como se ilustra en la Figura 3.50. Este gráfico se vinculó con los *feeds "min_water_level*" y "*max_water_level*", permitiendo al usuario ajustar manualmente los niveles mínimo y máximo de agua según sus requerimientos. En la Figura 3.60 se puede ver las variables que se elegirán.

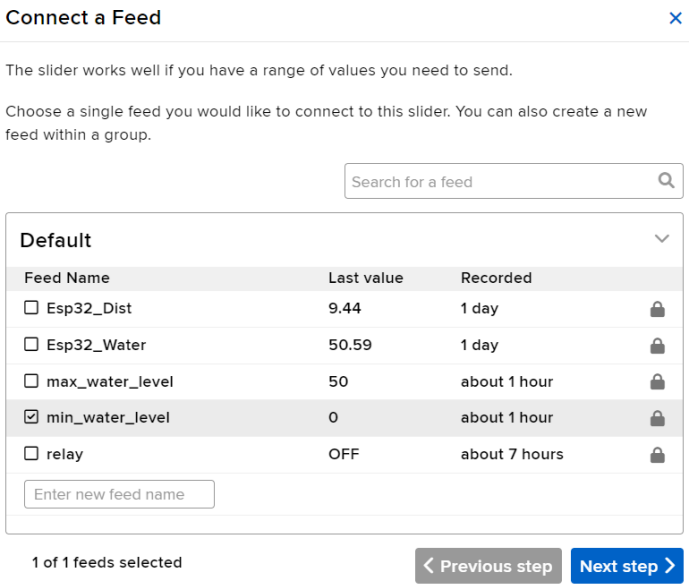


Figura 3.60 Elección de máximo y mínimo nivel de agua

Tras seleccionar el bloque, se le asigna un nombre descriptivo: "Nivel mínimo de agua y Nivel máximo de agua". Luego, se configuran sus parámetros según la Figura 3.61. Para el nivel mínimo de agua, se establece un rango de valores entre 0% y 50%, mientras que, para el nivel máximo, el rango es de 50% a 100%. Esta división en dos segmentos facilita la visualización y comprensión de la información para los usuarios, permitiendo identificar rápidamente cuándo el nivel de agua se encuentra en una fase crítica (bajo) o en una fase extrema (alto).

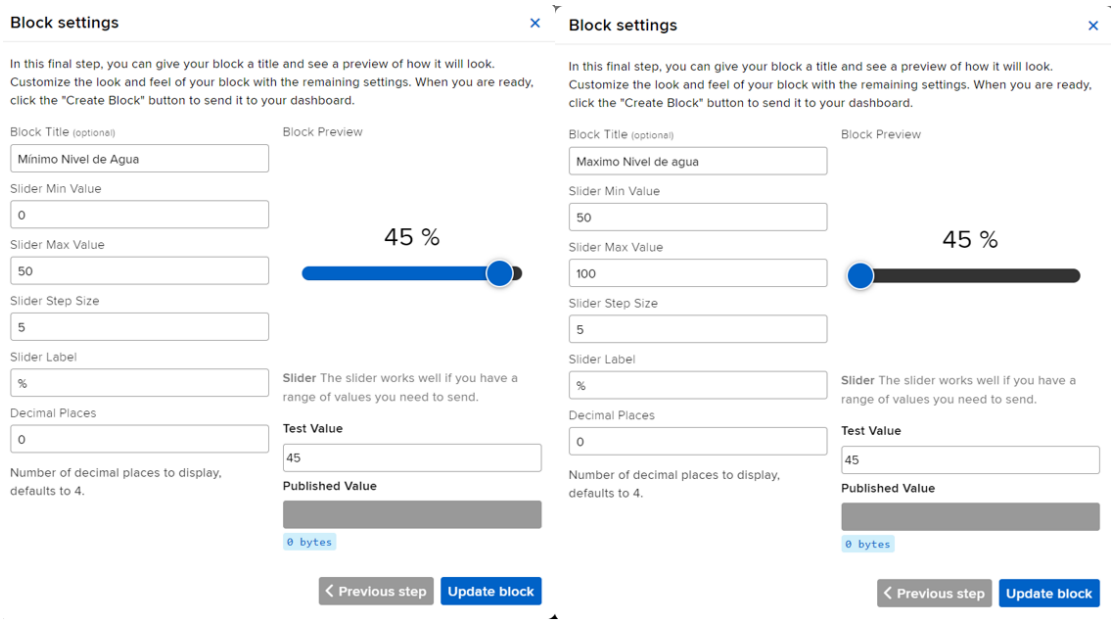


Figura 3.61 Ajuste de parámetros mínimo y máximo nivel de agua

La verificación de los bloques de "Nivel mínimo de agua" y "Nivel máximo de agua" se realiza finalmente en el *dashboard* de *Adafruit*. Para una mayor comodidad en el manejo

de datos, los valores seleccionables manualmente se presentan en intervalos de 5 unidades tal como se observa en la Figura 3.62.

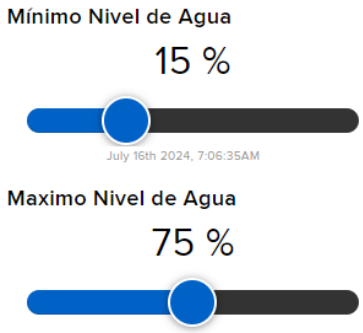


Figura 3.62 *Block* de mínimo y máximo nivel de agua

En el *dashboard* principal ahora integra los bloques de distancia, nivel de agua, botón, mínimo nivel de agua y máximo nivel de agua, permitiendo a los usuarios monitorizar diversos aspectos del sistema desde la interfaz web. Se ha agregado un botón que activa y desactiva la mini bomba de agua. Este control es fundamental para gestionar el funcionamiento del sistema de forma remota y asegurar que el agua se distribuye adecuadamente según los requisitos del usuario.

Además, la interfaz gráfica ofrece la posibilidad de configurar varios parámetros. Entre estos, destaca la opción de establecer un umbral para el llenado de la cisterna, lo cual proporciona un mayor control y prevención de desbordamientos. Esta característica permite ajustar los niveles de agua deseados de manera precisa y automática, optimizando así el uso del sistema.

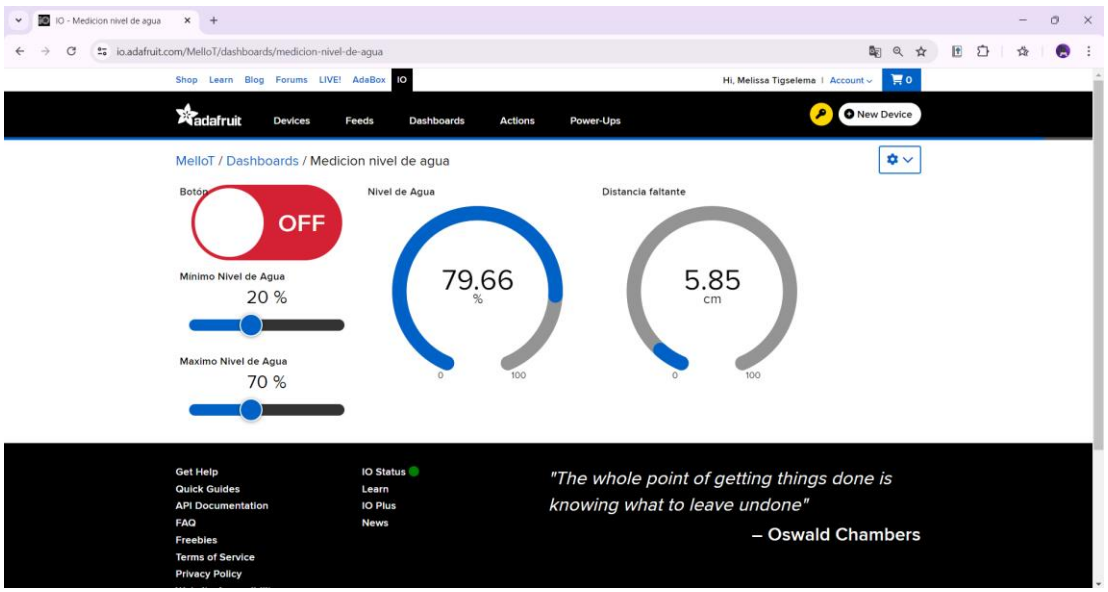


Figura 3.63 *Dashboard* medición de nivel de agua

El *block* del nivel de agua está diseñado para mostrar la información en forma de porcentaje. Esta representación visual es útil para que el usuario pueda ver de un vistazo cuánto se ha llenado la cisterna, facilitando el monitoreo continuo sin la necesidad de cálculos adicionales.

Además, el *block* de distancia proporciona información sobre la separación entre el sensor y la superficie del agua. Esta información es crucial para evaluar el estado actual del tanque. La integración de estos *blocks* en la interfaz web asegura una supervisión detallada y un control eficiente del sistema de gestión de agua como se ilustra en la Figura 3.63.

Creación de alertas

Para la creación de alertas hay que dirigirse a la opción "*Actions"* de *Adafruit IO* y seleccionar "*New Action"* en esta aparecerá varios apartados. Se debe dar "*click"* en la pestaña "*Reactive"* como se ve en la Figura 3.64.

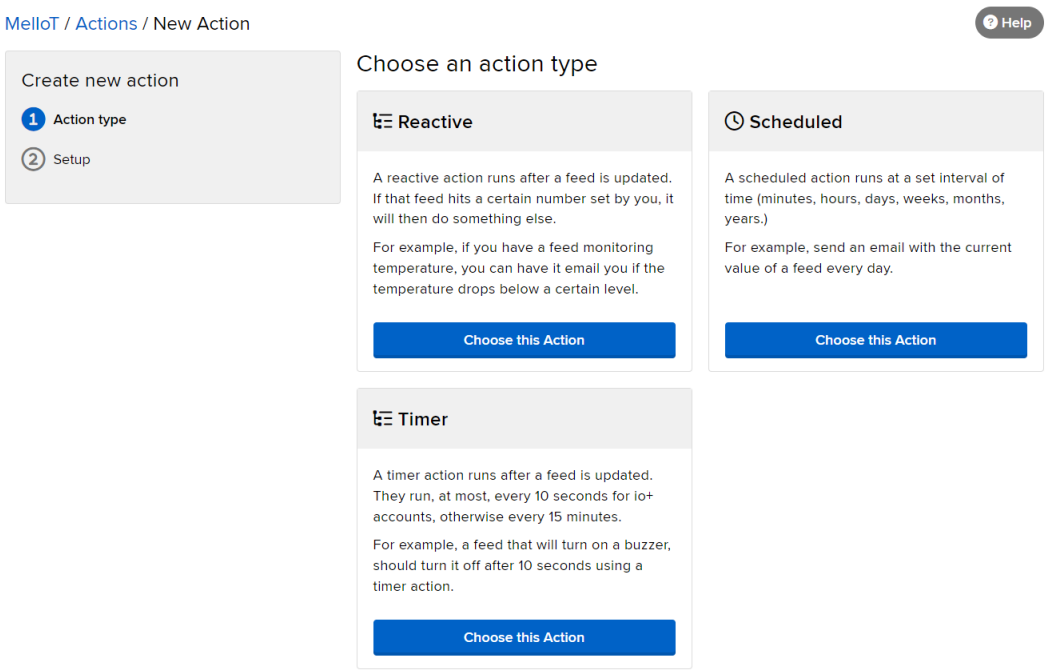


Figura 3.64 Selección de acción

Luego pedirá que se llene estos campos como se ilustra en la Figura 3.65 para crear la alerta.

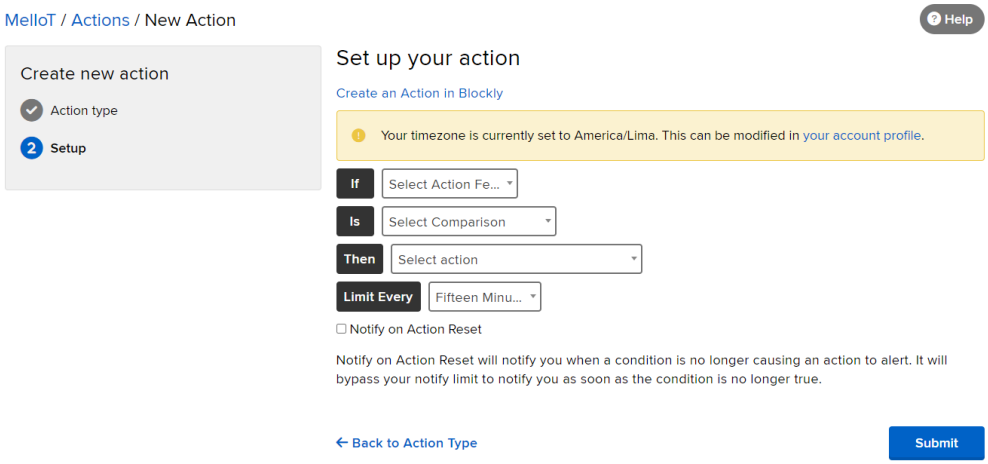


Figura 3.65 Configuración de alerta

Ya llenados los campos solicitados se procede a dar "*click*" en "*Submit*" para guardar los datos tal y como se observa en la Figura 3.66.

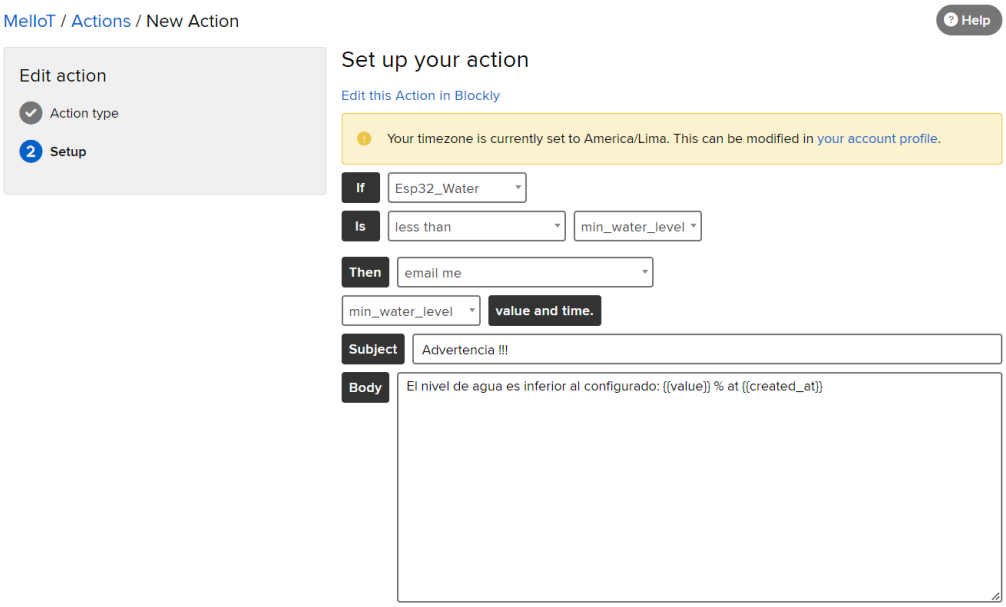


Figura 3.66 Llenado de campos

Las alertas creadas se pueden visualizar en la pestaña "*Actions*" tal y como se observa en la Figura 3.67.

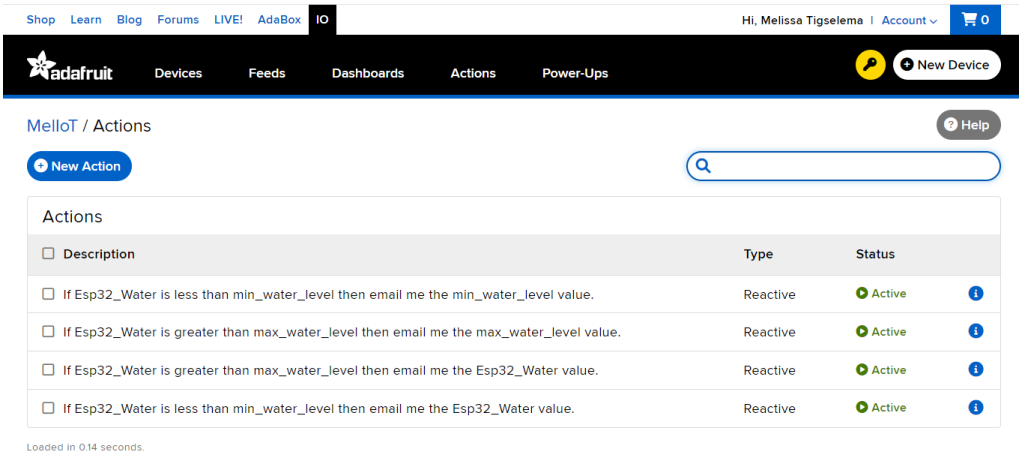


Figura 3.67 Alertas configuradas

3.4 Implementación del prototipo de control de nivel de agua

Desarrollo del código

Para el desarrollo del código de ambos *SoCs Wi-Fi LoRa ESP32 Heltec (V2)*, se utilizó el programa *Arduino IDE* el cual simplifica la implementación gracias a su compatibilidad con una amplia variedad de *hardware*, permitiendo centrarse en la lógica del código. El proceso comenzó con la selección de la placa adecuada dentro del *IDE*, en este caso, la placa *Wi-Fi LoRa ESP32 Heltec (V2)*.

A continuación, se seleccionó el puerto al que estaría conectado el *SoC*, asegurando una correcta comunicación entre el *IDE* y el *hardware*. Una vez establecidos estos parámetros, se procedió a compilar el código. Tras la compilación exitosa, se cargó el código en el *SoC.*

Integración y conexiones electrónicas

Para facilitar la comprensión de las conexiones, se ha elaborado un esquema general de los componentes electrónicos utilizados, que incluye el sensor HC-SR04 y el relé. En el diseño del proyecto, se prestó especial atención a la ubicación de los componentes, asegurando que encajen de la mejor manera posible para optimizar el funcionamiento y la eficiencia del montaje

Como se ve observar en la Figura 3.68, el sensor HC-SR04 utiliza un voltaje de alimentación de 5 (V), mientras que el relé opera con un voltaje de 3 (V). La utilización de espadines hembra no solo facilita la conexión de estos componentes, sino que también permite reemplazar el sensor fácilmente en caso de daño o fallo. Esto es crucial

para el mantenimiento del sistema, ya que minimiza el tiempo de inactividad y facilita la reparación.

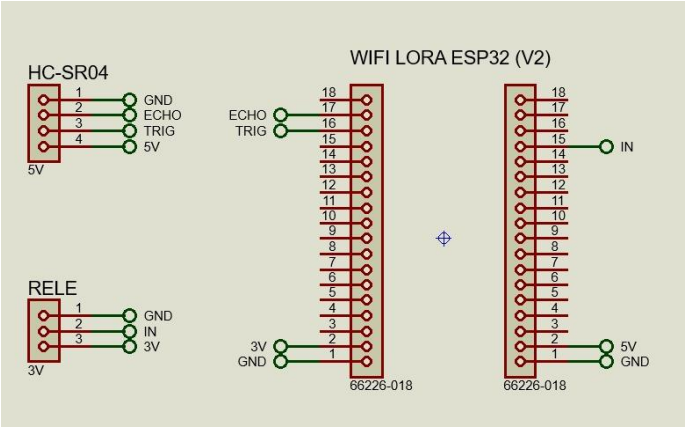


Figura 3.68 Esquema de conexión

Además, la fuente de alimentación ha sido diseñada para ser conectada y desconectada fácilmente, permitiendo configuraciones futuras según las necesidades del proyecto. Esto proporciona flexibilidad para adaptar el sistema a nuevas funcionalidades o requerimientos, sin necesidad de realizar modificaciones complejas en el *hardware*. De esta manera, se asegura una mayor versatilidad y adaptabilidad del proyecto a largo plazo, permitiendo actualizaciones y mejoras continuas sin complicaciones.

La elección de los pines del *Wi-Fi LoRa ESP32 Heltec (V2)* se llevó a cabo mediante una cuidadosa verificación de cuáles cumplen con los requisitos técnicos necesarios para el correcto funcionamiento del sistema. Esta verificación asegura que los pines seleccionados pueden manejar las corrientes y voltajes adecuados, así como la comunicación eficiente con otros componentes electrónicos.

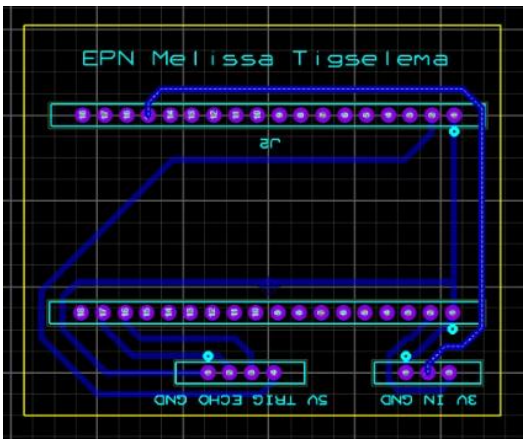


Figura 3.69 Diseño de *PCB*

La *PCB* (Placa de Circuito Impreso) se diseñó para tener un tamaño diminuto, optimizando el uso del espacio y facilitando su integración en aplicaciones donde es

limitado. Este diseño compacto no solo mejora la portabilidad y el manejo del dispositivo, sino que también permite una mayor flexibilidad en el montaje y la colocación de la *PCB* dentro de diferentes entornos como se ve en la Figura 3.69.

La distribución de pines se ha configurado de la siguiente manera: los pines 12 y 13 del *Wi-Fi LoRa ESP32 Heltec (V2)* están conectados a los pines *trigger* y *echo* del sensor HC-SR04, respectivamente. El pin *trigger* envía un pulso para iniciar la medición, y el pin *echo* recibe el pulso reflejado para calcular la distancia. Para la alimentación del sensor, se ha utilizado el pin de 5 (V) y la conexión a tierra (*GND*). De manera similar, el relé que actúa como interruptor está conectado al pin 2 del *Wi-Fi LoRa ESP32 Heltec (V2)*, mientras que la alimentación del relé se realiza mediante los pines 3 (V3) y *GND* como se visualiza en Tabla 3.7.

Tabla 3.7 Conexión de pines *Wi-Fi LoRa ESP32 Heltec (V2),* HC-SR04 *y* relé

Elemento Electrónico	**Pin**	**Identificador**	**Conexión**
Wi-Fi LoRa ESP32 Heltec (V2)	5V	5V	(V_{CC}) conectado al pin del HC-SR04
	GND	*GND*	Conectado al pin de puesta a tierra de HC-SR04
	12	GPI012	Conectado al pin *Trigger* del sensor HC-SR04
	13	GPI013	Conectado al pin *Echo* del sensor HC-SR04
	2	GPI02	Conectado al pin *IN* del relé
	3V3	3V3	Conexión al Vin del relé
	GND	*GND*	Conexión de puesta a tierra del relé
HC-SR04	5V	5V	Voltaje de entrada (+)
	GND	*GND*	Conectado al *GND* del *SoC*
	Trigger	GPI012	Conectado al pin 12 del *SoC*
	Echo	GPI013	Conectado al pin 13 del *SoC*
Relé	3V3	3V3	Conectado al Voltaje de 3V3 del *SoC*
	GND	*GND*	Conectado al *GND* del *SoC*
	IN	IN	Conectado al pin 2 del *SoC*

En la Figura 3.70 se puede observar la disposición de los espadines para la conexión de cada componente. Los dos espadines más largos están configurados para el *Wi-Fi LoRa ESP32 Heltec (V2)*. Los espadines en la parte inferior izquierda de la figura están destinados a la conexión del sensor HC-SR04 y del relé en el lado derecho, asegurando el montaje de todos los componentes.

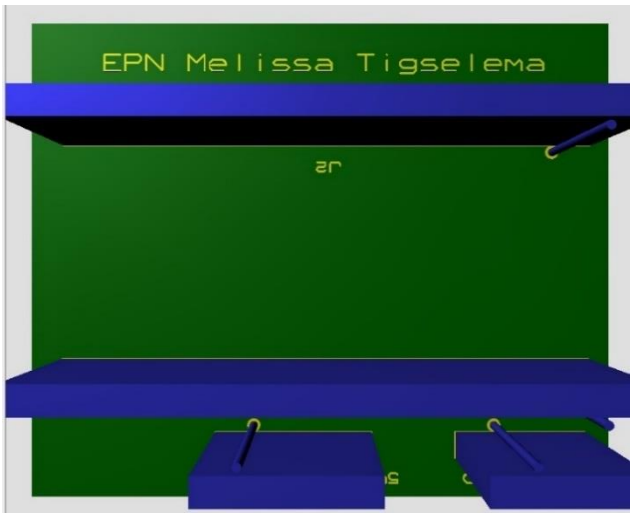


Figura 3.70 *PCB* en 3D

Para elaborar la baquelita, el proceso comienza con el diseño del circuito en Proteus. Una vez completado el diseño, se procede a preparar el material necesario. Se corta un trozo de baquelita con dimensiones específicas, en este caso de 4.5 por 5.5 (cm), y se realiza un lijado suave de la superficie que contiene el cobre. Posteriormente, se emplea alcohol isopropílico para limpiar meticulosamente la superficie y asegurarse de eliminar cualquier residuo que pueda interferir en el proceso de transferencia.

Después de preparar la superficie, se procede a imprimir el diseño del circuito en papel fotográfico. Es crucial imprimir el diseño en modo espejo para garantizar que, al transferirlo a la baquelita, la disposición de las pistas y componentes sea la correcta. A continuación, se posa el diseño impreso sobre la plancha de cobre, boca abajo (con la cara del circuito en contacto con la superficie de cobre). Luego se calienta una plancha para aplicar calor y que se transfiera durante unos minutos.

Después se coloca la placa en un recipiente de plástico, se sumerge en ácido férrico y se agita suavemente durante aproximadamente 25 (min) para facilitar el proceso. Posteriormente, la placa se retira con cuidado y se limpia meticulosamente para eliminar cualquier residuo de ácido. Luego, utilizando un taladro, se perforan los orificios necesarios en la placa, haciendo que los elementos se monten con precisión en sus posiciones correspondientes. Para comenzar el proceso de soldadura en una *PCB*, es crucial reunir todos los materiales necesarios como la *PCB*, espadines y pinzas. Luego,

se limpia meticulosamente la *PCB* utilizando alcohol isopropílico y un paño suave para eliminar cualquier suciedad.

Figura 3.71 Material para soldar

A continuación, se coloca los espadines en los orificios designados de la *PCB*, verificando que estén alineados correctamente. Posteriormente, se calienta el cautín en la Figura 3.71 se visualizan los elementos para proceder a soldar. Para empezar a aplicar la soldadura primero se derrite el estaño tocando la punta del cautín para formar pequeñas bolas y se suelda cada espadín cerciorándose de que la soldadura fluya correctamente y cubra la unión. Después de soldar, se debe dejar reposar para que las conexiones se enfríen completamente antes de manipular la *PCB*. Finalmente, se inspecciona cada unión soldada para asegurarse de que sean lisas, uniformes como se ilustra en la Figura 3.72.

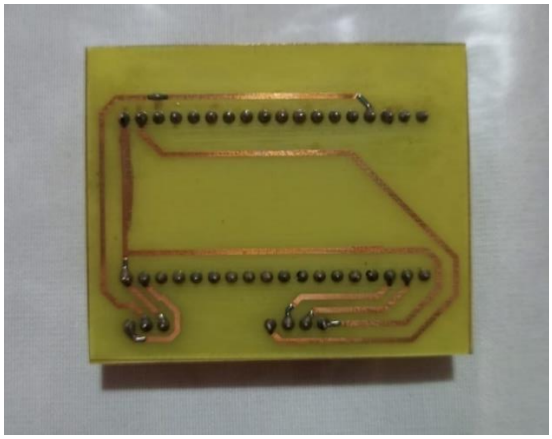


Figura 3.72 Espadines soldados

Implementación de sistema de protección

Con el objetivo de salvaguardar los componentes electrónicos sensibles, como el *PCB*, el módulo relé y el *SoC*, se diseñaron y fabricaron dos contenedores personalizados.

Cada contenedor está adaptado a las dimensiones específicas del dispositivo que alberga:

Contenedor para dispositivo esclavo: Este contenedor aloja el dispositivo en el que se encuentran el relé y el sensor y por ende el *SoC*. Sus dimensiones son de 5 (cm) de alto y 8 (cm) de ancho, proporcionando un amplio espacio para proteger los componentes internos como se muestra en la Figura 3.73.

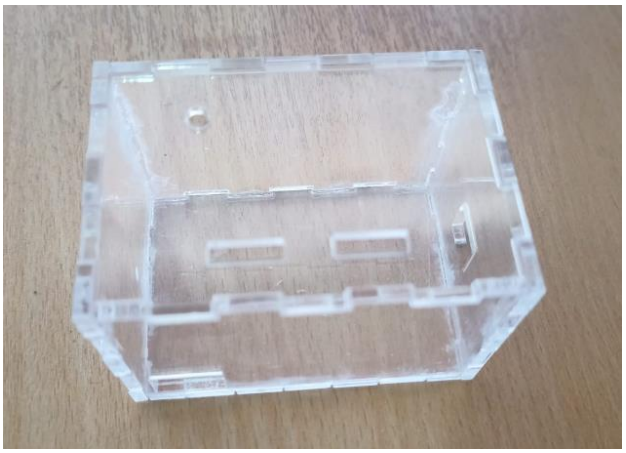


Figura 3.73 Contenedor del dispositivo esclavo

Contenedor para dispositivo maestro: Diseñado para proteger únicamente el *SoC*, este contenedor presenta dimensiones más compactas de 3 (cm) de alto y 6.5 (cm) de ancho. Su tamaño reducido permite una mejor integración en espacios limitados sin comprometer la protección del componente, así como se visualiza en la Figura 3.74.

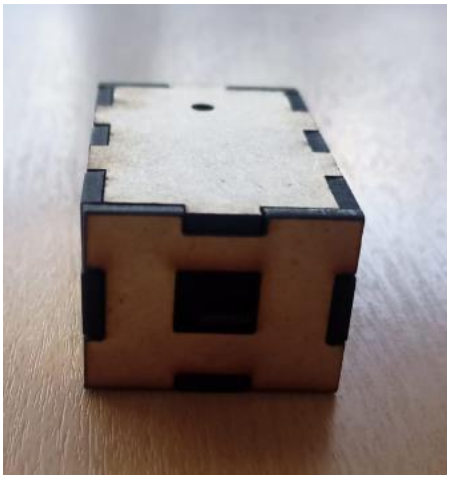


Figura 3.74 Contenedor del dispositivo maestro

La implementación de estos contenedores personalizados garantiza la integridad física de los componentes electrónicos, protegiéndolos de daños externos como golpes, vibraciones y polvo.

Implementación de la maqueta

Para replicar el funcionamiento de una cisterna, se propone utilizar una cubeta equipada con una llave de drenaje externa para vaciar el tanque una vez utilizado. En la cubeta se incorpora una manguera de llenado de igual manera se colocó un soporte para alojar el sensor de nivel de agua que monitorice el llenado progresivo en la siguiente Figura 3.75 se muestra.

Figura 3.75 Simulación de cisterna

La simulación de la cisterna también cuenta con un sistema de suministro de agua automatizado. Este sistema comprende una bomba de agua que se activa cuando el nivel de agua en la cubeta (que representa la cisterna) desciende por debajo de un umbral predefinido. La bomba extrae agua de una fuente externa y la envía a la cubeta a través de la manguera de llenado como se presenta en la Figura 3.76.

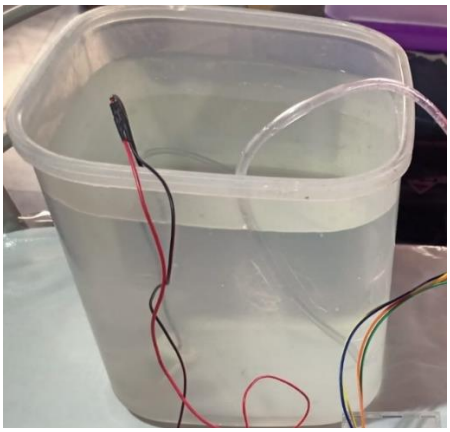


Figura 3.76 Suministro de agua

Finalizando la implementación, se ubicaron los contenedores que albergan tanto al dispositivo esclavo con su fuente de alimentación para el *SoC* y la mini bomba. Y al dispositivo maestro con su respectiva fuente de alimentación tal y como se muestra en la Figura 3.77.

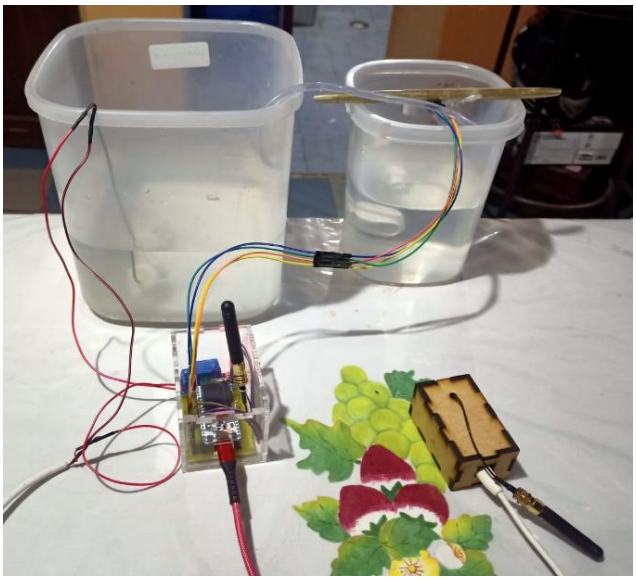


Figura 3.77 Maqueta para medir el nivel de agua

Para completar la configuración del sistema, se procede al inicio de sesión en la plataforma *Adafruit IO* utilizando las credenciales correspondientes. Una vez autenticado, se accede al *dashboard* previamente configurado con los bloques necesarios para el control y monitoreo de la simulación de la cisterna como se muestra en la Figura 3.78 *Dashboard* completo

.

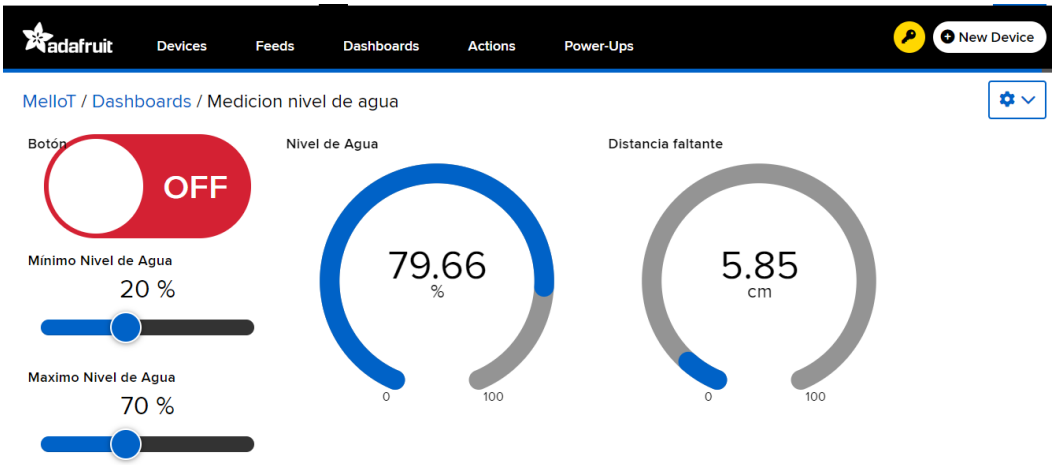


Figura 3.78 *Dashboard* completo

3.5 Implementación de las pruebas de funcionamiento

Luego de definir los requisitos, elegir el *hardware* y *software*, diseñar y construir el prototipo, se pasa a probar el sistema completo. En esta etapa, se verifica que el modelo simulado de cisterna funcione como se espera, verificando que cumple con los objetivos planteados y que funcione de manera correcta.

Inicio de la prueba

Como se mencionó anteriormente el usuario debe estar registrado en la interfaz web en donde se encuentra el *dashboard* para el monitoreo. De igual manera debe tener energizado el prototipo. Ya con esto en orden el *SoC* maestro establece conexión con internet y *Adafruit IO*. Los datos provenientes del sensor HC-SR04 son transmitidos vía *LoRa* desde el *SoC* esclavo al *SoC* maestro y se reflejan en esta plataforma web.

En la Figura 3.79 presentada a continuación se puede ver que los *blocks* de nivel de agua y distancia están en 16 (cm) y el nivel de agua se mantiene en 0 (cm). Y el recipiente que actúa como cisterna está completamente vacío.

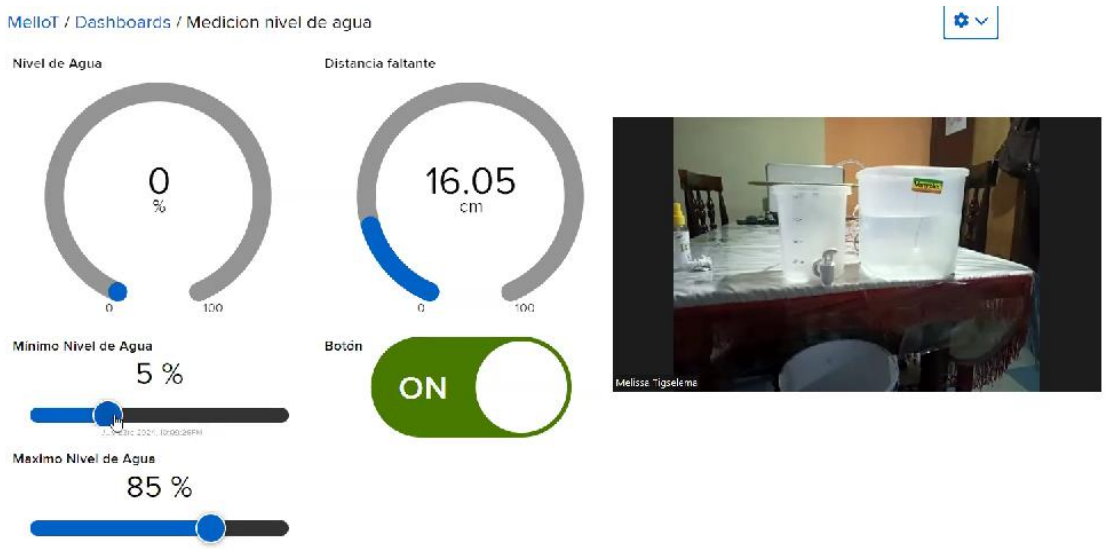


Figura 3.79 Inicialización del prototipo

Prueba de control automático en la interfaz web

Para la prueba de funcionamiento del control automático, como primer paso se deben ajustar los parámetros de nivel máximo y mínimo de agua. Esto permitirá que, al tener el recipiente vacío y el nivel de agua en 0, la mini bomba se active automáticamente. Tal y como se ilustra en la Figura 3.80. Esta función es de gran utilidad en caso de que el usuario olvide llenar la cisterna, evitando así que se quede sin agua.

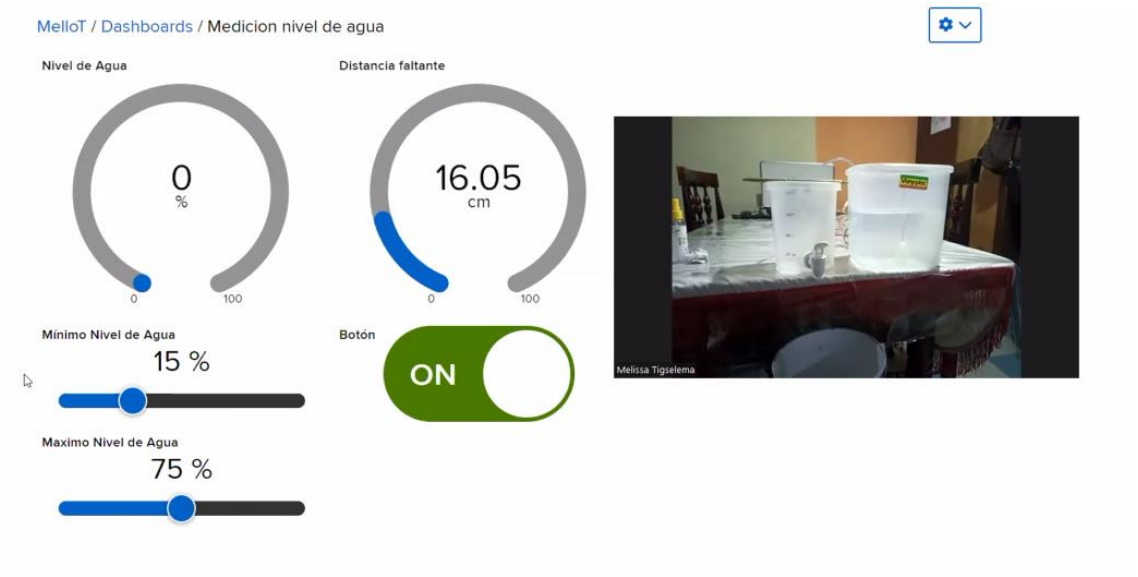


Figura 3.80 Encendido automático del sistema

Por otro lado, cuando el nivel de agua alcance el máximo establecido, la bomba se apagará automáticamente, previniendo el desbordamiento del agua. Así como se visualiza en la Figura 3.81.

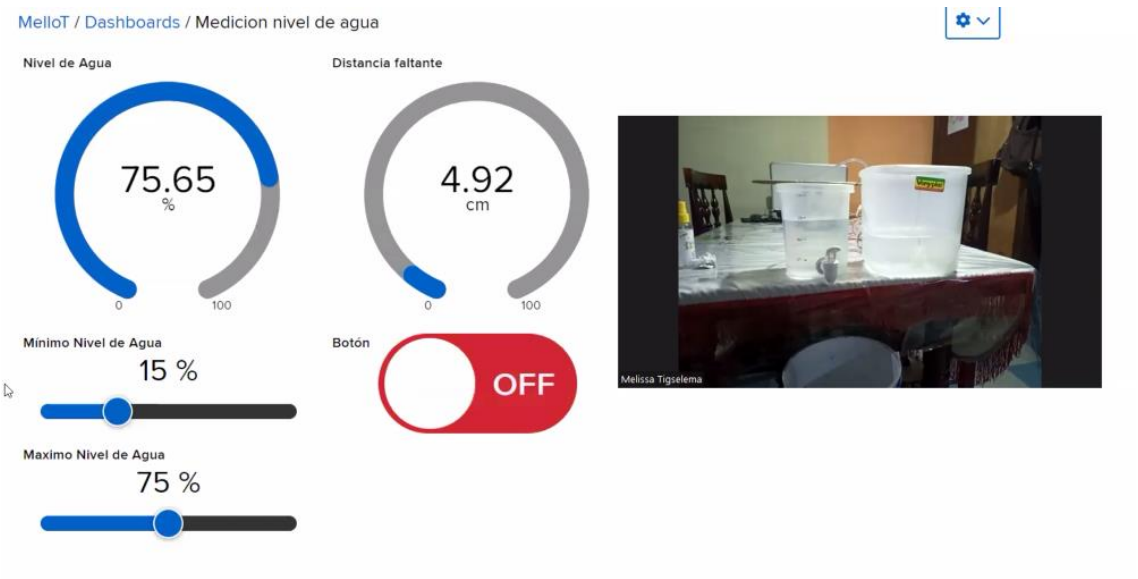


Figura 3.81 Apagado automático del sistema

Después del que se apague la mini bomba se realiza el vaciado manual de la cisterna y por consiguiente cuando llegue al mínimo nivel de establecido se activará el flujo de agua como se ve en la Figura 3.82.

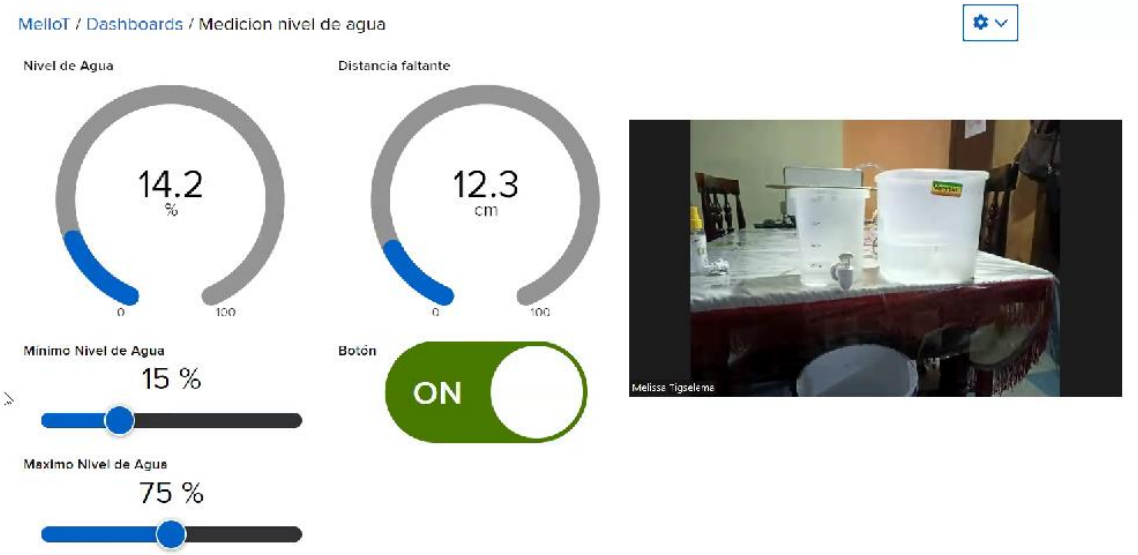


Figura 3.82 Reinicio del ciclo

Una vez completado el ciclo y llegando al nivel límite establecido se apaga la mini bomba tal y como se visualiza en la Figura 3.83.

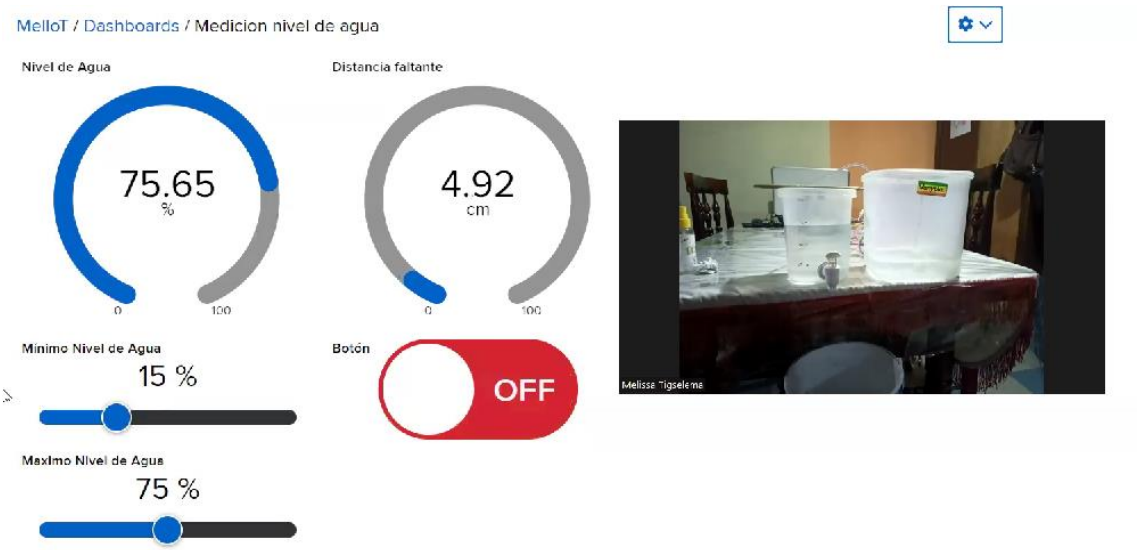


Figura 3.83 Finalización del ciclo

De igual manera, si el nivel de agua supera el límite establecido o desciende por debajo del nivel mínimo permitido, las alarmas se activarán automáticamente como se observa en la Figura 3.84.

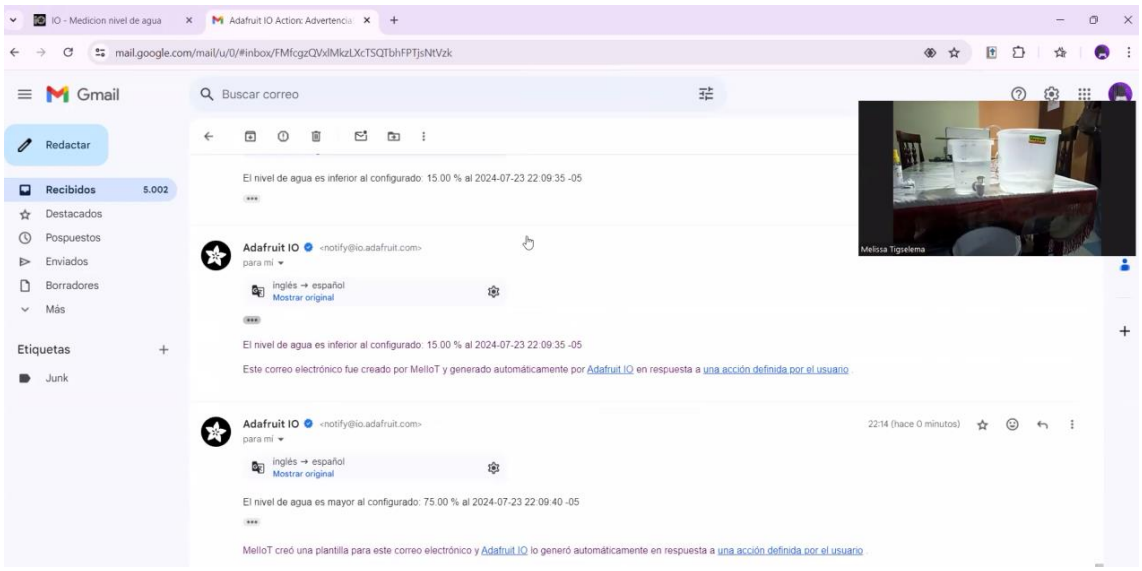


Figura 3.84 Alerta de sobrepasar el nivel máximo y mínimo

Estas alertas se enviarán al correo personal del usuario para informarlo sobre la situación y permitirle tomar las medidas correspondientes. Además, en la Figura 3.85 se puede observar los valores del nivel de agua.

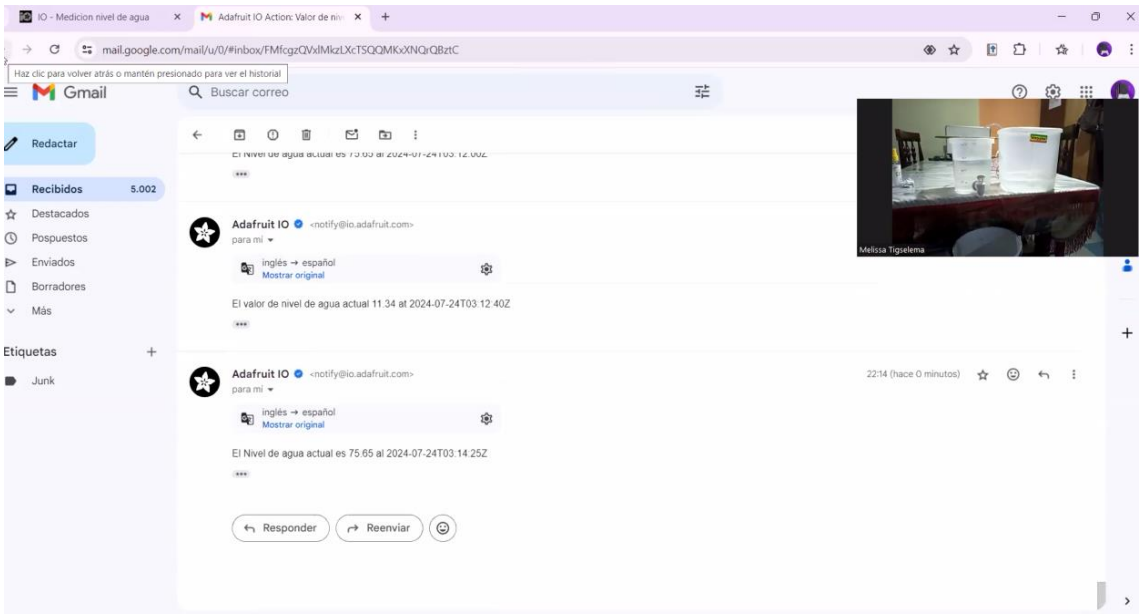


Figura 3.85 Alerta de los datos indicando nivel de agua

Resumen de las pruebas de funcionamiento

En la Tabla 3.8 se evidencian de forma breve las pruebas realizadas

Tabla 3.8 Desempeño de funcionamiento

Prueba de ejecución	Desempeño Correcto	Desempeño Incorrecto
Encendido del Sistema	X	
Sensor HC-SR04	X	
Configuración de parámetros realizados por el usuario	X	
Prendido automático de la mini bomba por mínimo nivel de agua	X	
Apagado automático de la mini bomba por máximo nivel de agua	X	
Envió de alerta por medio del correo	X	

La ejecución del prototipo ha demostrado que el sistema automático, funciona correctamente. Esto se debe a la óptima integración de todos los componentes utilizados.

Costo del prototipo

La siguiente Tabla 3.9 detalla los costos asociados al desarrollo del prototipo. Estos costos se basan en las facturas que detallan el precio individual de cada dispositivo.

Tabla 3.9 Costo del prototipo

Elemento	Cantidad	Precio Unitario	Precio Total
Wi-Fi LoRa ESP32 Heltec (V2)	2 (u)	$38.00	$76.00
Sensor HC-SR04	1 (u)	$4.00	$4.00
Relé	1 (u)	$3.60	$3.60
Mini bomba	1 (u)	$3.04	$3.04
Jumper M-M	40 (u)	$1.50	$1.50
Jumper M-H	40 (u)	$ 1.90	$1.90
Placa *PCB*	1(u)	$10.00	$10.00
Contenedor de protección	2 (u)	$15.00	$15.00

Elemento	Cantidad	Precio Unitario	Precio Total
Fuente de poder	2 (u)	$ 6.00	$12.00
Mano de obra	72 (h)	$ 10.00	$720.00
		Total	$847.04

A continuación, se presenta el enlace del video de funcionamiento: Video Funcionamiento Melissa Tigselema

4 CONCLUSIONES

- La implementación de la comunicación *LoRa* se convirtió en un requisito fundamental para la correcta interconexión de los *SoCs* dentro del sistema. Este protocolo de comunicación es ideal para dispositivos remotos y que tengan bajo consumo energético, resultando así ser la solución ideal para este proyecto ya que se logró establecer una comunicación bidireccional entre los *SoCs*, permitiendo un intercambio de información en ambas direcciones.
- Si bien existen diversos *SoCs* que integran comunicación *LoRa* y conexión *Wi-Fi*, la selección del *software* se basó en criterios de costo accesible y disponibilidad en Ecuador. Esta elección simplificó el desarrollo y la puesta en marcha del sistema, demostrando la importancia de considerar factores económicos y logísticos al momento de seleccionar tecnologías para proyectos tecnológicos.
- El diseño y desarrollo del prototipo se llevó a cabo de manera integral, considerando cada etapa del proyecto: desde la elaboración del código y la fabricación de la *PCB*. Esta metodología rigurosa permitió garantizar el funcionamiento adecuado de cada componente. La experiencia adquirida durante este proceso destaca la importancia de un enfoque detallado en el diseño de prototipos complejos, donde la integración armónica de todos los elementos es crucial para el logro de los objetivos planteados.
- La fase de implementación del proyecto se desarrolló de manera exitosa, logrando que el prototipo funcione correctamente con los materiales predefinidos. El proceso de ensamblaje de la maqueta resultó sencillo, gracias a la cuidadosa selección de materiales y a un diseño optimizado. Esta eficiencia permitió completar esta etapa del proyecto en un tiempo considerablemente menor al inicialmente previsto.
- La fase de pruebas de funcionamiento del proyecto culminó con éxito, permitiendo la configuración de los parámetros del sistema en función del nivel

de agua requerido. Además, se implementó un sistema de monitoreo mediante una interfaz web, lo que permite a los usuarios manejar y supervisar el sistema de forma remota desde cualquier lugar con acceso a internet.

5 RECOMENDACIONES

- Antes de conectar el *SoC*, es fundamental identificar los pines de entrada, salida y aquellos dedicados a la comunicación *LoRa*. Estos últimos pines no deben utilizarse para otros propósitos, ya que están reservados exclusivamente para la comunicación *LoRa*.
- Para obtener alertas en tiempo real cada segundo en *Adafruit IO*, es necesario actualizar tu cuenta a un plan pago. El plan gratuito solo permite enviar alertas con una frecuencia mínima de 15 minutos.
- Se recomienda, la calibración del sensor ya que es de suma importancia para que las mediciones sean más precisas. El sensor HC-SR04 es un dispositivo para medir distancias cortas. Sin embargo, las mediciones brutas del sensor se pueden ver afectadas por factores externos como la temperatura, la humedad y las variaciones en la producción del sensor.
- Las conexiones correctas son cruciales para el correcto funcionamiento del prototipo. Verificarlas minuciosamente antes de encender el dispositivo puede ahorrar tiempo y frustraciones.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] W. Torres, «“Nueve provincias de Ecuador con bajo acceso a agua potable”,» 03 Julio 2021. [En línea]. Available: https://www.primicias.ec/noticias/economia/provincias-ecuador-acceso-agua-potable/#:~:text=De%20acuerdo%20con%20la%20agencia,entre%20Napo%2C%20Ca%C3%B1ar%20y%20Bol%C3%ADvar. [Último acceso: 20 Julio 2024].

[2] R. Carrillo, «Qué es LoRa, cómo funciona y características principales - Venco Electrónica,» Venco Electrónica, 8 Agosto 2022. [En línea]. Available: https://www.vencoel.com/que-es-lora-como-funciona-y-caracteristicas-principales/. [Último acceso: 5 Mayo 2024].

[3] «“¿Qué es IoT (Internet Of Things)?,» Deloitte Spain, 8 Enero 2019. [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html . [Último acceso: 6 Mayo 2024].

[4] «“Modulo Wifi LoRa 32 Heltec | MCI Electronics.cl,”,» MCI Electronics, 11 Mayo 2024. [En línea]. Available: https://mcielectronics.cl/shop/product/modulo-wifi-lora-32-heltec-27575/. [Último acceso: 11 Mayo 2024].

[5] «“Mini Bomba de Agua Sumergible 120L/H - AV Electronics,”,» AV Electronics, 6 Abril 2024. [En línea]. Available: https://avelectronics.cc/producto/mini-bomba-de-agua-sumergible-120l-h/. [Último acceso: 2024 Mayo 2024].

[6] yuridia, « “Sensores,”,» SDI, 25 Junio 2021. [En línea]. Available: https://sdindustrial.com.mx/blog/sensores/ . [Último acceso: 6 Mayo 2024].

[7] A. Brunete, « “2.2 Actuadores eléctricos | Introducción a la Automatización Industrial,”,» Bookdown.org, 2014. [En línea]. Available: https://bookdown.org/alberto_brunete/intro_automatica/actuadoreselectricos.html. [Último acceso: 23 Mayo 2024].

[8] jecrespom, « “IDE Arduino,”,» Aprendiendo Arduino, 11 Diciembre 2016. [En línea]. Available: https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/. [Último acceso: 24 Mayo 2024].

[9] «“Aprende a utilizar la plataforma Adafruit IO para tus dispositivos IoT (parte 1) | MK Electronica,”,» MK Electronica, 16 Junio 2022. [En línea]. Available: https://mkelectronica.com/aprende-a-utilizar-la-plataforma-adafruit-io-para-tus-dispositivos-iot-parte-1/. [Último acceso: 23 Mayo 2024].

[10] «“Tecnología LoRA y LoRAWAN - Catsensors,”,» Catsensors.com, 2024. [En línea]. Available: https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan#:~:text=LoRa%20es%20una%20tecnolog%C3%ADa%20inal%C3%A1mbrica,fabricante%20de%20chips%20de%20radio. [Último acceso: 14 Mayo 2024].

[11] T. Tecnologia, «“ESP32 LoRa V2 Heltec 863 a 928 MHz com Display OLED, Bluetooth e WiFi,”,» Smartkits.com.br, 2022. [En línea]. Available: https://www.smartkits.com.br/esp32-lora-v2-heltec-863-a-928-mhz-com-display-oled-bluetooth-e-wifi. [Último acceso: 15 Junio 2024].

[12] J3, «“Bluetooth Onboard LED blink — WiFi LoRa ESP32 (Heltec),”,» Medium, 26 Enero 2024. [En línea]. Available: https://medium.com/jungletronics/bluetooth-onboard-led-blink-wifi-lora-esp32-heltec-4fc1b6e41c45. [Último acceso: 15 Junio 2024].

[13] «“Wireless Stick Lite(V3),”,» Heltec Automation, 2023. [En línea]. Available: https://heltec.org/project/wireless-stick-lite-v2/. [Último acceso: 16 Junio 2024].

[14] «“Wireless Stick Lite V3, I2C,”,» Heltec Automation Technical Community, 04 Agosto 2023. [En línea]. Available: http://community.heltec.cn/t/wireless-stick-lite-v3-i2c/13489/3. [Último acceso: 16 Junio 2024].

[15] J. Antonio, «“Esp32 características y pines - Pasión electrónica,”,» PASIÓN ELECTRÓNICA, 12 Febrero 2022. [En línea]. Available: https://pasionelectronica.com/esp32-caracteristicas-y-pines/ . [Último acceso: 16 Junio 2024].

[16] «“ESP32-WROOM-32D,”,» Sigma Electrónica, 14 Junio 2024. [En línea]. Available: https://www.sigmaelectronica.net/producto/esp32-wroom-32d/#:~:text=Especificaciones%3A,Tip%C3%ADco%3A%2080%20mA. [Último acceso: 15 Junio 2024].

[17] «“Sonido ¦ Ultrasonido : Módulo sensor de distancias HC-SR04,”,» Eneka.com.uy, 2014. [En línea]. Available: https://www.eneka.com.uy/robotica/sensores/sonido/m%C3%B3dulo-sensor-de-distancias-hc-sr04-detail.html. [Último acceso: 17 Junio 2024].

[18] «“JSN-SR04T,”,» Sigma Electrónica, 23 Abril 2024. [En línea]. Available: https://www.sigmaelectronica.net/producto/jsn-sr04t/. [Último acceso: 17 Junio 2024].

[19] «“Módulo Relé 1 Canal - AV Electronics,”,» AV Electronics, 12 Junio 2024. [En línea]. Available: https://avelectronics.cc/producto/modulo-rele-1-canal/. [Último acceso: 17 Junio 2024].

[20] «“Modulo de relé de 1 canal con regulador a 12V – Novatronic,”,» Novatronicec.com, 2020. [En línea]. Available: https://novatronicec.com/index.php/product/modulo-de-rele-de-1-canal-con-regulador-a-12v/ . [Último acceso: 17 Junio 2024].

[21] «“MINI BOMBA DE AGUA SUMERGIBLE 5V – Grupo Electrostore,”,» Grupoelectrostore.com, 2023. [En línea]. Available: https://grupoelectrostore.com/shop/motores/bombas-para-agua/mini-bomba-de-agua-sumergible-5v/. [Último acceso: 17 Junio 2024].

[22] FabioLeon, «“Baterías LiPo, características y cuidados! - DynamoElectronics,”,» DynamoElectronics, Octubre 2019. [En línea]. Available: https://www.dynamoelectronics.com/baterias-lipo-caracteristicas-y-cuidados/. [Último acceso: 17 Junio 2024].

[23] «“ADAPTADOR DE VOLTAJE 5V 1A CON CONECTOR MICRO USB TIPO B,”,» VISTRONICA S.A.S, 2016. [En línea]. Available: https://www.vistronica.com/fuente-de-voltaje/adaptadores/adaptador-de-voltaje-5v-1a-con-conector-micro-usb-tipo-b-detail.html. [Último acceso: 17 Junio 2024].

[24] «“5VPOW Fuente de Poder MikroTik 5V 1A,”,» Antenas y Dispositivos de Red - Sincables.EC, 27 Mayo 2024. [En línea]. Available: https://www.sincables.com.ec/product/5vpow-fuente-de-poder-mikrotik-5v-1a/. [Último acceso: 17 Junio 2024].

[25] «“Batería LiPo (de Polímero de litio) - 1200mAh - Ultra-lab,”,» Ultra-lab, 4 Marzo 2024. [En línea]. Available: https://ultra-lab.net/producto/bateria-lipo-de-polimero-de-litio-1200mah/. [Último acceso: 17 Junio 2024].

[26] C. Yañez, «“Qué es Arduino IoT Cloud | CEAC,”,» CEAC, 3 Noviembre 2020. [En línea]. Available: https://www.ceac.es/blog/que-es-arduino-iot-cloud. [Último acceso: 17 Junio 2024].

[27] A. Forum, «“Random state change and update from Arduino IoT Cloud,”,» Arduino Forum, 16 Noviembre 2021. [En línea]. Available: https://forum.arduino.cc/t/random-state-change-and-update-from-arduino-iot-cloud/925623. [Último acceso: 18 Junio 2024].

7 ANEXOS

ANEXO I. Certificado de originalidad

ANEXO II. Enlaces

ANEXO III. Conjunto de datos extensos

La numeración de los **Anexos** debe realizarse con números en formato romano minúscula.

ANEXO I: Certificado de Originalidad

F_AA_236

CERTIFICADO DE ORIGINALIDAD
TRABAJO DE INTEGRACIÓN CURRICULAR

Quito, D.M. 31 de julio de 2024

De mi consideración:

Yo, **LEANDRO ANTONIO PAZMIÑO ORTIZ**, en calidad de Director del Trabajo de Integración Curricular titulado **IMPLEMENTACIÓN DE UN PROTOTIPO CON SOC PARA CONTROL DE NIVEL DE AGUA EN UNA CISTERNA MEDIANTE COMUNICACIÓN LORA,** componente **IMPLEMENTACIÓN DE UN PROTOTIPO CON SOC PARA CONTROL DE NIVEL DE AGUA EN UNA CISTERNA MEDIANTE COMUNICACIÓN LORA Y A TRAVÉS DE UNA INTERFAZ WEB** elaborado por el estudiante **MELISSA ANAHÍ TIGSELEMA PACHECO** de la carrera en **TECNÓLOGÍA 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.

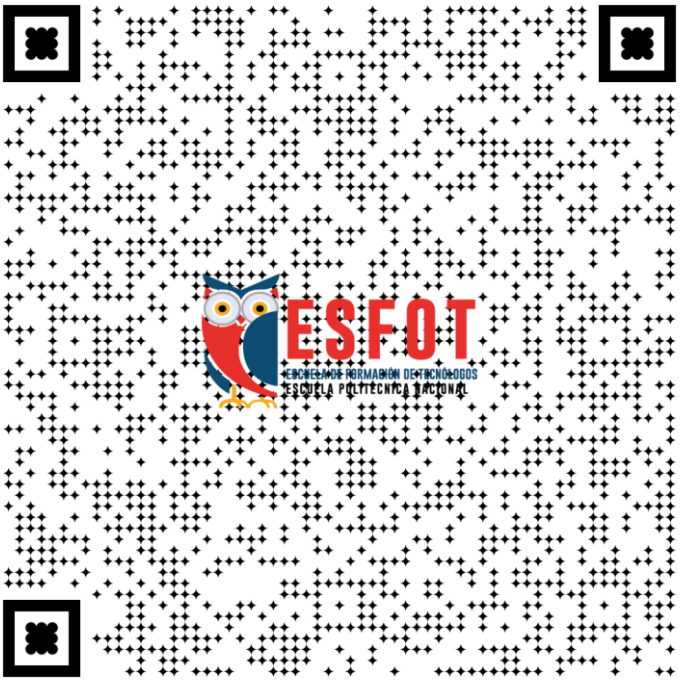
NOTA: Se adjunta el informe generado por la herramienta Turnitin.

Atentamente,

LEANDRO ANTONIO PAZMIÑO ORTIZ
Docente
Escuela de Formación de Tecnólogos

ANEXO II: Enlaces

En el siguiente QR se encuentran el video de funcionamiento del prototipo. Video Funcionamiento Melissa Tigselema



Anexo II.I Código QR de prueba de funcionamiento del prototipo

ANEXO III: Códigos Fuente

Código del *SoC* maestro

```
#include "Arduino.h"

#include "LoRaWan_APP.h"

#include <WiFi.h>

#include <PubSubClient.h>

#include <Adafruit_Sensor.h>

#include <Ticker.h>

//************************************************************************

// Credenciales Red WiFi

#define WIFI_SSID "Alfanet_WTigselema"

#define WIFI_PASSWORD "AMD301812#73"

// Credenciales Adafruit

#define ADAFRUIT_USER "MelloT"

#define ADAFRUIT_KEY "aio_FstB83A1TplfxDuGGDt8Em3rTsSc"

// Servidor

#define ADAFRUIT_SERVER "io.adafruit.com"

#define ADAFRUIT_PORT 1883

char ADAFRUIT_ID[30];

// Publicar

#define ADAFRUIT_FEED_Esp32_Dist        ADAFRUIT_USER "/feeds/Esp32_Dist"

#define      ADAFRUIT_FEED_Esp32_Water                    ADAFRUIT_USER
"/feeds/Esp32_Water"

#define ADAFRUIT_FEED_RELAY             ADAFRUIT_USER "/feeds/relay"

#define      ADAFRUIT_FEED_MAX_WATER_LEVEL                ADAFRUIT_USER
"/feeds/max_water_level"
```

```
#define       ADAFRUIT_FEED_MIN_WATER_LEVEL                    ADAFRUIT_USER
"/feeds/min_water_level"

// Suscripción

#define ADAFRUIT_DATA_IN   ADAFRUIT_USER "/feeds/data_in"

//************************************************************************************

#define RF_FREQUENCY                                916000000 // Hz

#define TX_OUTPUT_POWER                             14        // dBm

#define LORA_BANDWIDTH                              0         // [0: 125 kHz,

                                                    //  1: 250 kHz,

                                                    //  2: 500 kHz,

                                                    //  3: Reserved]

#define LORA_SPREADING_FACTOR                       7         // [SF7..SF12]

#define LORA_CODINGRATE                             1         // [1: 4/5,

                                                    //  2: 4/6,

                                                    //  3: 4/7,

                                                    //  4: 4/8]

#define LORA_PREAMBLE_LENGTH                        8         // Same for Tx and Rx

#define LORA_SYMBOL_TIMEOUT                         0         // Symbols

#define LORA_FIX_LENGTH_PAYLOAD_ON                  false

#define LORA_IQ_INVERSION_ON                        false


#define RX_TIMEOUT_VALUE                            1000

#define BUFFER_SIZE                                 30 // Defina el tamaño de la carga

//*************************************************************

//Declaración de Variables

char txpacket[BUFFER_SIZE];

char rxpacket[BUFFER_SIZE];
```

```
static RadioEvents_t RadioEvents;
int16_t rssi, rxSize;

String StrDataLoraIn, StrIndx;
float Val_Dist, Val_Water;
float Esp32_Dist, Esp32_Water;
int pastMaxWaterLevel, pastMinWaterLevel;
int relay, maxWaterLevel, minWaterLevel;
bool relayState = false;
bool lora_idle = true;

Ticker tiempo_1; // Declaración global de la variable Ticker
WiFiClient espClient;
PubSubClient client(espClient);
//*********************************************************
//Declaración de Funciones
String getValue(String data, char separator, int index);
void funcion_1();
void sendRelayState();
void sendMaxWaterLevel(String maxWaterLevel);
void sendMinWaterLevel(String minWaterLevel);
void setup_wifi();
void callback(char* topic, byte* payload, unsigned int length);
void reconnect();
void mqtt_publish(String feed, float val);
void mqtt_publish(String feed, String val);
```

```
void get_MQTT_ID();
//*****************************************************
void setup()
{
   Serial.begin(115200);
   //****
   get_MQTT_ID();
   setup_wifi();
   client.setServer(ADAFRUIT_SERVER, ADAFRUIT_PORT);
   client.setCallback(callback);
   //****
   Mcu.begin();


   rssi = 0;


   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(5, funcion_1); // Correcta llamada a attach
}
//*******************************************************
void loop()
{
    if (!client.connected())
    {
        reconnect();
    }
    client.loop();

    if (lora_idle)
    {
        lora_idle = false;
        Serial.println("into RX mode");
        Radio.Rx(0);
    }
    Radio.IrqProcess();
}
//*******************************************************
```

// Esta función es un temporizador que se activa cada 5 segundos. Publica los valores de Esp32_Dist y Esp32_Water en los feeds correspondientes de Adafruit utilizando MQTT.

```
void funcion_1()
{
    mqtt_publish(ADAFRUIT_FEED_Esp32_Dist, Esp32_Dist);
    mqtt_publish(ADAFRUIT_FEED_Esp32_Water, Esp32_Water);
    mqtt_publish(ADAFRUIT_FEED_RELAY, relay==1 ?"ON":"OFF");
    if(pastMaxWaterLevel != maxWaterLevel){
    mqtt_publish(ADAFRUIT_FEED_MAX_WATER_LEVEL, maxWaterLevel);
    }
    if(pastMinWaterLevel != minWaterLevel){
    mqtt_publish(ADAFRUIT_FEED_MIN_WATER_LEVEL, minWaterLevel);
    }
    pastMaxWaterLevel = maxWaterLevel;
    pastMinWaterLevel = minWaterLevel;
    sendRelayState();
}

void OnTxDone(void) {
    Serial.println("TX done.....");
    lora_idle = true;
    Radio.Rx(0);  // Habilitar el modo receptor después de la transmisión
}

void OnTxTimeout(void) {
    Radio.Sleep();
```

```
    Serial.println("TX Timeout.....");
    lora_idle = true;
    Radio.Rx(0);  // Activar el modo receptor después del tiempo de espera.
}
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)
{
    rssi = rssi;
    rxSize = size;
    memcpy(rxpacket, payload, size);
    rxpacket[size] = '\0';
    Radio.Sleep();
      Serial.printf("\r\nPaquete Recibido \"%s\" con RSSI %d , LONGITUD %d\r\n", rxpacket, rssi, rxSize);
    StrDataLoraIn = String(rxpacket);

    StrIndx = getValue(StrDataLoraIn, '@', 0);
    Val_Dist = (getValue(StrDataLoraIn, '@', 1)).toFloat();
    Val_Water = (getValue(StrDataLoraIn, '@', 2)).toFloat();
    relay = (getValue(StrDataLoraIn, '@', 3)).toInt();
    maxWaterLevel = (getValue(StrDataLoraIn, '@', 4)).toInt();
    minWaterLevel = (getValue(StrDataLoraIn, '@', 5)).toInt();
    if (StrIndx == "ArdEsp32")
    {
        Esp32_Dist = Val_Dist;
        Esp32_Water = Val_Water;
        relayState = relay == 1;
    }
```

```
    Serial.print(" Index: "); Serial.println(StrIndx);
    Serial.print(" Val_Dist: "); Serial.println(Val_Dist);
    Serial.print(" Val_Water: "); Serial.println(Val_Water);
    Serial.print(" Relay: "); Serial.println(relay);

    lora_idle = true;
}
// Se utiliza para dividir una cadena de texto en partes usando un separador y devolver una parte específica basada en el índice.
String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        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);  // Formatear a dos decimales
    if (client.connected()) {
        client.publish(feed.c_str(), value.c_str());
        Serial.println("Publicando al tópico: " + String(feed) + " | mensaje: " + value);
    }
}
void mqtt_publish(String feed, String val)
{
    if (client.connected()) {
        client.publish(feed.c_str(), val.c_str());
        Serial.println("Publicando al tópico: " + String(feed) + " | mensaje: " + val);
    }
}
// Función para enviar el estado del relé por LoRa
void sendRelayState()
{
    snprintf(txpacket, BUFFER_SIZE, "relay@%d", relayState ? 1 : 0);
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    Serial.println("Enviando estado del relé: " + String(relayState));
}
void sendMaxWaterLevel(String maxWaterLevel)
{
    snprintf(txpacket, BUFFER_SIZE, "max_water_level@%s", maxWaterLevel);
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    Serial.println("Enviando nivel máximo del agua: " + maxWaterLevel);
```

```
}
void sendMinWaterLevel(String minWaterLevel)
{
    snprintf(txpacket, BUFFER_SIZE, "min_water_level@%s", minWaterLevel);
    Radio.Send((uint8_t *)txpacket, strlen(txpacket));
    Serial.println("Enviando nivel mínimo del agua: " + minWaterLevel);
}
//*******************************************************
//  CONEXION A INTERNET
void setup_wifi()
{
    delay(10);

    // Conexión a 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 data por MQTT
void callback(char *topic, byte *payload, unsigned int length)
{
    String mensaje = "";
    String str_topic(topic);

    for (uint16_t i = 0; i < length; i++) {
        mensaje += (char)payload[i];
    }

    mensaje.trim();

    Serial.println("Tópico Ada: " + str_topic);
    Serial.println("Mensaje Ada: " + mensaje);

    if (str_topic == ADAFRUIT_FEED_RELAY) {
        if (mensaje == "ON") {
            relayState = true;
        } else if (mensaje == "OFF") {
            relayState = false;
        }
        sendRelayState();
    }
```

```
    if (str_topic == ADAFRUIT_FEED_MAX_WATER_LEVEL) {
        sendMaxWaterLevel(mensaje);
    }
    if (str_topic == ADAFRUIT_FEED_MIN_WATER_LEVEL) {
        sendMinWaterLevel(mensaje);
    }
}

// Capturar el ChipID para Id de MQTT
void get_MQTT_ID()
{
    uint64_t chipid = ESP.getEfuseMac();
    snprintf(ADAFRUIT_ID, sizeof(ADAFRUIT_ID), "%llu", chipid);
}

void reconnect()
{
    while (!client.connected())
    {
        if (client.connect(ADAFRUIT_ID, ADAFRUIT_USER, ADAFRUIT_KEY))
        {
            Serial.println("MQTT conectado!");
            client.subscribe(ADAFRUIT_DATA_IN);
            client.subscribe(ADAFRUIT_FEED_RELAY);
            client.subscribe(ADAFRUIT_FEED_MAX_WATER_LEVEL);
            client.subscribe(ADAFRUIT_FEED_MIN_WATER_LEVEL);
```

```
            Serial.println("Suscrito a los tópicos: " + String(ADAFRUIT_DATA_IN) + ", " +
String(ADAFRUIT_FEED_RELAY)                    +",                    "+
String(ADAFRUIT_FEED_MAX_WATER_LEVEL)+",                    "+
String(ADAFRUIT_FEED_MIN_WATER_LEVEL));
        }
        else
        {
            Serial.print("falló :( con error -> ");
            Serial.print(client.state());
            Serial.println(" Se intenta de nuevo en 5 segundos");
            delay(5000);
        }
    }
}
```

Código de *SoC* esclavo

```
//Librerias
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <NewPing.h>

// Definiciones de los pines conectados para el sensor HC-SR04 y Relé
#define TRIG_PIN 12  // Pin digital para el TRIG
#define ECHO_PIN 13  // Pin digital para el ECHO
#define RELAY_PIN 2 // Pin para el relé

//*********************************************************************
// Definicion de Constantes
#define SOUND_SPEED 0.0343  // Velocidad del sonido en cm/uS
```

```
#define RF_FREQUENCY 916000000 // Hz

#define TX_OUTPUT_POWER 14        // dBm

#define LORA_BANDWIDTH 0         // [0: 125 kHz,
                                 //  1: 250 kHz,
                                 //  2: 500 kHz,
                                 //  3: Reserved]
#define LORA_SPREADING_FACTOR 7  // [SF7..SF12]
#define LORA_CODINGRATE 1        // [1: 4/5,
                                 //  2: 4/6,
                                 //  3: 4/7,
                                 //  4: 4/8]
#define LORA_PREAMBLE_LENGTH 8   // Same for Tx and Rx
#define LORA_SYMBOL_TIMEOUT 0    // Symbols
#define LORA_FIX_LENGTH_PAYLOAD_ON false
#define LORA_IQ_INVERSION_ON false

#define RX_TIMEOUT_VALUE 1000
#define BUFFER_SIZE 30 // Define el tamaño del payload

//************************************************************************
//Variables y Funciones
char txpacket[BUFFER_SIZE];
char rxpacket[BUFFER_SIZE];
String StrDataLoraIn, StrIndx;
```

```
double txNumber;

bool lora_idle = true;

// filtros
int maxWaterLevel = 85;
int minWaterLevel = 5;
int relay = 0;
float Val_Dist, Val_Water = 0;

static RadioEvents_t RadioEvents;
void OnTxDone(void);
void OnTxTimeout(void);
void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr);
void turnOnRelay();
void turnOffRelay();
//*****************************************************************************

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

    txNumber = 0;

    pinMode(RELAY_PIN, OUTPUT);
    digitalWrite(RELAY_PIN, LOW); // Inicializa el relé apagado
```

```
    RadioEvents.RxDone = OnRxDone;
    RadioEvents.TxDone = OnTxDone;
    RadioEvents.TxTimeout = OnTxTimeout;



    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);


    // Inicialización de los pines del sensor HC-SR04
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);


    Radio.Rx(0);  // Habilitar el modo receptor al inicio
}
```

```
void loop() {
    if (lora_idle == true) {
        Radio.Rx(0);  // Habilitar continuamente el modo receptor
    }
    Radio.IrqProcess();
}

void turnOnRelay(){
  relay=1;
  digitalWrite(RELAY_PIN, HIGH); // Encender el relé
  Serial.println("Relé encendido");
}

void turnOffRelay(){
  relay=0;
  digitalWrite(RELAY_PIN, LOW); // Apagar el relé
  Serial.println("Relé apagado");
}

String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        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]) : "";
}

void OnRxDone(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)
{
    memcpy(rxpacket, payload, size);
    rxpacket[size] = '\0';
    Radio.Sleep();
   Serial.printf("\r\nPaquete Recibido \"%s\" con RSSI %d , LONGITUD %d\r\n", payload,
rssi, size);

    int command = 0;
    String event;

    StrDataLoraIn = String(rxpacket);

    event = getValue(StrDataLoraIn, '@', 0);
    command = (getValue(StrDataLoraIn, '@', 1)).toFloat();

    if(event == "relay"){
      if (command == 1) {
          turnOnRelay();
```

```
  } else if (command == 0) {
      turnOffRelay();
  }
}
if(event == "max_water_level"){
  maxWaterLevel = command;
}
if(event == "min_water_level"){
  minWaterLevel = command;
}

// Medición de la distancia utilizando el HC-SR04
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);

long duration = pulseIn(ECHO_PIN, HIGH);
Val_Dist = float(duration / 2.0) * SOUND_SPEED; // Convertir duración en cm

// Calcular el nivel de agua basado en la distancia
if (Val_Dist <= 2) {
    Val_Water = 100.0; // Nivel de agua lleno
} else if (Val_Dist >= 16) {
    Val_Water = 0.0; // Nivel de agua vacío
} else {
```

```
    // Interpolación lineal para calcular el nivel de agua entre 2 cm y 16 cm
    Val_Water = 100.0 - ((Val_Dist – 2) / (16 – 2) * 100.0);
  }
  if (isnan(Val_Dist) || isnan(Val_Water)) {
    Val_Dist = 0;
    Val_Water = 0;
    Serial.println(F("Error de lectura del sensor HC-SR04!"));
  }

  if(Val_Water >= maxWaterLevel){
   turnOffRelay();
  }else if(Val_Water <= minWaterLevel){
   turnOnRelay();
  };

  Serial.print("Val_Dist: ");
  Serial.print(Val_Dist);
  Serial.print(" cm  Val_Water: ");
  Serial.print(Val_Water);
  Serial.println(" %");

   Serial.printf("Relay status: %d\t MaxWaterLevel: %d\t MinWaterLevel: %d", relay, maxWaterLevel, minWaterLevel);

  // Crear el paquete con el formato requerido por el receptor
        sprintf(txpacket, "ArdEsp32@%.2f@%.2f,@%d@%d@%d", Val_Dist, Val_Water,relay,maxWaterLevel,minWaterLevel);
```

```
        Serial.printf("\r\nEnviando Paquete \"%s\" , longitud %d\r\n", txpacket, strlen(txpacket));
    Radio.Send((uint8_t *)txpacket, strlen(txpacket)); // Enviar el paquete
    lora_idle = false;
}

void OnTxDone(void) {
    Serial.println("TX done......");
    lora_idle = true;
    Radio.Rx(0);  // Habilitar el modo receptor después de la transmisión
}

void OnTxTimeout(void) {
    Radio.Sleep();
    Serial.println("TX Timeout......");
    lora_idle = true;
    Radio.Rx(0);  // Habilitar el modo receptor después del tiempo de espera
}
```