

CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Sistema de monitoreo de cultivos agrícolas

Autor:

Ing. Mario Fernando Aguilar Montoya

Director:

Esp. Ing. Julian Bustamante Narvaez (TECREA SAS)

Jurados:

Dr. Ing. Javier Andrés Redolfi (UTN-FRSF)
Mg. Lic. Leopoldo Zimperz (FIUBA)
Esp. Ing. Felipe Calcavecchia (FIUBA)

*Este trabajo fue realizado en la ciudad de Tarija,
entre junio de 2022 y agosto de 2023.*

Resumen

En la presente memoria se aborda el diseño e implementación de un sistema de adquisición de datos para el monitoreo de cultivos agrícolas realizado como emprendimiento personal. Este trabajo pretende ayudar al agricultor a gestionar de mejor manera sus recursos. Para su desarrollo fueron fundamentales los conocimientos adquiridos en la carrera tales como conceptos de modularización, testing de software, sistemas operativos en tiempo real, protocolos de comunicación y programación de microcontroladores.

Agradecimientos

En primer lugar a mi familia y amigos que siempre me brindan su apoyo y confianza.

A mi director por su paciencia, guía y consejo en todo momento.

A mis compañeros y maestros de la CESE, por los aportes y enseñanzas que me dieron a lo largo de la especialización.

Índice general

Resumen	I
1. Introducción general	1
1.1. IoT en la agricultura	1
1.1.1. Internet de las cosas	1
1.1.2. Sistemas de monitoreo de cultivos agrícolas	2
1.2. Estado del arte	2
1.2.1. Libelium	2
1.2.2. Nodo RF-M1 DropControl	3
1.3. Objetivo y alcances	4
1.3.1. Objetivo	4
1.3.2. Alcances	4
2. Introducción específica	5
2.1. Componentes principales de hardware	5
2.1.1. Plataforma de desarrollo STM32 NUCLEO-L432KC	5
2.1.2. Módulo de comunicación LTE IOT 2 CLICK	6
2.1.3. Sensor AHT10	7
2.1.4. Sensor ML8511	7
2.1.5. Sensor de humedad de suelo HL-69 (Resistivo)	7
2.2. Herramientas de software y testing utilizados	8
2.2.1. STM32 CubeIDE	8
2.2.2. FreeRTOS	8
2.2.3. CEDLING	9
2.3. Protocolos de Comunicación	9
2.3.1. UART	9
2.3.2. I2C	9
2.3.3. MQTT	9
2.4. Plataformas IoT	10
2.4.1. Ubidots	10
2.4.2. ThingsBoard	10
3. Diseño e implementación	11
3.1. Diagrama de bloques general del sistema	11
3.2. Arquitectura de firmware	12
3.2.1. Capa HAL	12
3.2.2. Capa drivers	13
3.2.3. Capa aplicación	13
3.3. Desarrollo del firmware	14
3.3.1. Tarea loop del sistema	14
3.3.2. Tarea adquisición de datos	15
3.3.3. Tarea manejador de alarmas	16
3.3.4. Tarea manejador del servidor	16

3.4.	Controladores implementados	18
3.4.1.	Controlador sensor AHT10	18
3.4.2.	Controlador módulo de comunicación BG96	19
3.5.	Desarrollo del hardware	23
3.5.1.	Esquemático	23
3.5.2.	PCB del hardware	25
3.5.3.	Fabricación circuito impreso	25
3.6.	Paneles de visualización	26
3.6.1.	Panel principal	26
3.6.2.	Panel nodo sensor	27
4.	Ensayos y resultados	29
4.1.	Pruebas unitarias drivers	29
4.2.	Pruebas de la plataforma IoT	32
4.2.1.	Pruebas de inyección de mensajes	32
4.2.2.	Prueba de la tabla de alarmas en thingsboard	33
4.2.3.	Prueba del widget de mapa	34
4.2.4.	Prueba de persistencia de datos	34
4.3.	Pruebas De Hardware	35
4.3.1.	Prueba comunicación sensor de humedad AHT10	35
4.3.2.	Pruebas comunicación modulo BG96	35
4.3.3.	Pruebas de alimentación	36
4.3.4.	Prueba microcontrolador en bajo consumo	36
4.4.	Pruebas Funcionales del sistema	37
4.4.1.	Prueba de lectura de sensores	37
4.4.2.	Prueba de envío de datos al broker mqtt	38
4.4.3.	Pruebas de alarmas	38
5.	Conclusiones	41
5.1.	Conclusiones generales	41
5.2.	Próximos pasos	41
Bibliografía		43

Índice de figuras

1.1. Módulo Smart Agriculture PRO.	3
1.2. Módulo RF-M1 de DropControl.	3
2.1. Plataforma de desarrollo NUCLEO-L432KC ¹	6
2.2. Módulo LTE IOT 2 CLICK ²	6
2.3. Sensor AHT10 ³	7
2.4. Módulo Sensor ML8511	7
2.5. Módulo sensor HL-69 ⁴	8
2.6. Logo FreeRTOS ⁵	8
2.7. Arquitectura de publicación/suscripción de MQTT ⁶	9
2.8. Ejemplo interfaz gráfica Ubidots ⁷	10
2.9. Ejemplo interfaz gráfica ThingsBoard ⁸	10
3.1. Diagrama general del sistema IoT.	11
3.2. Capas del firmware.	12
3.3. Diagrama de flujo de inicialización del firmware.	14
3.4. Diagrama de flujo de la tarea loop.	15
3.5. Diagrama de flujo tarea de adquisición de datos.	16
3.6. Diagrama de flujo de la tarea manejador de alarmas.	16
3.7. Diagrama de flujo de la tarea conexión server MQTT.	17
3.8. Máquina de estados down servidor.	17
3.9. Máquina de estados up servidor.	18
3.10. Esquemático página raíz.	23
3.11. Conector módulo BG96 y NUCLEO-L432KC.	23
3.12. Esquemático interfaz de debug.	24
3.13. Esquemático conectores sensores.	24
3.14. Esquemático conectores alimentación.	24
3.15. PCB del proyecto.	25
3.16. Modelo 3D de la tarjeta.	25
3.17. PCB ensamblado.	25
3.18. Panel principal de la interfaz gráfica.	26
3.19. Panel nodo sensor.	27
4.1. Informe de cobertura driver aht10.	31
4.2. Informe de cobertura driver BG96.	32
4.3. Envío de datos por el cliente MQTT de mosquitto.	32
4.4. Recepción de datos en el broker MQTT.	33
4.5. Tabla de alarmas activas.	33
4.6. Ubicación del nodo sensor implementado.	34
4.7. Persistencia de datos.	34
4.8. Trama de escritura al sensor AHT10.	35
4.9. Trama de lectura del sensor AHT10.	35
4.10. Envío y recepción de comandos por puerto UART.	35

4.11. Medición de voltaje.	36
4.12. Modos de consumo.	36
4.13. Implementación del prototipo.	37
4.14. Lectura de todos los sensores.	37
4.15. Comandos para enviar datos al broker mqtt.	38
4.16. Humedad menor y activación de la alarma.	39
4.17. Recepción de sms por la alarma.	39

Índice de tablas

Capítulo 1

Introducción general

En este capítulo se hace una breve introducción a la necesidad que condujo al desarrollo del trabajo. Se presenta el concepto de internet de las cosas o IoT (del inglés *Internet of Things*) y el estado del arte de dispositivos similares. Asimismo, se explica el objetivo y los alcances del trabajo.

1.1. IoT en la agricultura

En los últimos años la agricultura ha enfrentado muchos desafíos, desde una creciente población mundial a ser alimentada, hasta requisitos de sostenibilidad y restricciones ambientales debido al cambio climático y el calentamiento global.

La agricultura es uno de los sectores que más sufre la escasez de agua que existe actualmente en el mundo, uno de los objetivos de implementar la tecnología IoT en este sector, es el de lograr una gestión eficiente y sostenible de los recursos hídricos.

Esto obliga a implementar soluciones que permitan modernizar las prácticas agrícolas. En este contexto, la Agricultura 4.0 representa la última evolución de la agricultura de precisión. La misma se encuentra basada en el concepto de agricultura inteligente, donde convergen el uso de internet de las cosas, computación en la nube, aprendizaje automático para el análisis de grandes volúmenes de datos, vehículos no tripulados y robótica [1].

1.1.1. Internet de las cosas

El concepto de internet de las cosas se refiere a la interconexión digital de dispositivos y objetos a través de una red, es decir, dispositivos como sensores y/o actuadores, equipados con una interfaz de comunicación, unidades de procesamiento y almacenamiento. Estos dispositivos tienen la capacidad de adquirir, intercambiar y transferir datos a la red mediante alguna tecnología de comunicación inalámbrica [2].

El IoT puede usarse a favor de la sostenibilidad, no cabe duda de que Internet es un facilitador de iniciativas sostenibles. De acuerdo con el Foro Económico Mundial la mayoría de los proyectos con internet de las cosas se centran en la eficiencia energética en las ciudades, energías sostenibles y el consumo responsable [3].

Por ejemplo:

- Eficiencia energética: en este sector se interconectan sensores, algoritmos y redes de comunicación para anticipar la demanda eléctrica y así realizar una distribución sostenible de la energía para reducir el precio del kW.

- Uso del agua: esta tecnología pone en funcionamiento máquinas para recoger datos en tiempo real que permitan hacer uso eficiente del agua y reducir su consumo.

1.1.2. Sistemas de monitoreo de cultivos agrícolas

Los sistemas de monitoreo de cultivos agrícolas se encargan de monitorear las distintas variables ambientales a las que están expuestos los cultivos agrícolas, los datos adquiridos ayudan a la toma de decisiones y a manejar de una manera eficiente los recursos con los que cuentan los agricultores.

Cuentan con tres partes fundamentales:

- El nodo sensor, que en sí sería la parte física o hardware, es generalmente de bajo consumo.
- El firmware que abarca la lógica del sistema y se encarga de realizar la adquisición, procesamiento y transferencia de datos que puede o no estar sobre un sistema operativo de tiempo real.
- Nube o plataforma IoT, que ofrecen diferentes servicios como ser almacenamiento, procesamiento, análisis, visualización, etc. Esta parte del sistema permite al usuario del sistema poder visualizar los valores de las variables medidas y así poder tomar decisiones con respecto a las mediciones.

1.2. Estado del arte

Durante la etapa de investigación del trabajo se realizó la búsqueda de productos comerciales en el mercado local e internacional. Se encontraron algunos productos de similares características al que se pretende realizar, un dato interesante a resaltar es que todos los productos encontrados son del mercado internacional, no se encontró ningún producto o empresa que ofrezca este tipo de soluciones en el mercado local.

A continuación se describen los productos encontrados, estas opciones varían con respecto a la tecnología que utilizan.

1.2.1. Libelium

Smart Agriculture PRO figura 1.1 es un módulo de IoT que está diseñado para realizar monitoreo de viñedos para mejorar la calidad del vino, riego selectivo en campos de golf y control de condiciones en invernaderos, entre otros. Permite monitorear múltiples parámetros ambientales que involucran una amplia gama de aplicaciones, desde el análisis del desarrollo en crecimiento hasta la observación del clima. Para ello se ha dotado de sensores de temperatura y humedad del aire y del suelo, luminosidad, radiación solar, velocidad y dirección del viento, precipitaciones, presión atmosférica, humedad de las hojas, distancia y diámetro del fruto o tronco [4].



FIGURA 1.1. Módulo Smart Agriculture PRO.

1.2.2. Nodo RF-M1 DropControl

El nodo RF-M1 es adecuado para tareas de monitoreo simples como parte de una red DropControl o por sí solo. Posee una combinación de entradas que le permite realizar múltiples tareas de monitoreo y almacenarlas en la nube. En la figura 1.2 se muestra el módulo físicamente [5]. Características del dispositivo:

- Redes RF *mesh* o comunicación celular.
- Energía autónoma, solar + batería.
- Actualización del firmware vía aérea, configuraciones y soporte por internet.
- Protección externa IP65.
- Amplia variedad de compatibilidad con sensores.
- Unidad de bajo costo para resolver necesidades básicas de monitoreo.



FIGURA 1.2. Módulo RF-M1 de DropControl.

1.3. Objetivo y alcances

1.3.1. Objetivo

El objetivo principal del trabajo es el diseño e implementación de un prototipo funcional de un sistema de monitoreo de cultivos agrícolas.

1.3.2. Alcances

- Implementación de un prototipo funcional con hardware de bajo consumo.
- Desarrollo del firmware sobre un sistema operativo de tiempo real.
- Transmisión de la información por red celular.
- Visualización de los datos en Ubidots [6].

Capítulo 2

Introducción específica

En el presente capítulo se describen los componentes de hardware, software, protocolos de comunicación y plataformas IoT utilizados para realizar el trabajo.

2.1. Componentes principales de hardware

2.1.1. Plataforma de desarrollo STM32 NUCLEO-L432KC

La placa STM32 Nucleo-L432KC que se muestra en la figura 2.1 proporciona una forma asequible y flexible para que los usuarios prueben nuevos conceptos y construyan prototipos eligiendo entre las diversas combinaciones de funciones de rendimiento y consumo de energía que proporciona el microcontrolador STM32L4KC [7].

Características:

- Microcontrolador STM32L4KC en paquete 32 de pines.
- Led de usuario.
- Pulsador de reset.
- Conector de expansión Arduino Nano V3.
- Conector USB Micro-AB para ST-LINK.
- Opciones flexibles de fuente de alimentación.
- Depurador/programador ST-LINK integrado.
- Compatibilidad con una amplia variedad de entornos de desarrollo integrado.
- Oscilador de cristal de 24 MHz.
- Compatible con Arm Mbed Enabled.



FIGURA 2.1. Plataforma de desarrollo NUCLEO-L432KC¹.

2.1.2. Módulo de comunicación LTE IOT 2 CLICK

LTE IoT 2 click que se muestra en la 2.2 está equipado con el módulo BG96 LTE de Quectel Wireless Solutions, que admite tecnologías LTE CAT M1 y NB1, desarrolladas para aplicaciones IoT. Además, admite EGPRS a 850/900/1800/1900 MHz, lo que significa que se puede usar globalmente; no está restringido a ninguna región [8].

Características:

- Protocolos de internet integrados (TCP/UDP/PPP).
- Conectores SMA integrados.
- Leds de alimentación e indicación de estado.
- Conector USB para conectarlo con la aplicación de software de Quectel.
- Interfaz UART.
- Tensión de alimentación 5 V o 3,3 V.



FIGURA 2.2. Módulo LTE IOT 2 CLICK².

¹Datasheet https://www.st.com/resource/en/user_manual/um1956-stm32-nucleo32-boards-mb1180-stmicroelectronics.pdf

²Imagen tomada de la página <https://www.mikroe.com/lte-iot-2-click>

2.1.3. Sensor AHT10

El sensor AHT10 presentado en la figura 2.3 permite obtener lecturas de temperatura y humedad, es de bajo costo y excelente rendimiento. Se utiliza este sensor en aplicaciones de control automático de temperatura, aire acondicionado, estaciones meteorológicas, aplicaciones en el hogar, regulador de humedad y temperatura [9].



FIGURA 2.3. Sensor AHT10³.

2.1.4. Sensor ML8511

El módulo ML8511 presentado en la figura 2.4 es un sensor de luz ultravioleta (UV), entrega una señal de tensión analógica que depende de la cantidad de luz UV que detecta. Sensor ideal para proyectos de monitoreo de condiciones ambientales como el índice UV, aplicaciones meteorológicas, cuidado de la piel, medición industrial de nivel UV. El sensor ML8511 detecta luz con una longitud de onda entre 280-390 nm, este rango cubre tanto al espectro UV-B como al UV-A. La salida analógica está relacionada linealmente con la intensidad UV (mW/cm^2) [10].



FIGURA 2.4. Módulo Sensor ML8511.⁴

2.1.5. Sensor de humedad de suelo HL-69 (Resistivo)

El módulo HL-69 presentado en la figura 2.5, un sensor de humedad de suelo que utiliza la conductividad entre dos terminales para determinar parámetros relacionados al agua, líquidos y humedad [11].

³Imagen tomada de la página <https://esphome.io/components/sensor/aht10.html>

⁴Manual del sensor <https://learn.sparkfun.com/tutorials/ml8511-uv-sensor-hookup-guide/all>

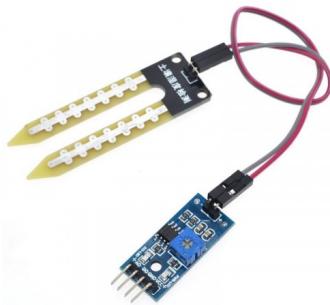


FIGURA 2.5. Módulo sensor HL-69⁵.

2.2. Herramientas de software y testing utilizados

2.2.1. STM32 CubeIDE

STM32CubeIDE es una herramienta de desarrollo multi-OS todo en uno, que forma parte del ecosistema de software STM32Cube. Se trata de una plataforma de desarrollo C/C++ que permite compilar y depurar código para los microcontroladores STM32. Se basa en el marco Eclipse y la cadena de herramientas GCC para el desarrollo y GDB para la depuración. Permite la integración de los cientos de plugins existentes que completan las funcionalidades del IDE de Eclipse [12].

Integra las funcionalidades de configuración y creación de proyectos de STM32 de STM32CubeMX para ofrecer una experiencia de herramienta todo en uno y ahorrar tiempo de instalación y desarrollo [12].

2.2.2. FreeRTOS

FreeRTOS es un sistema operativo en tiempo real (RTOS) líder en el mercado para microcontroladores y pequeños microprocesadores. Distribuido libremente bajo la licencia de código abierto del MIT, FreeRTOS incluye un núcleo y un conjunto creciente de bibliotecas adecuadas para su uso en todos los sectores de la industria. FreeRTOS está diseñado con énfasis en la confiabilidad, la accesibilidad y la facilidad de uso [13]. El logo de FreeRTOS se muestra en la figura 2.6.



FIGURA 2.6. Logo FreeRTOS⁶.

⁵Tienda <https://tienda.sawers.com.bo/hl-69-modulo-sensor-humedad-suelo>

⁶Imagen tomada de la página <https://www.freertos.org/>

2.2.3. CEDLING

Ceedling es un sistema de compilación diseñado para proyectos en lenguaje C, que podría describirse como una extensión del sistema de compilación Rake (similar a make) de Ruby. Ceedling está dirigido principalmente al desarrollo basado en pruebas en C y está diseñado para reunir CMock, Unity y CException [14].

2.3. Protocolos de Comunicación

2.3.1. UART

UART (*Universal Asynchronous Receiver / transmitter*, por sus siglas en inglés) define un protocolo o un conjunto de normas para el intercambio de datos en serie entre dos dispositivos. UART es sumamente simple y utiliza solo dos hilos entre el transmisor y el receptor para transmitir y recibir en ambas direcciones. Ambos extremos tienen una conexión a masa. La comunicación en UART puede ser simplex (los datos se envían en una sola dirección), semidúplex (cada extremo se comunica, pero solo uno al mismo tiempo), o dúplex completo (ambos extremos pueden transmitir simultáneamente). Los datos se transmiten en forma de tramas [15].

2.3.2. I2C

El protocolo I2C (*Inter-Integrated Circuit*) es un protocolo diseñado para permitir la comunicación entre múltiples circuitos integrados digitales y uno o más chips controladores. Similar a la interfaz periférica en serie, está destinado a comunicaciones de corta distancia dentro de un solo dispositivo. Al igual que las interfaces seriales asíncronas, solo requiere dos cables de señal para el intercambio de información [16].

2.3.3. MQTT

MQTT es un protocolo de mensajería estándar de OASIS para IoT. Está diseñado como un transporte de mensajería de publicación/suscripción extremadamente liviano que es ideal para conectar dispositivos remotos con un espacio de código pequeño y un ancho de banda de red mínimo. MQTT hoy en día se utiliza en una amplia variedad de industrias, como la automotriz, la manufactura, las telecomunicaciones, el petróleo y el gas, etc [17]. En la figura 2.7 se muestra la arquitectura del protocolo MQTT.

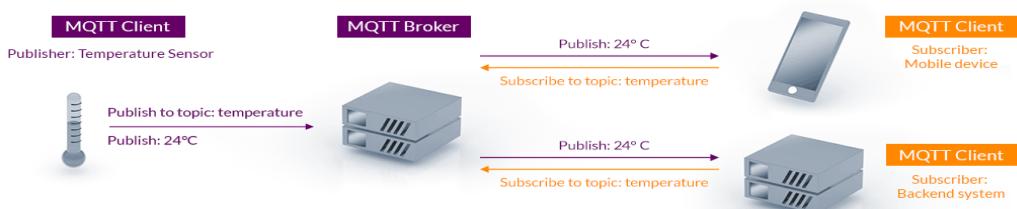


FIGURA 2.7. Arquitectura de publicación/suscripción de MQTT⁷.

⁷Imagen tomada de la página <https://mqtt.org/>

2.4. Plataformas IoT

2.4.1. Ubidots

Ubidots permite enviar datos de sensores a la nube, configurar tableros y alertas, conectarse con otras plataformas, usar herramientas de analítica y arrojar mapas de datos en tiempo real [6]. En la figura 2.8 se muestra un ejemplo de interfaz gráfica en Ubidots.

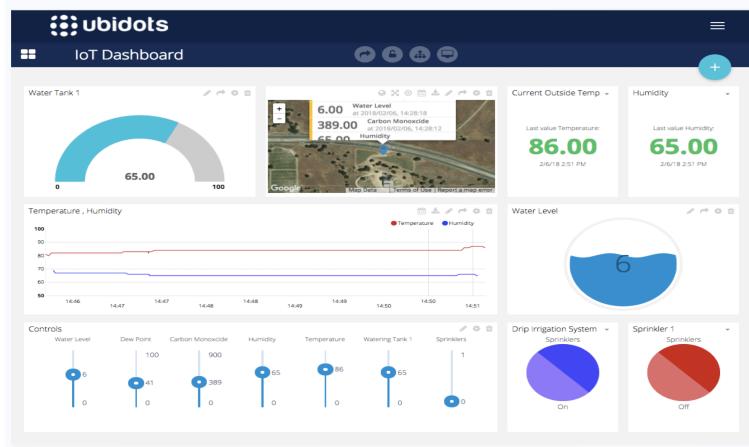


FIGURA 2.8. Ejemplo interfaz gráfica Ubidots⁸.

2.4.2. ThingsBoard

ThingsBoard es una plataforma IoT de código abierto para la recopilación, el procesamiento, la visualización y la gestión de dispositivos. Permite la conectividad de dispositivos a través de protocolos estándares de la industria IoT: MQTT, CoAP y HTTP. ThingsBoard combina escalabilidad, tolerancia a fallas y rendimiento [18]. En la figura 2.9 se muestra un ejemplo de una interfaz gráfica desarrollada en ThingsBoard.

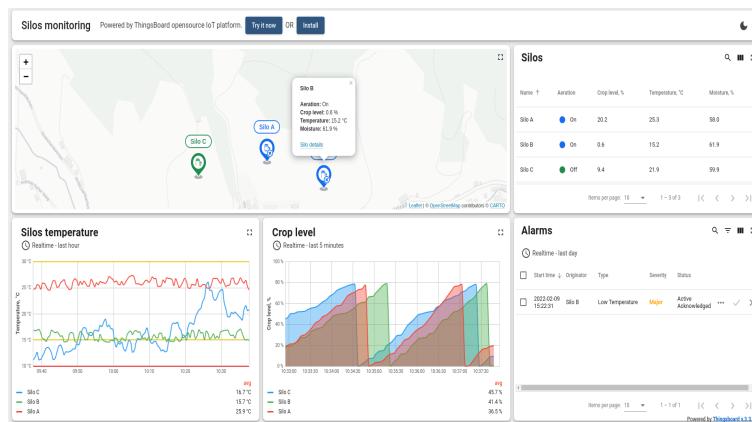


FIGURA 2.9. Ejemplo interfaz gráfica ThingsBoard⁹.

⁸Imagen tomada de la página <https://ubidots.com/platform/time-series>

⁹Imagen tomada de la página <https://thingsboard.io/smart-farming/>

Capítulo 3

Diseño e implementación

En este capítulo se abordará la descripción de la arquitectura general del sistema, arquitectura del firmware, código de los controladores desarrollados, desarrollo del hardware y la configuración de la plataforma IoT.

3.1. Diagrama de bloques general del sistema

En la figura 3.1 se muestra el diagrama en bloques general del sistema donde se describe la arquitectura IoT aplicada al trabajo que consta de tres capas: percepción, red y aplicación.

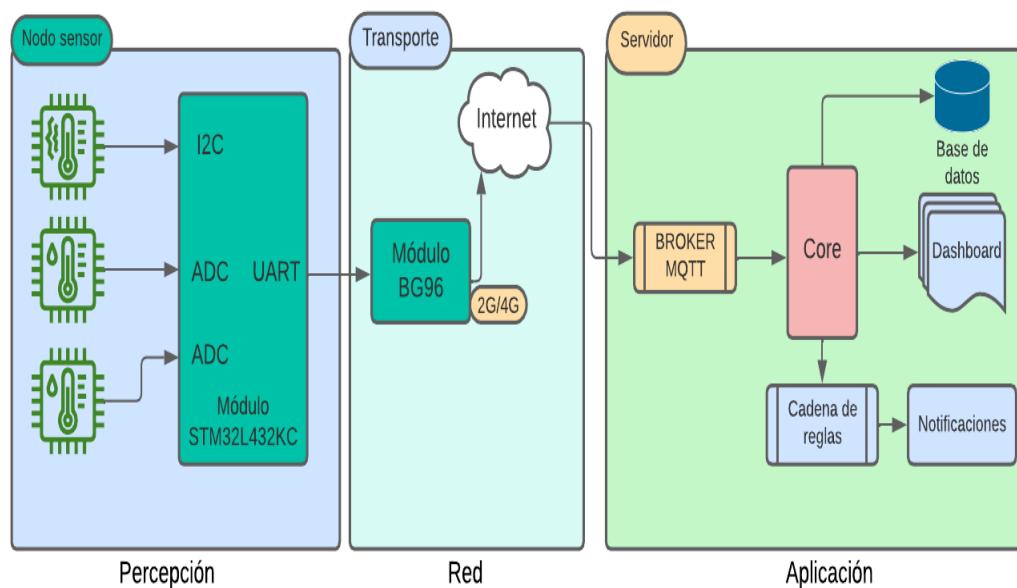


FIGURA 3.1. Diagrama general del sistema IoT.

En cada una de las capas se despliegan tecnologías y componentes de hardware y software. A continuación se describe cada una de las capas.

- Capa de percepción: en la capa de percepción se encuentra el nodo sensor que es el encargado de medir las variables físicas, hacer un preprocesamiento y posteriormente enviar los datos a la capa de red. Para su desarrollo se utilizó la tarjeta de prueba STM32L432KC que contiene el firmware del sistema, además, se utilizaron los siguientes sensores: sensor de humedad y temperatura ambiente AHT10, sensor de humedad de suelo HL-69 y el sensor de luz UV ML8511.

- Capa de red: en lo que respecta a la conectividad de red, se empleó un módulo Quectel BG96 que es capaz de establecer conexión de manera automática con las redes 2G, 4G y NB-IoT, según las condiciones de red en el lugar de implementación del nodo sensor. Este módulo se comunica con el microcontrolador mediante comandos AT a través del puerto UART.
- Capa de aplicación: en la capa de aplicación, se utilizó ThingsBoard como plataforma IoT que brinda los microservicios de broker MQTT como puerta de entrada al servidor, base de datos para el almacenamiento, interfaz gráfica para la visualización de los datos y permite gestionar las alarmas del sistema.

3.2. Arquitectura de firmware

El desarrollo del firmware fue la tarea más compleja del trabajo debido a que uno de los objetivos fue lograr un firmware estructurado en capas para facilitar el desarrollo y reducir la complejidad del código. La figura 3.2 muestra la división en capas del firmware desarrollado.

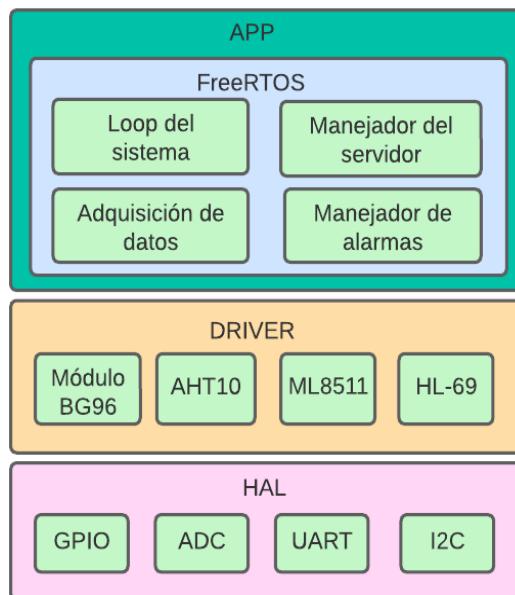


FIGURA 3.2. Capas del firmware.

3.2.1. Capa HAL

La capa HAL (*Hardware Abstraction Layer*) proporcionada por el fabricante del microcontrolador es la más baja del sistema, proporciona a las capas superiores la capacidad de interactuar con los periféricos del microcontrolador a través de funciones en lenguaje C.

- GPIO: la API proporciona funciones para gestionar las entradas y salidas del microcontrolador. Fueron utilizadas por la capa de APP para el control de los leds de debug y para el encendido y reset del módulo de comunicación.

- ADC: proporcionan funciones para la configuración, lectura y escritura de los pines del microcontrolador para trabajar con señales analógicas. Se utilizaron estas funciones para hacer la lectura de los sensores de humedad del suelo y el sensor de luz UV.
- UART: brinda funciones para la lectura y escritura del puerto UART del microcontrolador. El firmware utiliza estas funciones para la comunicación con el módulo BG96.
- I2C: proporciona funciones para la lectura y escritura por protocolo I2C. El driver del sensor AHT10 utiliza estas funciones para hacer la lectura de los datos.

3.2.2. Capa drivers

La capa drivers (manejador de dispositivos) está compuesta por los manejadores que se desarrollaron para interactuar con el hardware externo al microcontrolador. Se desarrollaron dos drivers que se describen a continuación:

- Driver BG96: las funciones más importantes que proporciona el driver son:
 - Estado del módulo.
 - Descripción del módulo.
 - Configuración APN de la red.
 - Conexión TCP.
 - Conexión al broker MQTT.
- Driver AHT10: se desarrolló utilizando la hoja de datos del sensor, proporciona funciones de inicialización y lectura de humedad y temperatura obtenidos por el sensor.

Para los sensores ML8511 y HL-69 que son sensores analógicos no se desarrollaron drivers, sino que se crearon funciones para convertir el valor analógico entregado a un valor significativo para el usuario con respecto a la variable física medida.

3.2.3. Capa aplicación

La capa de APP o aplicación es la de mayor nivel jerárquico. Se desarrolló sobre freeRTOS que permite hacer un código más escalable.

Se implementaron cuatro tareas.

- Loop del sistema: esta tarea es la que brinda la secuencialidad del sistema.
- Manejador del servidor: se encarga de manejar la conexión a la red y al broker MQTT.
- Adquisición de datos: se encarga de hacer la lectura de los sensores.
- Manejador de alarmas: esta tarea se encarga de hacer el control de las alarmas del sistema.

3.3. Desarrollo del firmware

Para el desarrollo del firmware se utilizó STM32CubeIDE que es el entorno de desarrollo oficial de STMicroelectronic.

El firmware fue desarrollado sobre freeRTOS, se utilizaron algunas de sus funcionalidades como colas, semáforos, tareas e interrupciones.

En la figura 3.3 se muestra en diagrama de flujo de inicialización del firmware.



FIGURA 3.3. Diagrama de flujo de inicialización del firmware.

El firmware comienza realizando la siguiente secuencia de acciones: configuración del hardware del microcontrolador, inicialización de los drivers del hardware externo, creación de los recursos del sistema operativo y finalmente inicialización del scheduler.

Para el control del sistema se crearon cuatro tareas sobre freeRTOS, que se comunican y sincronizan a través de colas y semáforos.

3.3.1. Tarea loop del sistema

La secuencialidad del sistema es manejada por la tarea loop, la figura 3.4 muestra el diagrama de flujo de la tarea loop. La tarea comienza iniciando el timer del sistema el cual es el encargado de manejar el tiempo en que se repite el ciclo de la tarea loop, luego la tarea se bloquea en el un semáforo que sera desbloqueada en el momento de la ejecución del handler de la interrupción del timer, luego se envia datos por la cola de adquisición de datos y server mqtt para levantar el servidor y la tarea se vuelve a bloquear en el semáforo, cuando se desbloquea pregunta si se logró levantar el servidor, si se logró se manda un mensaje por la cola de server mqtt con el evento de enviar datos y se bloquea nuevamente pero si no se logró levantar el servidor no se enviarán datos, luego se envía un evento

a la cola de alarmas y se bloquea nuevamente esperando que envíen las alarmas, cuando se desbloquea se manda un evento a la cola de server mqtt para desconectar el servidor y se bloquea la tarea en el semáforo esperando la desconexión finalmente se inicia nuevamente el timer y mandamos al sistema a modo de bajo consumo.

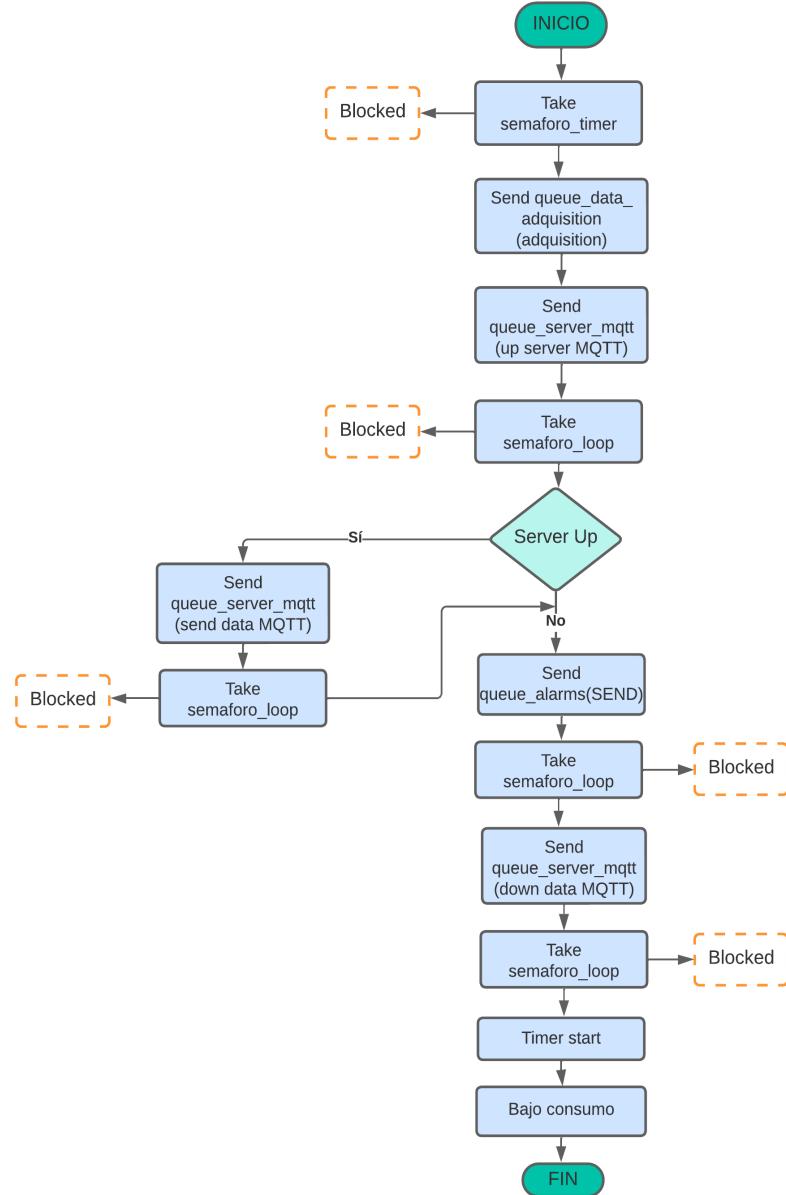


FIGURA 3.4. Diagrama de flujo de la tarea loop.

3.3.2. Tarea adquisición de datos

La figura 3.5 muestra el diagrama de flujo de la tarea de adquisición de datos. La tarea inicia revisando si hay datos en la cola de adquisición. Si existen datos se realiza la lectura de todos los sensores, para posteriormente enviar los valores leídos por la cola de datos y alarmas. Si no hay datos en la cola la tarea se bloquea.

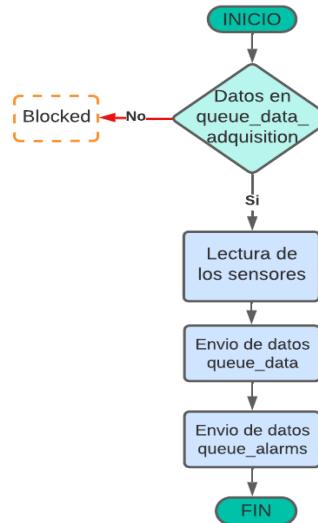


FIGURA 3.5. Diagrama de flujo tarea de adquisición de datos.

3.3.3. Tarea manejador de alarmas

El control de las alarmas se realiza a través una tarea del sistema operativo, la figura 3.6 muestra el diagrama de flujo de la tarea que maneja las alarmas.

Al entrar al bucle infinito lo primero que realiza la tarea es revisar la cola de alarmas. Si existen datos en la cola, se analiza el evento que contiene el dato recibido. Se tiene dos posibles eventos: monitorear y enviar. Si el evento es monitorear se compara el valor del sensor de humedad con un valor mínimo permitido, si es menor se activa una alarma advierte al sistema que ocurrió esto a través de una variable. Si el evento es enviar se revisa si hay alarmas activas, enviando un mensaje de texto si así fuera. Si no hay datos en la cola de alarmas la tarea se bloquea.

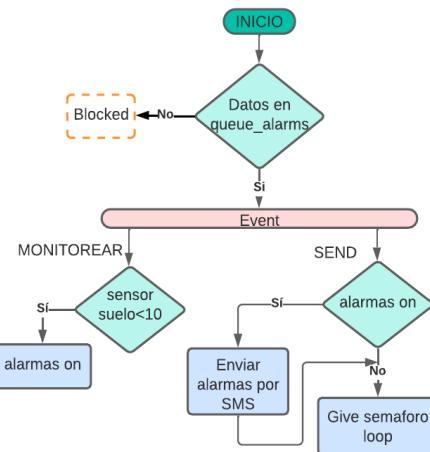


FIGURA 3.6. Diagrama de flujo de la tarea manejador de alarmas.

3.3.4. Tarea manejador del servidor

La tarea comienza esperando datos en la cola del servidor, al llegar datos se analiza el evento que se recibe. Se tiene tres posibles eventos: UP, DOWN y SEND. Si el evento es UP la tarea entra a la máquina de estados que se muestra en la figura 3.9 encargada de levantar una conexión con el servidor. Si el evento es DOWN

la tarea ejecuta la máquina de estados que se ve en la figura 3.8 que se encarga de terminar la conexión con el servidor. Si el evento es SEND la tarea obtiene los últimos datos leídos por los sensores, armar la trama y publica los datos al broker MQTT.

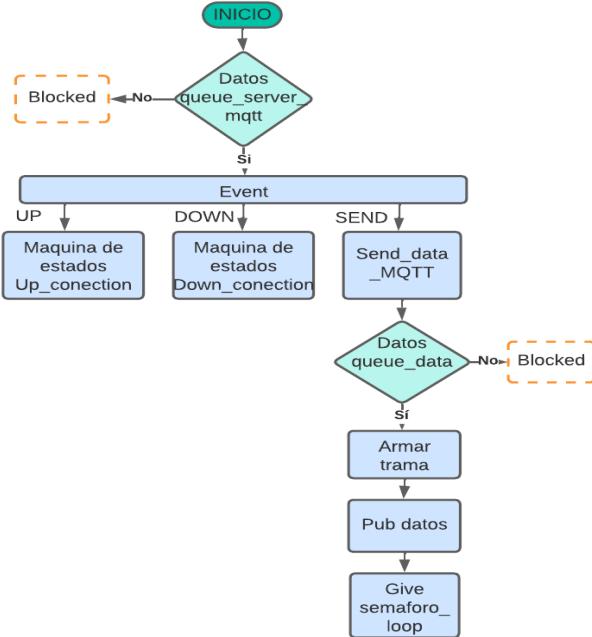


FIGURA 3.7. Diagrama de flujo de la tarea conexión server MQTT.

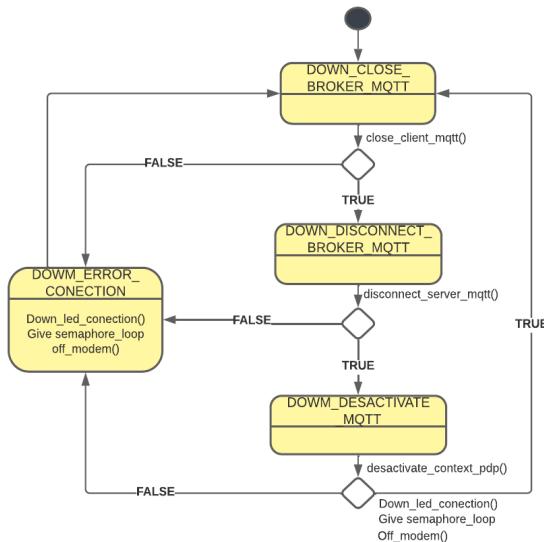


FIGURA 3.8. Máquina de estados down servidor.

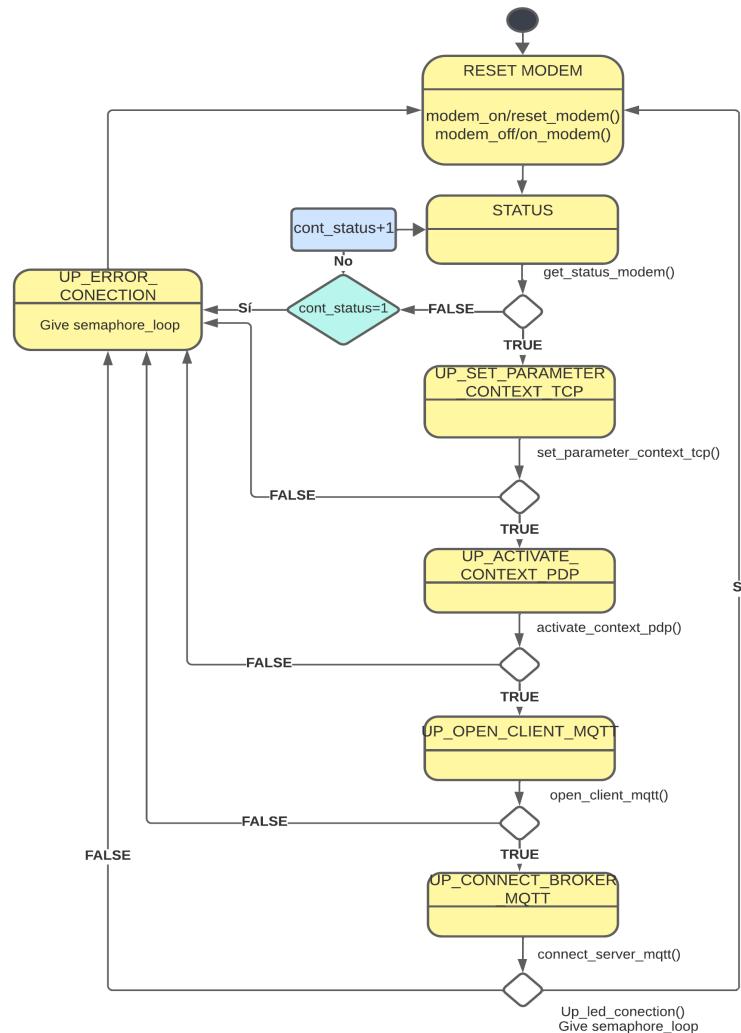


FIGURA 3.9. Máquina de estados up servidor.

3.4. Controladores implementados

Se implementaron dos controladores para el modulo de comunicacion BG96 y para el sensor de humedad y temperatura ambiente AHT-10.

3.4.1. Controlador sensor AHT10

En el código 3.1 se ven las funciones mas utilizadas del driver en el firmware.

```

1 //Estructura para manejar los datos del driver del sensor
2 typedef struct
3 {
4     aht10WriteFcn_t    writeI2C;
5     aht10ReadFcn_t    readI2C;
6     delay1ms_t        delay_ms_I2C;
7     aht10_status_fnc status_fun;
8 } aht10_config_t;
9
10 //Funcion para inicializar el driver
  
```

```

11 void aht10Init(aht10_config_t *obj, aht10WriteFcn_t fncWritePort,
12                 aht10ReadFcn_t fncReadPort, delay1ms_t fncDelayPort)
13 {
14     obj->writeI2C=fncWritePort;
15     obj->readI2C=fncReadPort;
16     obj->delay_ms_I2C=fncDelayPort;
17 }
18 //Funcion para obtener el valor de la humedad
19 aht10_status_fnc aht10_get_humidity(aht10_config_t*obj, uint8_t *data)
20 {
21     if (obj== NULL)
22     {
23         return AHT10_ERROR;
24     }
25     obj->status_fun=AHT10_ERROR;
26     uint8_t bufferRead [6]={0};
27     uint32_t data_humidity=0;
28     obj->status_fun=aht10_launch_measurement(obj);
29     if (obj->status_fun==AHT10_OK)
30     {
31         obj->status_fun= obj->readI2C(AHT10_ADDRESS_SLAVE,bufferRead ,6);
32         if (obj->status_fun==AHT10_OK)
33         {
34             data_humidity=(((uint32_t)bufferRead[1]<<16) | ((uint16_t)
35             bufferRead[2]<<8) | (bufferRead [3]))>>4;
36             *data= HUMEDITY(data_humidity);
37         }
38     }
39 }
40 //Funcion para obtener el valor de la temperatura
41 aht10_status_fnc aht10_get_temperature(aht10_config_t*obj, int8_t *data)
42 {
43     if (obj== NULL)
44     {
45         return AHT10_ERROR;
46     }
47     uint8_t buffer_read [6]={0};
48     uint32_t data_temperature=0;
49     obj->status_fun=AHT10_ERROR;
50     obj->status_fun=aht10_launch_measurement(obj);
51     if (obj->status_fun==AHT10_OK)
52     {
53         obj->status_fun=obj->readI2C(AHT10_ADDRESS_SLAVE ,buffer_read ,6);
54         if (obj->status_fun==AHT10_OK)
55         {
56             data_temperature=((uint32_t)(buffer_read [3] & 0x0F)<<16) | ((
57             uint16_t) buffer_read[4]<<8)| buffer_read [5];
58             *data= TEMPERATURE(data_temperature);
59         }
60     }
61 }
62 
```

CÓDIGO 3.1. Funciones principales del driver del sensor AHT10.

3.4.2. Controlador módulo de comunicación BG96

En el código 3.2 se muestran las funciones mas utilizadas por el firmware del driver que son las de configuracion y activacion del APN, conexion y desconexion al servidor MQTT.

```

1 //Estructura para manejar los datos del driver
2 typedef struct
3 {
4     em_status_modem status_modem;
5     em_state_server_mqtt_conection status_mqtt_server;
6     pf_send_data    send_data_device;
7     pf_reset_modem f_reset_modem;
8     st_config_parameters_mqtt self_mqtt;
9     st_config_context_tcp self_tcp;
10    st_info_product info_product;
11    uint8_t          last_error;
12    char buffer_resp [100];
13    char *current_cmd;
14    em_bg96_error_handling ft_resp;
15 }st_bg96_config;
16
17 //Funcion de inicio del driver
18 em_bg96_error_handling init_driver(st_bg96_config *self ,pf_send_data
19                                ft_send_data_device ,pf_reset_modem ft_reset_modem)
20 {
21     if (ft_send_data_device!=NULL) {
22         self->send_data_device=ft_send_data_device;
23     }
24     if (ft_reset_modem!=NULL) {
25         self->f_reset_modem=ft_reset_modem;
26     }
27     self->status_modem=OFF;
28     self->ft_resp=FT_BG96_OK;
29     self->last_error=BG96_NO_ERROR;
30     self->self_tcp.context_id=1;
31     self->self_tcp.context_type=1;
32     self->self_tcp.method_authentication=1;
33     self->self_tcp.tcp_password="";
34     self->self_tcp.tcp_username="";
35     self->self_mqtt.identifier_socket_mqtt=0;
36     self->self_mqtt.quality_service=0;
37     self->self_mqtt.port=1883;
38     self->self_mqtt.mqtt_client_id="123a56cb9";
39     self->status_mqtt_server=SERVER_MQTT_DOWN;
40     self->self_mqtt.host_name="\\"mqtt.thingsboard.cloud\"";
41     self->self_mqtt.mqtt_username="";
42     self->self_mqtt.mqtt_password="";
43     self->self_tcp.tcp_apn="internet.tigo.bol";
44     return self->ft_resp;
45 }
46 //Funcion para obtener el estado del modulo
47 em_bg96_error_handling get_status_modem(st_bg96_config* self)
48 {
49     self->ft_resp=FT_BG96_OK;
50     self->current_cmd="AT\r";
51     self->ft_resp=self->send_data_device(self->current_cmd ,RS_BG96_OK, self
52           ->buffer_resp ,1000);
53     if (self->ft_resp !=FT_BG96_OK)
54     {
55         self->last_error=BG96_ERROR_STATUS_MODEM;
56     }
57     return self->ft_resp ;
58 }
59 //Funcion para enviar mensaje de texto
60 em_bg96_error_handling send_sms_bg96(st_bg96_config *self ,char*number,
61                                     char*message)
62 {
63     self->ft_resp=FT_BG96_OK;

```

```

61     char buffer_message[256];
62     char buffer_number[20];
63     sprintf(buffer_number , "AT+CMGS=\"%s\"\r",number);
64     sprintf(buffer_message , "%\x1a\r",message);
65
66     self->ft_resp=self->send_data_device(buffer_number,RS_BG96_SIGNAL,self
67     ->buffer_resp,12000);
68     if (FT_BG96_OK==self->ft_resp)
69     {
70         self->ft_resp=self->send_data_device(buffer_message,RS_BG96_OK,self
71         ->buffer_resp,12000);
72         if (FT_BG96_OK!=self->ft_resp)
73         {
74             self->last_error=BG96_ERROR_SEND_SMS;
75         }
76     }
77 //Funcion para configurar el contexto de la red
78 em_bg96_error_handling set_parameter_context_tcp(st_bg96_config *self)
79 {
80     self->ft_resp=FT_BG96_OK;
81     char cmd[100];
82     sprintf(cmd, "AT+QICSGP=%u,%u,\\""%s\"",\""%s\"",\""%s\"",%u\r",self->self_tcp
83     .context_id, self->self_tcp.context_type, self->self_tcp.tcp_apn, self
84     ->self_tcp.tcp_username, self->self_tcp.tcp_password, self->self_tcp.
85     method_authentication);
86     self->ft_resp=self->send_data_device(cmd,RS_BG96_OK, self->buffer_resp
87     ,3000);
88     if (self->ft_resp !=FT_BG96_OK)
89     {
90         self->last_error=BG96_ERROR_SET_PARAMETER_CONTEXT_TCP;
91     }
92     return self->ft_resp;
93 }
94 //Funcion para activar el contexto pdp
95 em_bg96_error_handling activate_context_pdp(st_bg96_config *self)
96 {
97     self->ft_resp=FT_BG96_OK;
98     char cmd[30];
99     sprintf(cmd, "AT+QIACT=%u\r",self->self_tcp.context_id);
100    self->ft_resp=self->send_data_device(cmd,RS_BG96_OK, self->buffer_resp
101     ,15000);
102    if (self->ft_resp !=FT_BG96_OK)
103    {
104        self->last_error=BG96_ERROR_ACTIVATE_CONTEXT_PDP;
105    }
106    return self->ft_resp;
107 }
108 //Funcion para abrir un cliente MQTT
109 em_bg96_error_handling open_client_mqtt(st_bg96_config *self)
110 {
111     self->ft_resp=FT_BG96_ERROR;
112     char cmd[100];
113     sprintf(cmd, "AT+QMTOOPEN=%u,%s,%u\r",self->self_mqtt.
114     identifier_socket_mqtt, self->self_mqtt.host_name, self->self_mqtt.
115     port);
116     self->ft_resp=self->send_data_device(cmd,RS_BG96_CERO, self->
117     buffer_resp,75000);
118     if (self->ft_resp !=FT_BG96_OK)
119     {
120         self->last_error=BG96_ERROR_OPEN_CLIENT_MQTT;
121     }

```

```

114     return self->ft_resp;
115 }
116 //Funcion para conectarse al servidor MQTT
117 em_bg96_error_handling connect_server_mqtt(st_bg96_config *self)
118 {
119     self->ft_resp=FT_BG96_ERROR;
120     char cmd[150]={0};
121     sprintf(cmd, "AT+QMTCNN=%u,\"%s\", \"%s\", \"%s\"\r", self->self_mqtt.
122         identifier_socket_mqtt, self->self_mqtt.mqtt_client_id, self->
123         self_mqtt.username, self->self_mqtt.mqtt_password);
124     self->ft_resp=self->send_data_device(cmd,RS_BG96_CERO, self->
125         buffer_resp,10000);
126     if (self->ft_resp!=FT_BG96_OK)
127     {
128         self->last_error=BG96_ERROR_CONNECT_SERVER_MQTT;
129     }
130     return self->ft_resp;
131 }
132 //Funcion para publicar un mensaje al topico configurado
133 em_bg96_error_handling publish_message(st_bg96_config *self ,char *topic ,
134                                         char *data)
135 {
136     self->ft_resp=FT_BG96_ERROR;
137     char cmd[50]={0};
138     char buffer_data[220]={0};
139     sprintf(buffer_data, "%\x1a\r",data);
140     sprintf(cmd, "AT+QMTPUB=%u,0,0,0,\"%s\"\r", self->self_mqtt.
141         identifier_socket_mqtt, topic);
142     self->ft_resp=self->send_data_device(cmd,RS_BG96_SIGNAL, self->
143         buffer_resp,3000);
144     if (FT_BG96_OK==self->ft_resp)
145     {
146         self->ft_resp=self->send_data_device(buffer_data,RS_BG96_CERO, self->
147             buffer_resp,15000);
148         if (self->ft_resp!=FT_BG96_OK)
149         {
150             self->last_error=BG96_ERROR_PUBLISH_MESSAGE;
151         }
152     }
153     else self->last_error=BG96_ERROR_PUBLISH_MESSAGE;
154     return self->ft_resp;
155 }
```

CÓDIGO 3.2. Funciones principales del driver del sensor AHT10.

3.5. Desarrollo del hardware

Para el diseño del hardware se empleó KiCad 6.0, una herramienta de diseño que se utilizó durante el desarrollo de la especialización.

3.5.1. Esquemático

Al ser un prototipo lo que se hizo fue desarrollar un tarjeta donde se puedan ensamblar y conectar los módulos utilizados en el trabajo: módulo de comunicación celular, tarjeta de desarrollo con el microcontrolador y los módulos sensores.

En la figura 3.10 se muestra la página raíz del esquemático del trabajo, está dividida en tres zonas:

- Zona 1: índice de las hojas esquemáticas del trabajo.
 - Zona 2: modelo 3D de la tarjeta desarrollada.
 - Zona 3: conexiones entre los diferentes hojas esquemáticas .

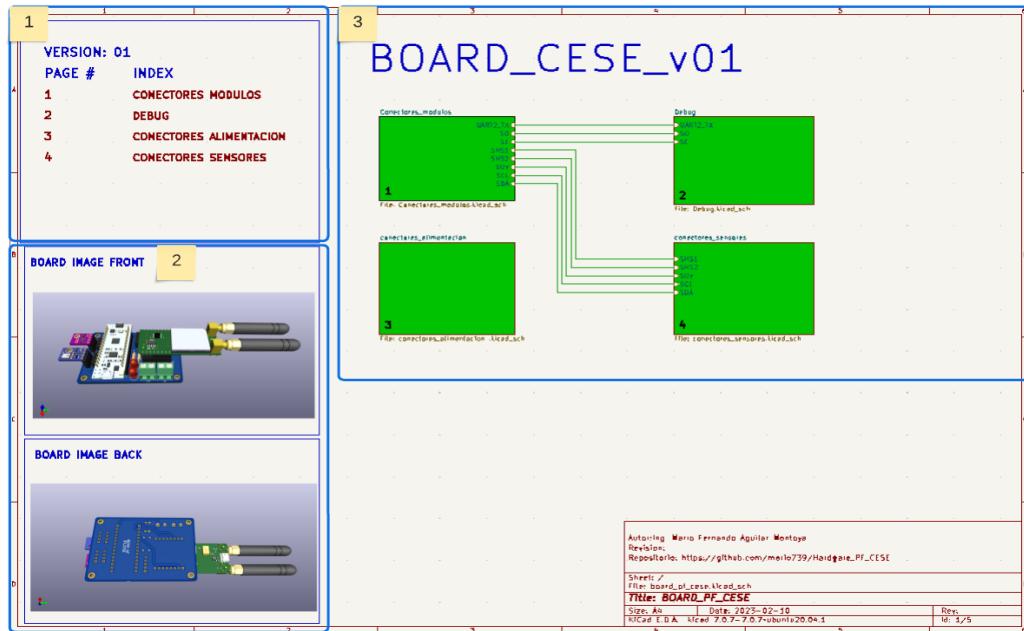


FIGURA 3.10. Esquemático página raíz.

En figura 3.11 se muestran los conectores de los módulos más importantes: el módulo de comunicación BG96, módulo NUCLEO-L432KC y la conexión entre ellos.

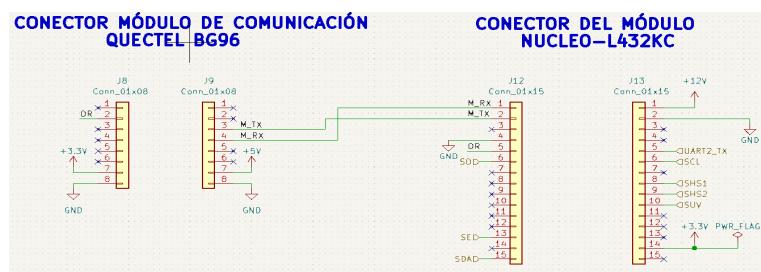


FIGURA 3.11. Conector módulo BG96 y NUCLEO-L432KC.

Se colocaron dos leds para debug, uno para señalizar si se logró conectar al servidor MQTT y el otro led para señalizar si el sistema entra en un estado de error. Se colocó un conector para una puerto serial por donde el módulo manda la secuencias de comandos que manda y recibe del módulo de comunicación. En la figura 3.12 se puede ver el circuito.

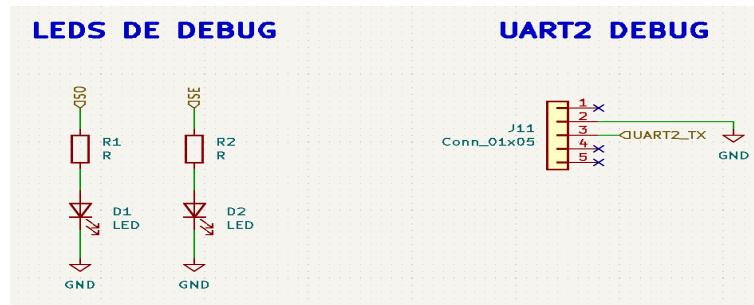


FIGURA 3.12. Esquemático interfaz de debug.

En la figura 3.13 se pueden ver los conectores que se colocaron a la placa para conectar los sensores del nodo: sensor de humedad 1, sensor de humedad 2, sensor de luz UV y el sensor de humedad y temperatura ambiente AHT-10.

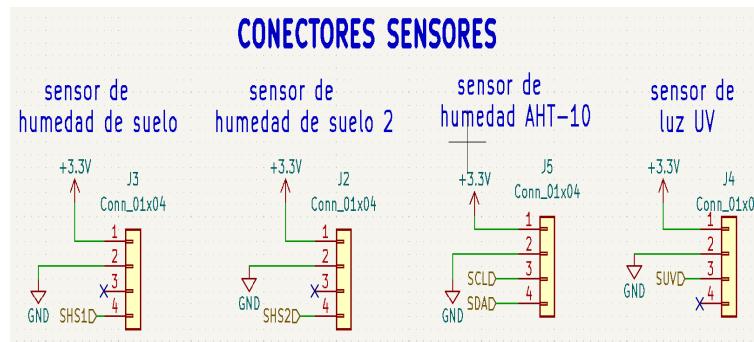


FIGURA 3.13. Esquemático conectores sensores.

Se colocaron dos colectores de alimentación como se muestra en la figura 3.14, el conector de 12V para alimentar al módulo del microcontrolador y el otro conector para alimentar al módulo de comunicación.

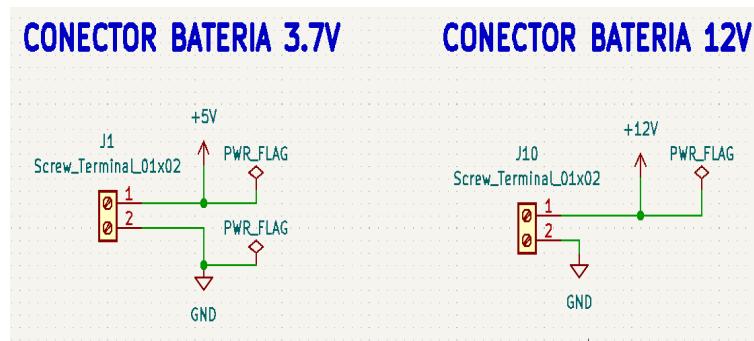


FIGURA 3.14. Esquemático conectores alimentación.

3.5.2. PCB del hardware

La figura 3.15 muestra el circuito impreso diseñado para el proyecto.

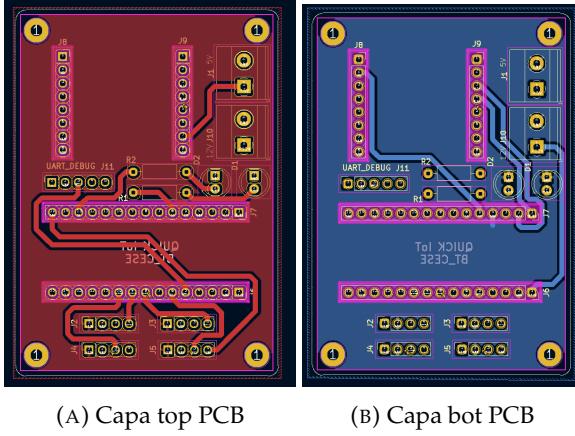


FIGURA 3.15. PCB del proyecto.

En la figura 3.16 se muestra el diseño de la tarjeta del circuito impreso en 3D.

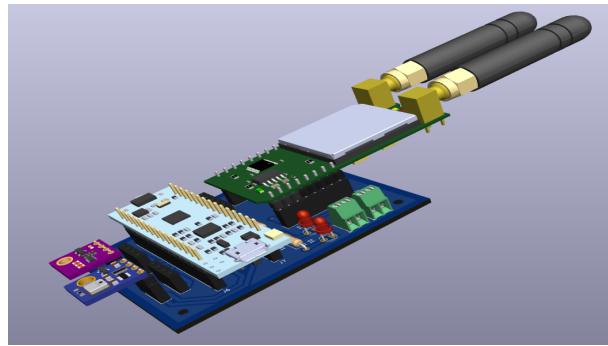


FIGURA 3.16. Modelo 3D de la tarjeta.

3.5.3. Fabricación circuito impreso

Una vez completado y validado el diseño, se generaron los archivos de fabricación y se mandaron a la empresa JLCPCB para su producción. La figura 3.17 muestra la tarjeta ya ensamblada con los módulos.

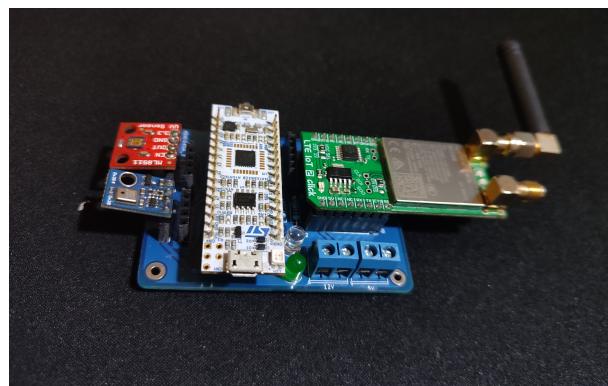


FIGURA 3.17. PCB ensamblado.

3.6. Paneles de visualización

La herramienta de visualización de ThingsBoard es muy versátil para el armado de paneles de visualización escalables y altamente configurables.

Se armó un panel de visualización principal que muestra los nodos sensores monitoreados por el sistema y un panel secundario que muestra las variables monitoreadas por cada nodo sensor a través gráficas, tablas, etc.

3.6.1. Panel principal

En la figura 3.18 se aprecia el panel principal de la interfaz gráfica. A continuación se describirán cada zona del panel principal. El panel está dividido en las siguientes zonas:

- Zona 1: listado de todos los nodos sensores implementados y activos, haciendo click en el sensor se navega al panel de visualización secundario.
- Zona 2: gráficas que muestra los cambios que van teniendo los valores de las variables medidas por los sensores con respecto al tiempo.
- Zona 3: mapa con la ubicación de los nodos sensores implementados.

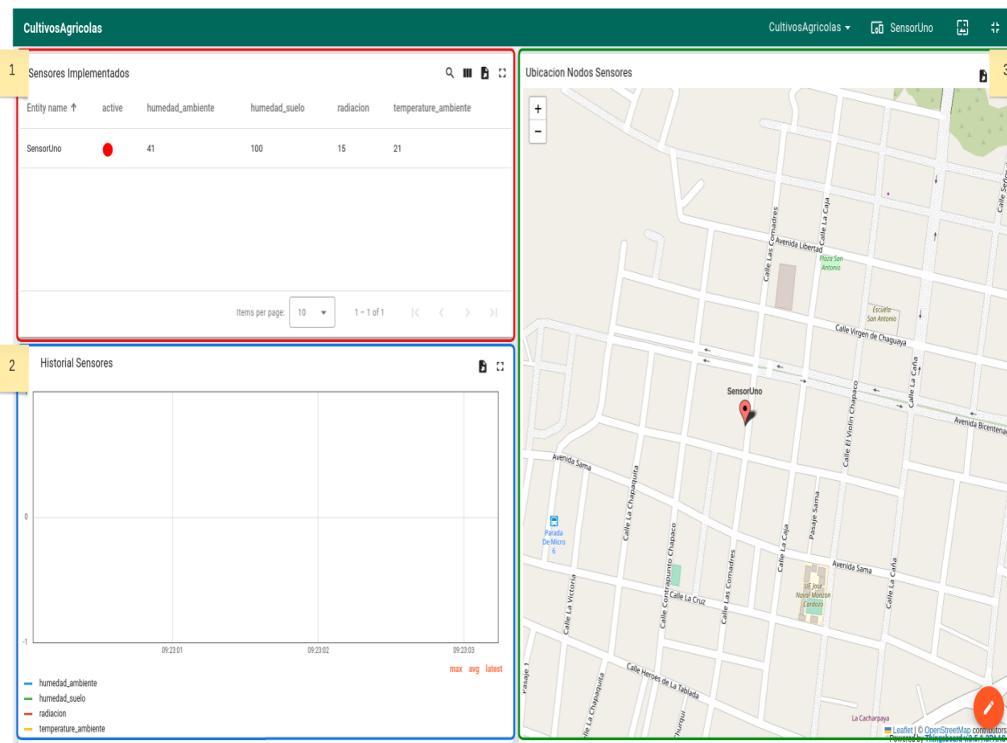


FIGURA 3.18. Panel principal de la interfaz gráfica.

3.6.2. Panel nodo sensor

Para tener un mayor detalle de todos los parámetros de monitoreo de cada nodo sensor se creó un panel secundario que se muestra en la figura 3.19.

El panel está dividido en las siguientes zonas:

- Zona 1: gráficas que muestran los cambios que van teniendo los valores de las variables medidas por los sensores con respecto al tiempo.
- Zona 2: widgets que muestran el último valor obtenido por el nodo sensor de cada variable monitoreada.
- Zona 3: tabla que muestra las alarmas que se activaron.
- Zona 4: se tiene un mapa que muestra la ubicación del nodo sensor implementado.

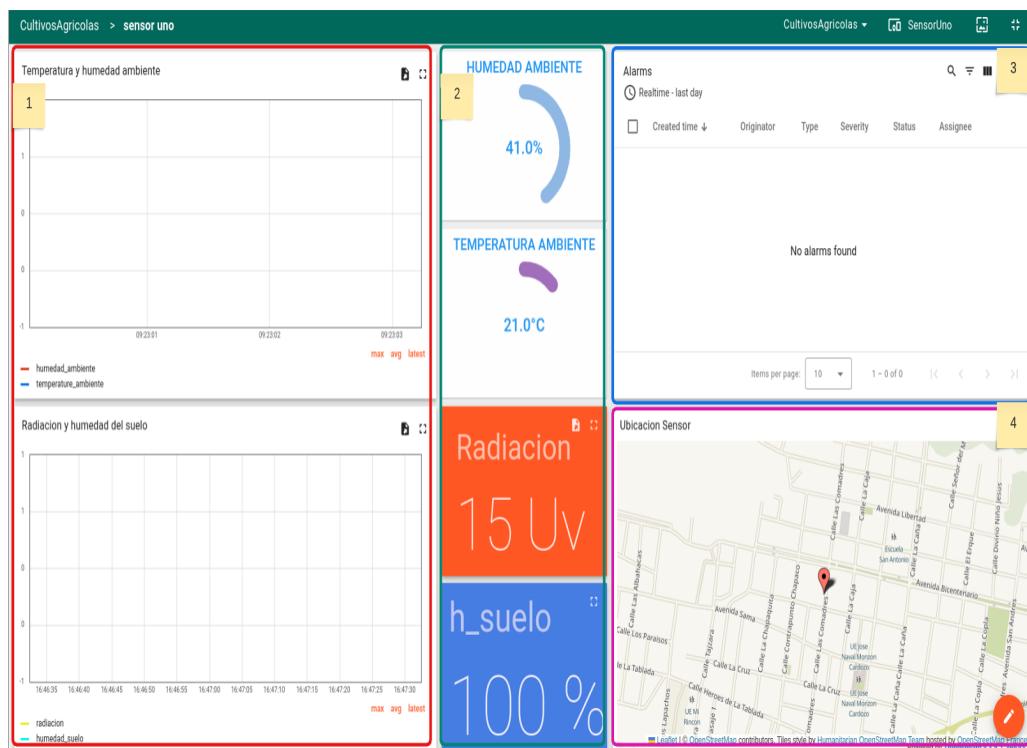


FIGURA 3.19. Panel nodo sensor.

Capítulo 4

Ensayos y resultados

En este capítulo se explican las pruebas realizadas al hardware, firmware , controladores y a la plataforma IoT a lo largo del trabajo.

4.1. Pruebas unitarias drivers

Para el desarrollo de los drivers del módulo BG96 y el sensor AHT10 se utilizó la metodología de desarrollo TDD. Esto implica que se escribieron pruebas unitarias para los drivers utilizando cedding herramienta para el desarrollo de pruebas automáticas. En el código 4.1 y 4.2 se pueden ver algunas pruebas unitarias para los drivers desarrollados.

```

1 //Prueba de funcion para obtener el estado del sensor AHT10
2 void test_estado_del_sensor(void)
3 {
4     uint8_t buffer[1]={0};
5     uint8_t data=0;
6     read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,buffer,1,
7         AHT10_OK);
8     read_I2C_STM32L432_port_ReturnThruPtr_buffer(&data);
9     TEST_ASSERT_EQUAL(SENSOR_IDLE,aht10_get_status(&aht10config));
10
11    data=255;
12    read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,buffer,1,
13        AHT10_OK);
14    read_I2C_STM32L432_port_ReturnThruPtr_buffer(&data);
15    TEST_ASSERT_EQUAL(SENSOR_BUSY,aht10_get_status(&aht10config));
16
17 }
18
19 //Prueba para la funcion para obtener el valor de la humedad
20 void test_obtener_humedad(void)
21 {
22     uint8_t bufferRead[6]={0};
23     uint8_t humedad=0;
24     uint8_t cmd[3] = {AHT10_CMD_TRIGGER_MEASUREMENT,AHT10_CMD_DATO_0,
25         AHT10_CMD_DATO_1};
26     uint8_t buffer[1]={0};
27     write_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,cmd,3,
28         AHT10_OK);
29     delay_STM32L432_port_Expect(AHT10_DELAY_LAUNCH_MEASUREMENT);
30     read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,buffer,1,
31         AHT10_OK);

```

```

29 write_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,cmd,3 ,
   AHT10_OK) ;
30 delay_STM32L432_port_Expect(AHT10_DELAY_LAUNCH_MEASUREMENT) ;
31 read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,bufferRead
   ,6,AHT10_OK) ;
32 TEST_ASSERT_EQUAL(AHT10_OK,aht10_get_humidity(&aht10config,&humedad)) ;
33 TEST_ASSERT_EQUAL(0,humedad) ;
34
35 write_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,cmd,3 ,
   AHT10_ERROR) ;
36 TEST_ASSERT_EQUAL(AHT10_ERROR,aht10_get_humidity(&aht10config,&humedad
   )) ;
37
38 write_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,cmd,3 ,
   AHT10_OK) ;
39 delay_STM32L432_port_Expect(AHT10_DELAY_LAUNCH_MEASUREMENT) ;
40 read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,buffer ,1 ,
   AHT10_OK) ;
41 write_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,cmd,3 ,
   AHT10_OK) ;
42 delay_STM32L432_port_Expect(AHT10_DELAY_LAUNCH_MEASUREMENT) ;
43 read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE,bufferRead
   ,6,AHT10_ERROR) ;
44 TEST_ASSERT_EQUAL(AHT10_ERROR,aht10_get_humidity(&aht10config,&humedad
   )) ;
45 }

```

CÓDIGO 4.1. Tests del driver sensor AHT10.

```

1 //Prueba para la funcion para obtener el estado en el que se encuentra
   el modulo bg96
2 void test_get_status_modem(void)
3 {
4     char buffer_resp[20]={0};
5     send_data_ExpectAndReturn(CMD_BG96_STATUS_MODEM,RS_BG96_OK,buffer_resp
   ,1000,FT_BG96_OK) ;
6     TEST_ASSERT_EQUAL(FT_BG96_OK,get_status_modem(&config_module)) ;
7
8     send_data_ExpectAndReturn(CMD_BG96_STATUS_MODEM,RS_BG96_OK,buffer_resp
   ,1000,FT_BG96_ERROR) ;
9     TEST_ASSERT_EQUAL(FT_BG96_ERROR,get_status_modem(&config_module)) ;
10 }

11 //Prueba de la funcion para mandar sms
12 void test_send_sms_bg96(void)
13 {
14     char buffer_resp[30]={0};
15     send_data_ExpectAndReturn("AT+CMGS=\\"72950576\\r",RS_BG96_SIGNAL,
   buffer_resp,12000,FT_BG96_OK) ;
16     send_data_ExpectAndReturn("HOLA\x1a\r",RS_BG96_OK,buffer_resp,12000,
   FT_BG96_OK) ;
17     TEST_ASSERT_EQUAL(FT_BG96_OK,send_sms_bg96(&config_module,"72950576",
   "HOLA")) ;
18
19     send_data_ExpectAndReturn("AT+CMGS=\\"72950576\\r",RS_BG96_SIGNAL,
   buffer_resp,12000,FT_BG96_ERROR) ;
20     TEST_ASSERT_EQUAL(FT_BG96_ERROR,send_sms_bg96(&config_module,"72950576",
   "HOLA")) ;
21
22     send_data_ExpectAndReturn("AT+CMGS=\\"72950576\\r",RS_BG96_SIGNAL,
   buffer_resp,12000,FT_BG96_OK) ;
23     send_data_ExpectAndReturn("HOLA\x1a\r",RS_BG96_OK,buffer_resp,12000,
   FT_BG96_ERROR) ;
24

```

```

25 TEST_ASSERT_EQUAL(FT_BG96_ERROR, send_sms_bg96(&config_module , "72950576
26     " , "HOLA" ));
27 }
28 //Prueba de la funcion para publicar mensajes al broker mqtt
29 void test_publish_message(void)
30 {
31     char buffer_resp[30]={0};
32     char topic[19]="/v1.6/devices/demo";
33     char data[25] = "\\" demo \":10,\\" humedad \":60\}";
34     send_data_ExpectAndReturn("AT+QMTPUB=0,0,0,0,"/v1.6/devices/demo\"\r",
35         ,RS_BG96_SIGNAL,buffer_resp,3000,FT_BG96_OK);
36     send_data_ExpectAndReturn("\\" demo \":10,\\" humedad \":60\}\x1a\r",
37         RS_BG96_CERO,buffer_resp,15000,FT_BG96_OK);
38     TEST_ASSERT_EQUAL(FT_BG96_OK,publish_message(&config_module ,topic ,data
39     ));
40
41     send_data_ExpectAndReturn("AT+QMTPUB=0,0,0,0,"/v1.6/devices/demo\"\r",
42         ,RS_BG96_SIGNAL,buffer_resp,3000,FT_BG96_ERROR);
43     TEST_ASSERT_EQUAL(FT_BG96_ERROR,publish_message(&config_module ,topic ,
44     data));
44 }

```

CÓDIGO 4.2. Tests del driver modulo bg96.

Una forma cuantitativa de evaluar estas pruebas son los informes de cobertura generados por cedding.

En la figura 4.1 se puede observar el informe de cobertura del driver aht10, donde se puede apreciar que las pruebas ejecutan el 100 % de las líneas de código escritas y explora el 100 % de las combinaciones en los saltos de condicionales.

En la figura 4.2 también se observa el informe de cobertura del driver de bg96, con 98.4 % de líneas ejecutadas y explora más del 98.3 % de combinaciones posibles.

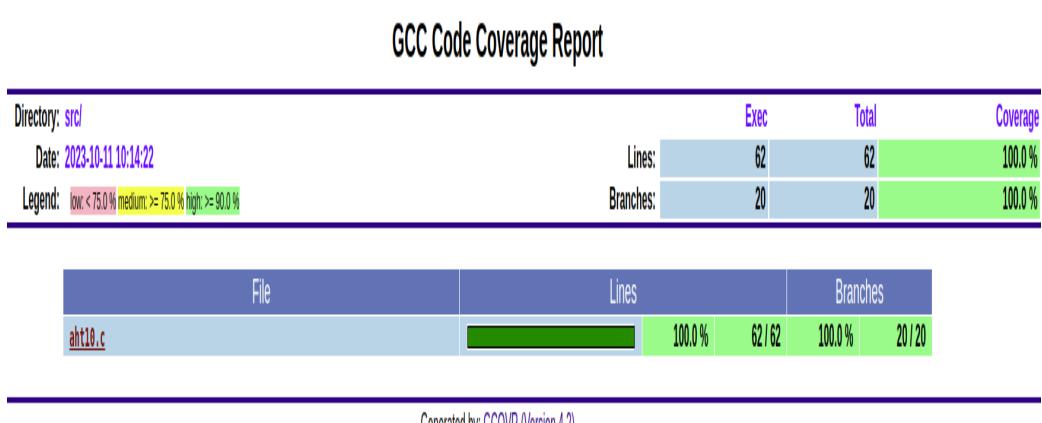


FIGURA 4.1. Informe de cobertura driver aht10.

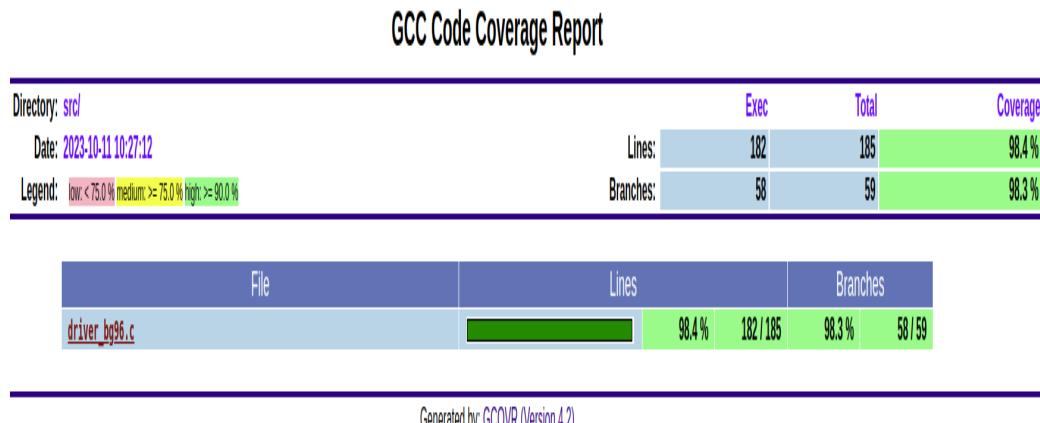


FIGURA 4.2. Informe de cobertura driver BG96.

4.2. Pruebas de la plataforma IoT

4.2.1. Pruebas de inyección de mensajes

El objetivo de las pruebas de inyección de mensajes a la plataforma IoT es evaluar la llegada de los mensajes por protocolo MQTT al servidor. Para realizar el envío de datos a broker MQTT, se utiliza el cliente MQTT de mosquitto ejecutando el comando que se muestra en la figura 4.3.

```
mario738@DESKTOP-L5GN2NG:~$ mosquitto_pub -d -q 1 -h "mqtt.thingsboard.cloud" -t "v1/devices/me/telemetry" -u "ZDqUF9f4VEDj6THx6cAd" -m '{"humedad_ambiente":10,"temperatura_ambiente":40,"humedad_suelo":80,"radiacion":30,"latitud":-21.532614,"longitude":-64.762743}'
Client mosq-DQuIytXMuQUN639LsR sending CONNECT
Client mosq-DQuIytXMuQUN639LsR received CONNACK (0)
Client mosq-DQuIytXMuQUN639LsR sending PUBLISH (d0, q0, r0, m1, 'v1/devices/me/telemetry', ... (116 bytes))
Client mosq-DQuIytXMuQUN639LsR received PUBACK (Mid: 1, RC:0)
Client mosq-DQuIytXMuQUN639LsR sending DISCONNECT
```

FIGURA 4.3. Envio de datos por el cliente MQTT de mosquitto.

Donde:

- -h dirección del broker
- -t tópico
- -u token
- -m mensaje en formato json

Al ejecutar el comando podemos ver que el cliente mqtt primeramente se conecta al broker mqtt luego publica el mensaje en el topico escogido y finalmente se desconecta del servidor.

Para comprobar la llegada de los valores al broker de ThingsBoard tenemos que ir a la sección dispositivos, seleccionar el dispositivo al que se le envió los datos y entrar a la pestaña de última telemetría, en la figura 4.4 podemos ver que los datos llegaron correctamente.

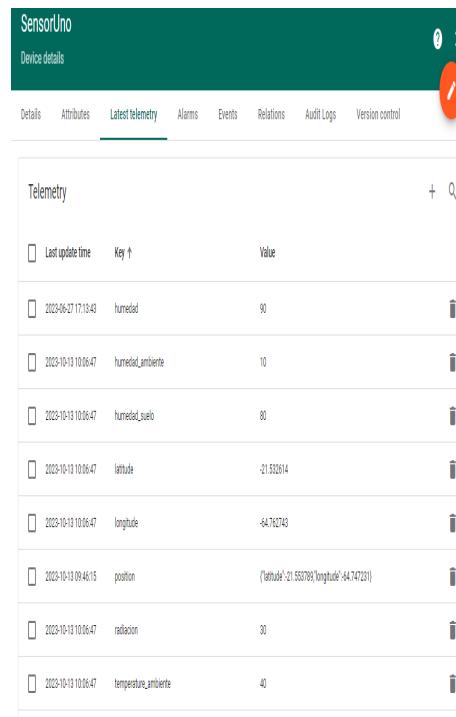


FIGURA 4.4. Recepción de datos en el broker MQTT.

4.2.2. Prueba de la tabla de alarmas en thingsboard

La plataforma permite configurar alarmas con respecto a las variables monitoreadas, en el panel de visualización de cada sensor se tiene una tabla de alarmas que muestran las notificaciones de las alarmas que se activaron. En la figura 4.5 podemos ver la notificación de una alarma cuando la temperatura ambiente del sistema sobrepasa los 43 grados centígrados.

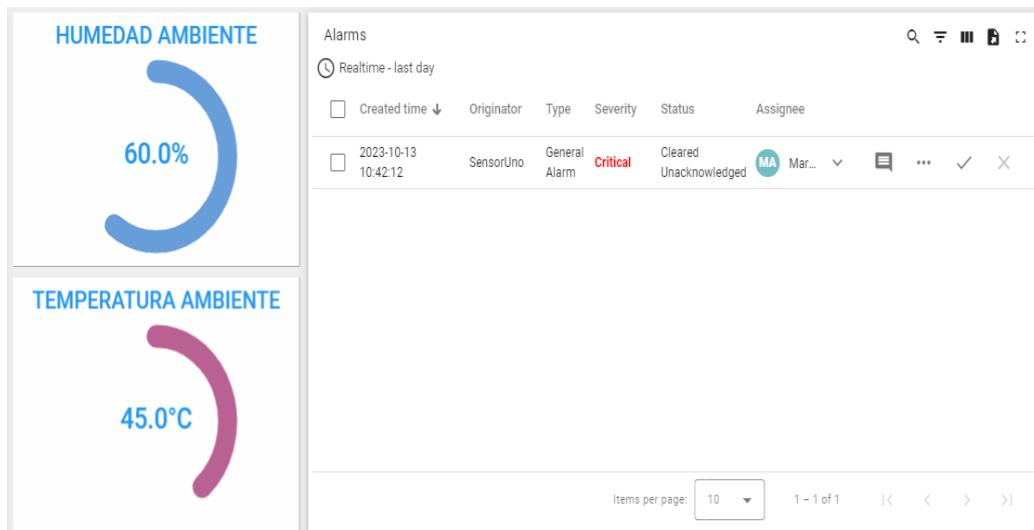


FIGURA 4.5. Tabla de alarmas activas.

4.2.3. Prueba del widget de mapa

En los paneles de visualización tenemos mapas con la ubicación del lugar donde se implementó el dispositivo. En la figura 4.6 podemos ver la ubicación del sensor implementado para el proyecto.

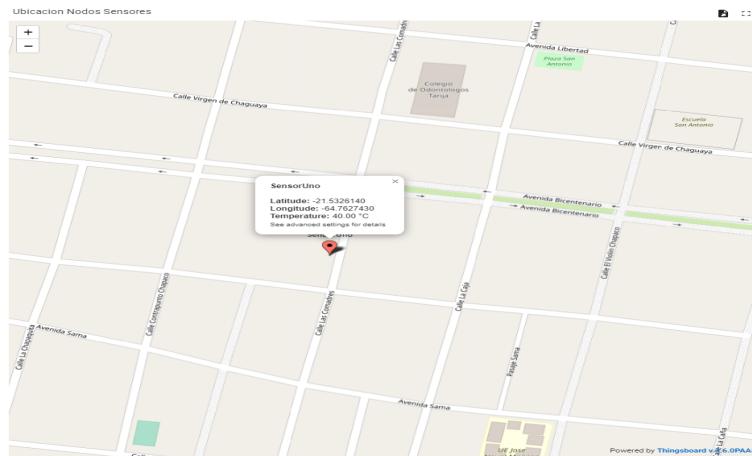


FIGURA 4.6. Ubicación del nodo sensor implementado.

4.2.4. Prueba de persistencia de datos

Para realizar la prueba de persistencia de datos se configuró en el panel de visualización, las gráficas con un entorno de tiempo más amplio. En las gráficas se estableció un rango de tiempo de 7 días. El resultado se muestra en la figura 4.7.

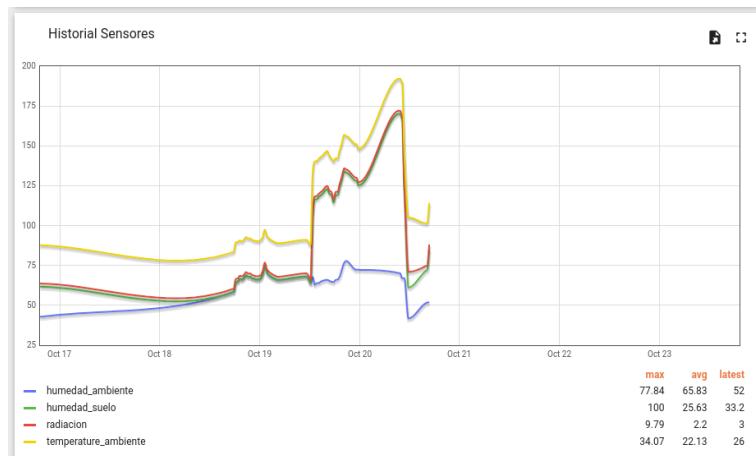


FIGURA 4.7. Persistencia de datos.

4.3. Pruebas De Hardware

4.3.1. Prueba comunicación sensor de humedad AHT10

Para probar el correcto funcionamiento del sensor AHT10 y la correcta comunicación con el microcontrolador se comprobó la trama I2C con un analizador lógico, en la figura 4.8 podemos ver la trama capturada cuando queremos escribir en un registro del sensor y en la figura 4.9 tenemos la trama cuando leemos los registros del sensor donde se obtiene 6 bytes en los que se encuentra la información de la humedad y la temperatura.

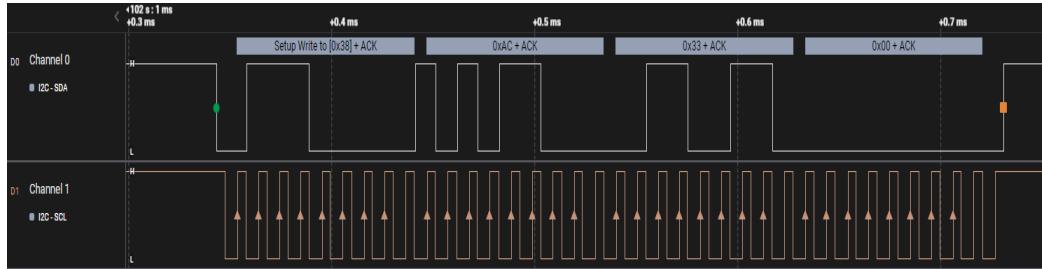


FIGURA 4.8. Trama de escritura al sensor AHT10.

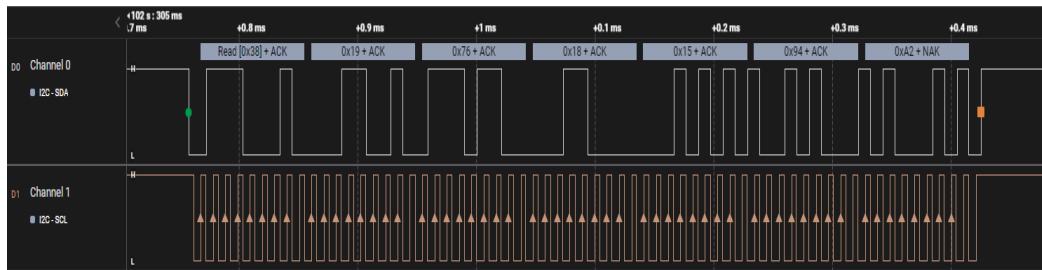


FIGURA 4.9. Trama de lectura del sensor AHT10.

4.3.2. Pruebas comunicación modulo BG96

Para probar la comunicación del microcontrolador con el módulo bg96 se utilizó un analizador lógico que nos permite ver los comandos que envía el microcontrolador y la respuesta del módulo a estos comandos por el puerto UART. En la figura 4.10 vemos dos canales del analizador lógico el canal 2 muestra un comando mandado por el microcontrolador al módulo de comunicación y en el canal 3 la respuesta del módulo al comando enviado.

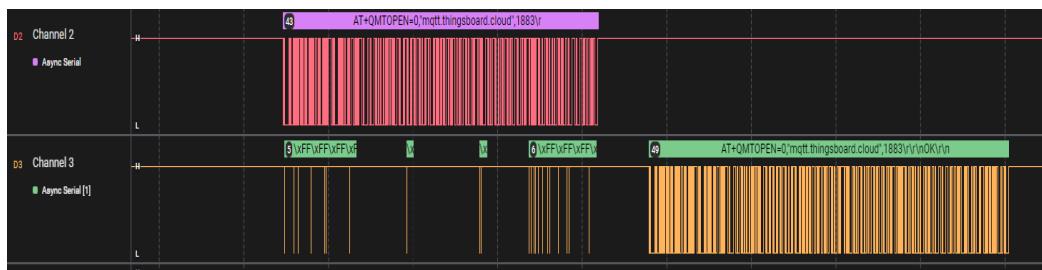


FIGURA 4.10. Envío y recepción de comandos por puerto UART.

4.3.3. Pruebas de alimentación

Se realizaron pruebas a las fuentes de alimentación del sistema, se midió el voltaje que recibe el microcontrolador que debería ser 12V como se muestra en la figura 4.11b, tambien se probó la alimentación al módulo de comunicación que debería ser 3.7V como se muestra en la figura 4.11a.

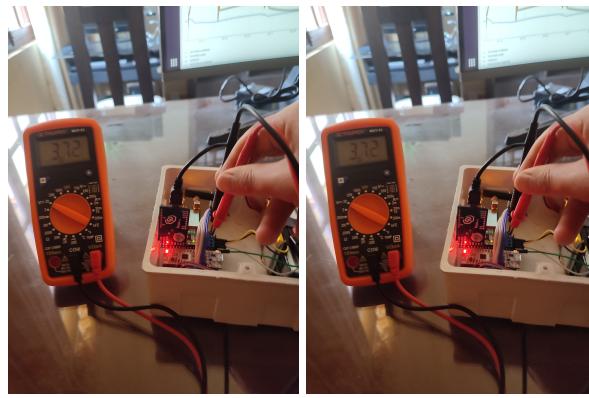


FIGURA 4.11. Medición de voltaje.

4.3.4. Prueba microcontrolador en bajo consumo

Al ser un dispositivo que funciona a batería lo que busca el firmware es consumir lo menos posible, por lo tanto el sistema entra en bajo consumo en los momentos en que el microcontrolador no realiza ninguna función. En la figura 4.12a podemos ver el consumo del sistema normalmente y en la figura 4.12bel consumo bajo consumo.

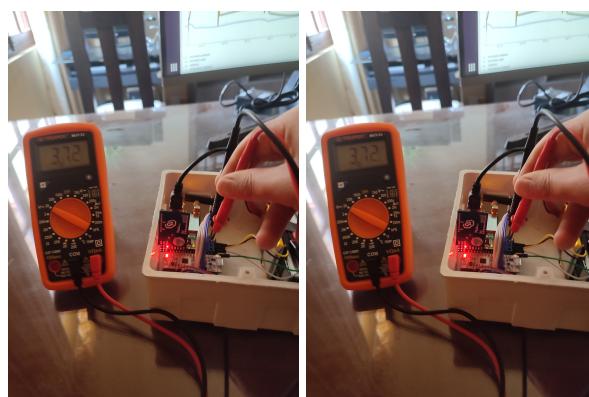


FIGURA 4.12. Modos de consumo.

4.4. Pruebas Funcionales del sistema

Para realizar las pruebas funcionales de todo el sistema se implementó el prototipo en un cultivo de tomate como se muestra en la figura 4.13.



FIGURA 4.13. Implementación del prototipo.

4.4.1. Prueba de lectura de sensores

Se realizó la lectura de todos los sensores y se mando los datos obtenidos al servidor mqtt .La figura 4.14 muestra la lectura de los datos adquiridos.

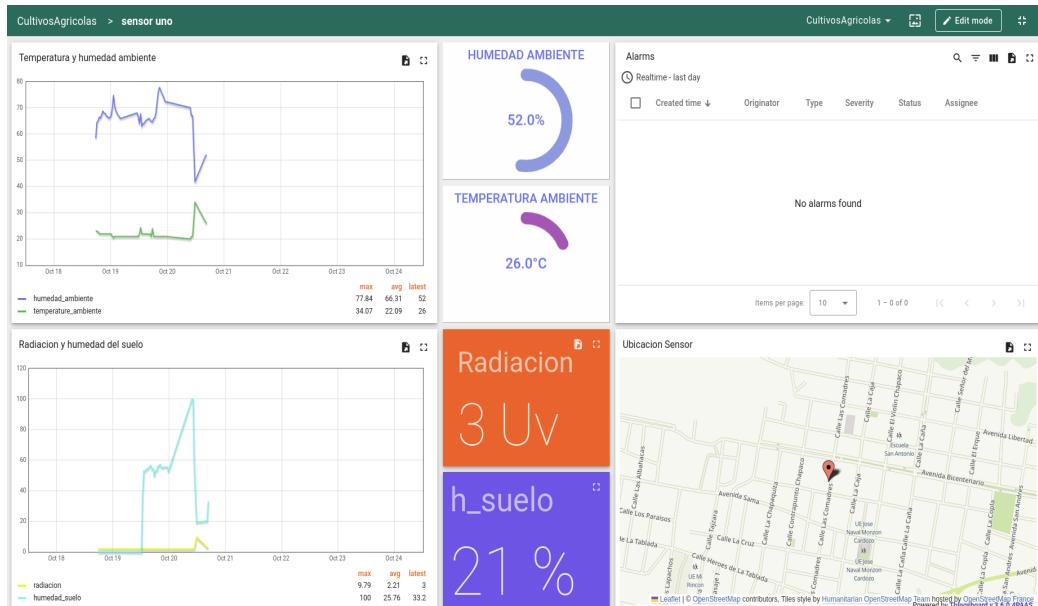


FIGURA 4.14. Lectura de todos los sensores.

Las figuras muestran las pruebas que se realizaron a las lecturas del sensor de humedad se suelo en diferentes casos, cuando el suelo está húmedo la lectura del sensor es 45 por ciento y cuando el suelo está seco la lectura del sensor es 10.

La lectura del sensor de radiación es de 3 que significa que la radiación es moderada la lectura en horas del mediodía donde el sol da con más fuerza en el lugar de la implementación.

4.4.2. Prueba de envío de datos al broker mqtt

Una de las tareas más importante del firmware es la del manejo del servidor, para verificar el correcto funcionamiento de esta tarea tenemos que ver la secuencia de comandos que son mandados del microcontrolador al módulo de comunicación por el puerto UART. Para realizar esta prueba se conectó un convertidor UART a USB al conector de debug que tiene nuestro dispositivo, con este convertidor podemos ver todos los comandos que el microcontrolador manda al módulo de comunicación. En la figura 4.15 podemos observar toda la secuencia de comandos que realiza el firmware para realizar el envío de datos a la plataforma IoT, como vemos primeramente se verifica si el módulo está activo, luego se configura el apn de la red, luego se conecta al broker mqtt , se publican los datos al servidor y finalmente el realizar el proceso de desconexión.

```

AT
AT          Comando estado
OK

AT+QICSGP=1,1,"internet.tigo.bol","","","",1
AT+QICSGP=1,1,"internet.tigo.bol","","","",1

OK          Configurar red y activacion
AT+QIACT=1
AT+QIACT=1

OK
AT+QMTOPEN=0,"mqtt.thingsboard.cloud",1883
AT+QMTOPEN=0,"mqtt.thingsboard.cloud",1883

OK          Conexion broker
+QMTOPEN: 0,0
AT+QMTCNN=0,"123a56cb9","ZDqUF9f4VEDj6THx6cAd","ZDqUF9f4VEDj6THx6cAd"
AT+QMTCNN=0,"123a56cb9","ZDqUF9f4VEDj6THx6cAd","ZDqUF9f4VEDj6THx6cAd"

OK          MQTT
+QMTCNN: 0,0,0
AT+QMTPUB=0,0,0,0,"v1/devices/me/telemetry"
AT+QMTPUB=0,0,0,0,"v1/devices/me/telemetry"

> {"humedad_suelo":55,"humedad_ambiente":63,"radiacion":3,"temperatura_ambiente":22,"position":{"latitude":-21.553789,"longitude":-64.747231}}0
("humedad_suelo":55,"humedad_ambiente":63,"radiacion":3,"temperatura_ambiente":22,"position":{"latitude":-21.553789,"longitude":-64.747231})
OK          Publicar datos
+QMTPUB: 0,0,0
AT+QMTCLOSE=0
AT+QMTCLOSE=0

OK          Comandos de
AT+QMDISC=0
AT+QMDISC=0
desconexion

OK
AT+QIDEACT=1
AT+QIDEACT=1

OK
+QMTCLOSE:0

```

FIGURA 4.15. Comandos para enviar datos al broker mqtt.

4.4.3. Pruebas de alarmas

El objetivo de esta prueba es comprobar el buen funcionamiento de las alarmas del sistema. Cuando la humedad del suelo baja por debajo del rango permitido por el sistema, el firmware manda un sms al usuario con la alarma ocurrida. En la figura 4.16 podemos ver que la humedad bajo de 10 % y en la figura 4.17 se ve que se recibió el sms.

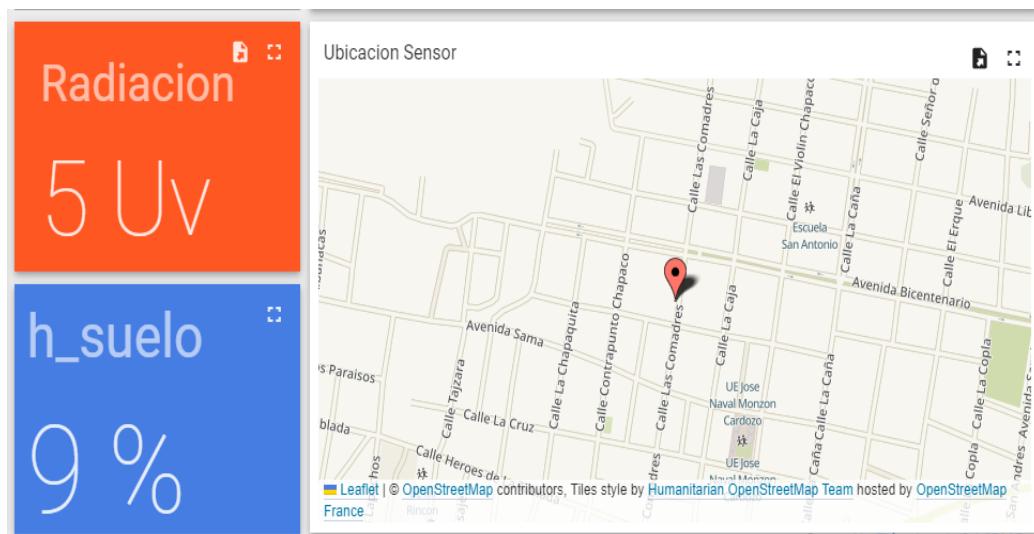


FIGURA 4.16. Humedad menor y activación de la alarma.

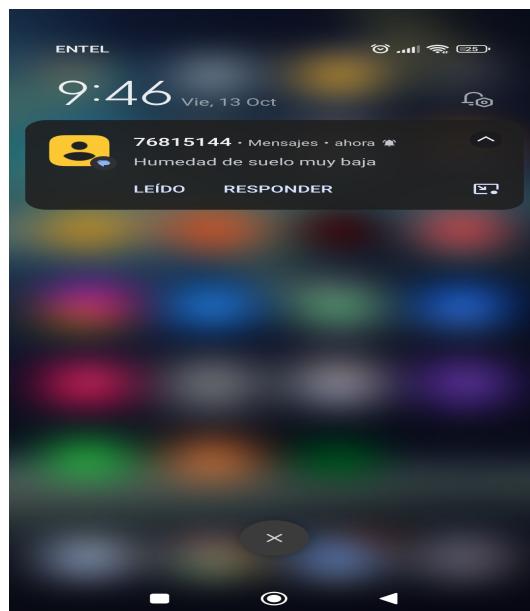


FIGURA 4.17. Recepción de sms por la alarma.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se pudo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

Bibliografía

- [1] Sara Oleiro Araujo y col. «Characterising the Agriculture 4.0». En: *Agronomy* 11.4, 2021, pág. 667.
- [2] Marco Centenaro y col. «Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios». En: *IEEE Wireless Communications* 23.5, 2016, págs. 60-67.
- [3] Ferrovial. *Internet de las cosas(IoT)*. Visitado: 2023-05-27. URL: <https://www.ferrovial.com/es/recursos/internet-de-las-cosas/>.
- [4] Libelium. *Smart Agriculture PRO,TECHNICAL GUIDE*. Visitado: 2023-05-27. URL: <https://development.libelium.com/agriculture-sensor-guide/>.
- [5] WiseConn. *Nodo RF-M1*. Visitado: 2023-05-27. URL: <https://www.wiseconn.cl/dropcontrol/hardware/rf-m1/>.
- [6] connectamericas. *Descripción de la empresa,Definición de UBIDOTS*. Visitado: 2023-05-27. URL: <https://connectamericas.com/es/company/ubidots>.
- [7] STMicroelectronics. *Especificacion de Producto,STM32 Nucleo-32 boards*. Ultima actualizacion 2019-06-05 V8.0. URL: https://www.st.com/resource/en/data_brief/nucleo-l432kc.pdf.
- [8] MikroElectronic. *Especificacion de Producto,LTE IOT 2 CLICK*. Visitado: 2023-05-27. URL: <https://www.mikroe.com/lte-iot-2-click>.
- [9] SSDIELECT ELECTRONICA SAS. *Especificacion de Producto,AHT10 SENSOR DE TEMPERATURA Y HUMEDAD I2C*. Visitado: 2023-05-27. URL: <https://ssdiselect.com/temperatura/3885-aht10.html>.
- [10] naylampmechatronics. *Especificacion de Producto,MÓDULO SENSOR DE LUZ ULTRAVIOLETA (UV) ML8511*. Visitado: 2023-05-27. URL: <https://ssdiselect.com/temperatura/3885-aht10.html>.
- [11] PANAMAHITEK. *Especificacion de Producto,Módulo HL-69: Un sensor de humedad de suelo*. Visitado: 2023-05-27. URL: <https://panamahitek.com/modulo-hl-69-un-sensor-de-humedad-de-suelo/>.
- [12] STMicroelectronics. *Descripción del producto,Integrated Development Environment for STM32*. Visitado: 2023-05-27. URL: <https://www.st.com/en/development-tools/stm32cubeide.html>.
- [13] FreeRTOS. *Acerca del Sistema Operativo,Descripción General*. Visitado: 2023-05-27. URL: <https://www.freertos.org/RTOS.html>.
- [14] CEDLING. *Descripción del producto,Descripción General*. Visitado: 2023-05-27. URL: <http://www.throwtheswitch.org/ceedling>.
- [15] rohde-schwarz. *Descripción del protocolo,Qué es UART*. Visitado: 2023-05-27. URL: https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart_254524.html.
- [16] sparkfun. *Definicion del Protocolo*. Visitado: 2023-05-27. URL: <https://learn.sparkfun.com/tutorials/i2c/all>.
- [17] MQTT. *Descripción del protocolo,Introducción a MQTT*. Visitado: 2023-05-27. URL: <https://mqtt.org/>.

- [18] ThingsBoard. *Descripción de la plataforma IoT, Definición de ThingsBoard.* Visitado: 2023-05-27. URL: <https://thingsboard.io/>.