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. CEEDLING . . . 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 . . . 15
  - 3.3.4. Tarea manejador del servidor . . . 16

v

Índice general

Resumen I

1. Introducción general 1
- 1.1. Internet de las cosas 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. CEEDLING . . . 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 . . . 15
  - 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 . . . 21
3.5.1. Esquemático . . . 21
3.5.2. PCB del hardware . . . 23
3.5.3. Fabricación circuito impreso . . . 23
3.6. Paneles de visualización . . . 24
3.6.1. Panel principal . . . 24
3.6.2. Panel nodo sensor . . . 25

4. Conclusiones **27**
4.1. Conclusiones generales . . . 27
4.2. Próximos pasos . . . 27

Bibliografía **29**

VI

3.4. Controladores implementados . . . 18
3.4.1. Controlador para el sensor AHT10 . . . 18
3.4.2. Controlador para el módulo de comunicación BG96 . . . 19
3.5. Desarrollo del hardware . . . 21
3.5.1. Esquemático . . . 21
3.5.2. PCB del hardware . . . 23
3.5.3. Fabricación del circuito impreso . . . 23
3.6. Paneles de visualización . . . 24
3.6.1. Panel principal . . . 24
3.6.2. Panel nodo sensor . . . 25

4. Ensayos y resultados **27**
4.1. Pruebas unitarias . . . 27
4.2. Pruebas de la plataforma IoT . . . 29
4.2.1. Prueba de inyección de mensajes . . . 29
4.2.2. Prueba de la tabla de alarmas del panel visualización . . . 30
4.2.3. Prueba del widget de mapa . . . 30
4.2.4. Prueba de persistencia de datos . . . 30
4.3. Pruebas de hardware . . . 31
4.3.1. Prueba de comunicación entre el sensor AHT10 y el microcontrolador . . . 31
4.3.2. Prueba de comunicación entre el módulo BG96 y el microcontrolador . . . 32
4.4. Pruebas funcionales del sistema . . . 32
4.4.1. Pruebas de lectura de los sensores . . . 32
Prueba uno . . . 32
Prueba dos . . . 33
4.4.2. Prueba del envío de datos al broker MQTT . . . 34
4.4.3. Prueba de envio de alarmas por SMS . . . 35

5. Conclusiones **37**
5.1. Conclusiones generales . . . 37
5.2. Próximos pasos . . . 37

Bibliografía **39**

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 conexion server MQTT. 17
3.8. Máquina de estados down servidor. 17
3.9. Máquina de estados up servidor. 18
3.10. Esquemático página raíz. 21
3.11. Conector módulo BG96 y NUCLEO-L432KC. 22
3.12. Esquemático interfaz de debug. 22
3.13. Esquemático conectores sensores. 22
3.14. Esquemático conectores alimentación. 23
3.15. PCB del proyecto. 23
3.16. Modelo 3D de la tarjeta. 23
3.17. PCB ensamblado. 24
3.18. Panel principal de la interfaz gráfica. 24
3.19. Panel nodo sensor. 25

VII

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 conexion server MQTT. 17
3.8. Máquina de estados down servidor. 17
3.9. Máquina de estados up servidor. 18
3.10. Esquemático página raíz. 21
3.11. Conector módulo BG96 y NUCLEO-L432KC. 22
3.12. Esquemático interfaz de debug. 22
3.13. Esquemático conectores sensores. 22
3.14. Esquemático conectores de alimentación. 23
3.15. PCB del trabajo. 23
3.16. Modelo 3D de la tarjeta. 23
3.17. PCB ensamblado. 24
3.18. Panel principal de la interfaz gráfica. 24
3.19. Panel nodo sensor. 25

4.1. Informe de cobertura driver aht10. 28
4.2. Informe de cobertura driver BG96. 28
4.3. Envio de datos por el cliente MQTT de mosquitto. 29
4.4. Recepción de datos en el broker MQTT. 29
4.5. Tabla de alarmas activas. 30
4.6. Ubicación del nodo sensor implementado. 30
4.7. Persistencia de datos. 31
4.8. Trama de escritura al sensor AHT10. 31
4.9. Trama de lectura del sensor AHT10. 31
4.10. Envío y recepción de comandos por puerto UART. 32

4.11. Implementación del prototipo. 32
4.12. Condiciones ambientales prueba 1. 33
4.13. Panel de visualización con las lecturas obtenidas en la prueba 1. . . 33
4.14. Condiciones ambientales prueba 2. 34
4.15. Panel de visualización con las lecturas obtenidas en la prueba 2. . . 34
4.16. Comandos para enviar datos al broker MQTT. 35
4.17. Humedad del suelo menor al 10 %. 35
4.18. Recepción del SMS con el mensaje de alarma. 35

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.

1

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. Internet de las cosas 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.

- 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 desarrollo 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.

3.2. Arquitectura de firmware 13

- 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, que se describen a continuación:

- 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 tarea loop comienza iniciando un timer que se encarga controlar el tiempo de repetición del ciclo de la tarea. Luego la tarea se bloquea. Cuando se termina el tiempo del timer, se ejecuta el handler de la interrupción desbloqueando la tarea loop. La tarea envía un evento a cola de adquisición de datos para realizar la lectura de los sensores y un evento a cola que maneja la conexión para levantar el servidor. Posteriormente la tarea comprueba si se logró levantar una conexión. Si la conexión existe la tarea manda un evento por la cola del servidor para que se envíen los datos al broker MQTT. Luego la tarea envía un evento a la cola de alarmas para mandar los sms de las alarmas activas del sistema. Finalmente después de monitorear las alarmas la tarea manda un evento a la cola de servidor

14 Capítulo 3. Diseño e implementación

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 tarea loop comienza iniciando un timer que se encarga controlar el tiempo de repetición del ciclo de la tarea. Luego la tarea se bloquea. Cuando se termina el tiempo del timer, se ejecuta el handler de la interrupción desbloqueando la tarea loop. La tarea envía un evento a la cola de adquisición de datos para realizar la lectura de los sensores y un evento a la cola que maneja la conexión para levantar el servidor. Posteriormente la tarea comprueba si se logró levantar una conexión. Si la conexión existe la tarea manda un evento por la cola del servidor para que se envíen los datos al broker MQTT. Luego la tarea envía un evento a la cola de alarmas para mandar los SMS(Mensaje de texto) de las alarmas activas del sistema. Finalmente, después de monitorear las alarmas la tarea manda un evento

para la desconexión. Al finalizar el ciclo de la tarea, inicia el timer nuevamente y manda al microcontrolador a modo de bajo consumo para ahorrar energía.

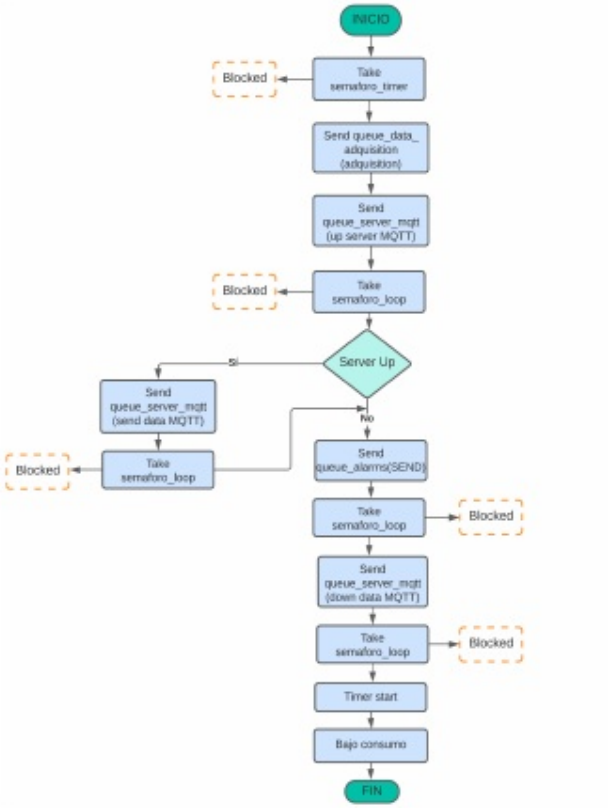


FIGURA 3.4. Diagrama de flujo de la tarea loop.

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.

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.

a la cola de servidor para la desconexión. Al finalizar el ciclo de la tarea, inicia el timer nuevamente y manda al microcontrolador a modo de bajo consumo para ahorrar energía.

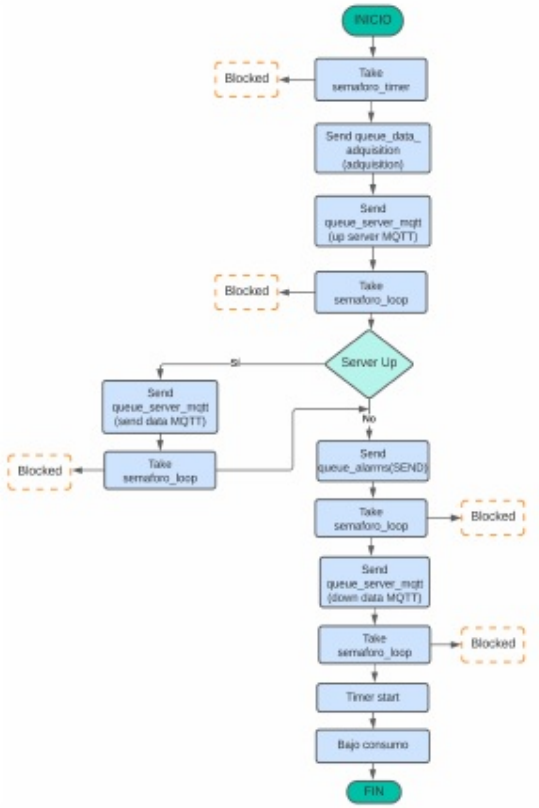


FIGURA 3.4. Diagrama de flujo de la tarea loop.

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.

3.3.3. Tarea manejador de alarmas

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

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

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.

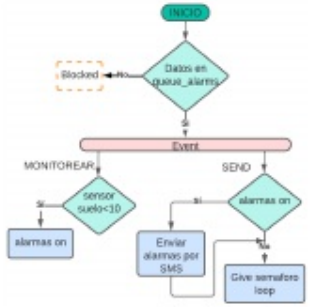


FIGURA 3.6. Diagrama de flujo de la tarea manejador de alarmas.

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.

16 Capítulo 3. Diseño e implementación

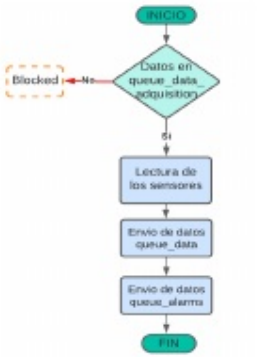


FIGURA 3.5. Diagrama de flujo tarea de adquisición de datos.

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 si el valor del sensor de humedad es menor a 10. Aumentando en uno la variable que cuenta las alarmas activas si esto fuera verdadero. 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.

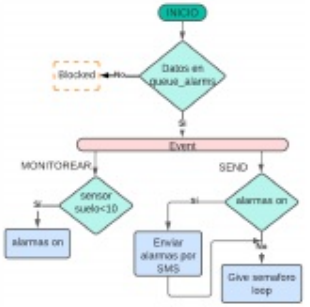


FIGURA 3.6. Diagrama de flujo de la tarea manejador de alarmas.

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 ingresa a la máquina de estados que se muestra en la figura 3.9, la que se encarga de establecer 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, arma la trama y publica los datos al broker MQTT.

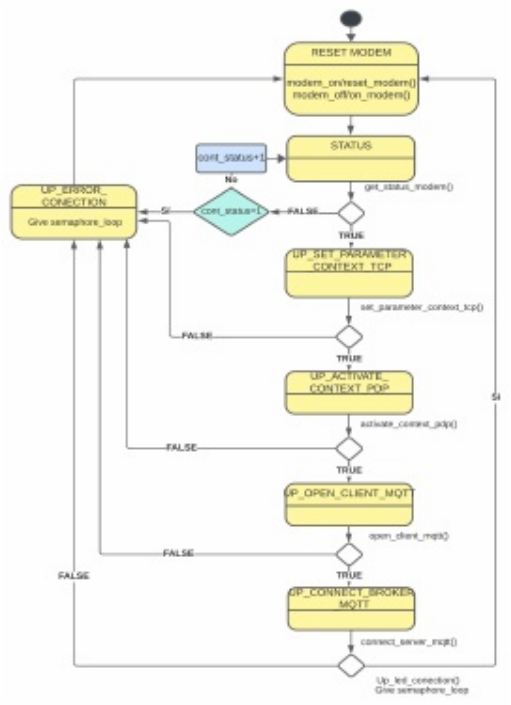


FIGURA 3.9. Máquina de estados up servidor.

3.4. Controladores implementados

Se implementaron dos controladores: para el modulo de comunicacion BG96 y para el sensor de humedad y temperatura ambiente AHT10.

3.4.1. Controlador sensor AHT10

Para la lectura de humedad y temperatura, se desarrolló en el driver las funciones de *aht10_get_humedity()* y *aht10_get_temperature()* que se muestran en el código 3.1. Las funciones comienzan iniciando una medición en el sensor con la función *aht10_launch_measurement()*, luego realizan la lectura de los registros del sensor, almacenando los valores de retorno en un buffer. Con los bytes obtenidos se realiza un corrimiento para obtener los bytes que contienen la información de humedad y temperatura. Finalmente se aplican las fórmulas de las líneas 1 y 2 del código 3.1 para convertir los bytes en valores significativos de las variables físicas.

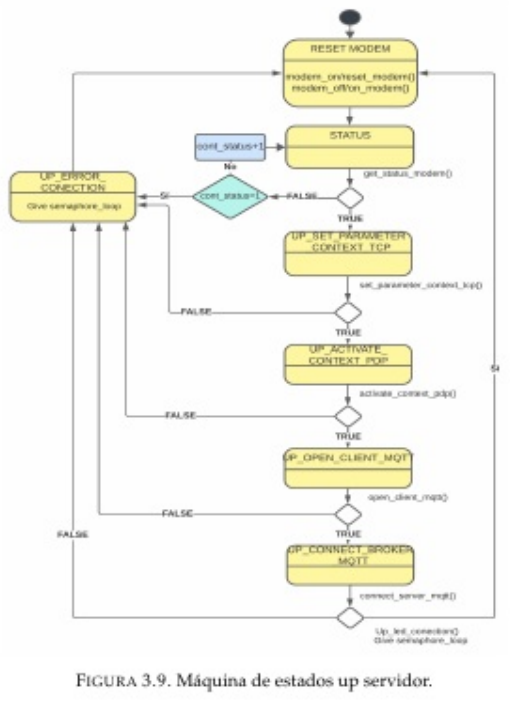


FIGURA 3.9. Máquina de estados up servidor.

3.4. Controladores implementados

Se implementaron dos controladores: para el módulo de comunicación BG96 y para el sensor de humedad y temperatura ambiente AHT10.

3.4.1. Controlador para el sensor AHT10

Para la lectura de humedad y temperatura, se desarrolló en el driver las funciones de *aht10_get_humedity()* y *aht10_get_temperature()* que se muestran en el código 3.1. Las funciones comienzan iniciando una medición en el sensor con la función *aht10_launch_measurement()*, luego realizan la lectura de los registros del sensor, almacenando los valores de retorno en un buffer. Con los bytes obtenidos se realiza un corrimiento para obtener los bytes que contienen la información de humedad y temperatura. Finalmente, se aplican las fórmulas de las líneas 1 y 2 del código 3.1 para convertir los bytes en valores significativos de las variables físicas.

```
1 #define TEMPERATURE(A)                    (int8_t) ((A *0.000191)-50)
2 #define HUMEDITY(A)                       (uint8_t) (A *0.000095)
3 //Funcion para obtener el valor de la humeda
4 aht10_status_fnc aht10_get_humedity(aht10_config_t*obj, uint8_t *data)
5 {
6   if (obj== NULL)
```

```
#define TEMPERATURE(A)                          (int8_t) ((A *0.000191)-50)
#define HUMEDITY(A)                             (uint8_t) (A *0.000095)
//Funcion para obtener el valor de la humeda
aht10_status_fnc aht10_get_humedity(aht10_config_t*obj, uint8_t *data)
{
  if (obj== NULL)
  {
    return AHT10_ERROR;
  }
  obj->status_fun=AHT10_ERROR;
  uint8_t bufferRead[6]={0};
  uint32_t data_humedity=0;
  obj->status_fun=aht10_launch_measurement(obj);
  if (obj->status_fun==AHT10_OK)
  {
    obj->status_fun= obj->readI2C(AHT10_ADDRESS_SLAVE,bufferRead,6);
    if (obj->status_fun==AHT10_OK)
    {
      data_humedity=(((uint32_t)bufferRead[1]<<16) | ((uint16_t)
    bufferRead[2]<<8) | (bufferRead[3]))>>4;
      *data= HUMEDITY(data_humedity);
    }
  }
  return obj->status_fun;
}
//Funcion para obtener el valor de la temperatura
aht10_status_fnc aht10_get_temperature(aht10_config_t*obj, int8_t *data)
{
  if (obj== NULL)
  {
    return AHT10_ERROR;
  }
  uint8_t buffer_read[6]={0};
  uint32_t data_temperature=0;
  obj->status_fun=AHT10_ERROR;
  obj->status_fun=aht10_launch_measurement(obj);
  if (obj->status_fun==AHT10_OK)
  {
    obj->status_fun=obj->readI2C(AHT10_ADDRESS_SLAVE ,buffer_read,6);
    if (obj->status_fun==AHT10_OK)
    {
      data_temperature=((uint32_t)(buffer_read[3] & 0x0F)<<16) | ((
    uint16_t) buffer_read[4]<<8)| buffer_read[5];
      *data= TEMPERATURE(data_temperature);
    }
  }
  return obj->status_fun;
}
```

CÓDIGO 3.1. Funciones de lectura de humedad y temperatura.

3.4.2. Controlador módulo de comunicación BG96

Se desarrollaron varias funciones para el driver BG96, que permiten configurar el módulo, configurar la red y las más importantes para el trabajo las de conexión, publicación y desconexión al servidor MQTT.

El código 3.2 muestra dos funciones desarrolladas en el driver BG96. La función *connect_server_mqtt()* utilizada para conectarse al servidor MQTT, internamente lo que hace es mandar el comando que se ve en la línea 6 y esperar la respuesta del módulo, que puede ser un OK o ERROR. La función *publish_message()* es una


```
{
  return AHT10_ERROR;
}
obj->status_fun=AHT10_ERROR;
uint8_t bufferRead[6]={0};
uint32_t data_humedity=0;
obj->status_fun=aht10_launch_measurement(obj);
if (obj->status_fun==AHT10_OK)
{
  obj->status_fun= obj->readI2C(AHT10_ADDRESS_SLAVE,bufferRead,6);
  if (obj->status_fun==AHT10_OK)
  {
    data_humedity=(((uint32_t)bufferRead[1]<<16) | ((uint16_t)
  bufferRead[2]<<8) | (bufferRead[3]))>>4;
    *data= HUMEDITY(data_humedity);
  }
}
return obj->status_fun;
}
//Funcion para obtener el valor de la temperatura
aht10_status_fnc aht10_get_temperature(aht10_config_t*obj, int8_t *data)
{
if (obj== NULL)
{
  return AHT10_ERROR;
}
uint8_t buffer_read[6]={0};
uint32_t data_temperature=0;
obj->status_fun=AHT10_ERROR;
obj->status_fun=aht10_launch_measurement(obj);
if (obj->status_fun==AHT10_OK)
{
  obj->status_fun=obj->readI2C(AHT10_ADDRESS_SLAVE ,buffer_read,6);
  if (obj->status_fun==AHT10_OK)
  {
    data_temperature=((uint32_t)(buffer_read[3] & 0x0F)<<16) | ((
  uint16_t) buffer_read[4]<<8)| buffer_read[5];
    *data= TEMPERATURE(data_temperature);
  }
}
return obj->status_fun;
}
```

CÓDIGO 3.1. Funciones de lectura de humedad y temperatura.

3.4.2. Controlador para el módulo de comunicación BG96

Se han creado diversas funciones destinadas al controlador BG96, las que posibilitan la configuración del módulo, la configuración de la red, y las más cruciales para el funcionamiento: las relacionadas con la conexión, publicación y desconexión del servidor MQTT.

El código 3.2 presenta dos funciones implementadas en el controlador BG96. La función *connect_server_mqtt()* se emplea para establecer la conexión con el servidor MQTT. En su interior, esta función envía el comando visible en la línea 6 y aguarda la respuesta del módulo, que puede ser *OK* o *ERROR*.

La función *publish_message()* es una de las más utilizadas por el firmware, es la encargada de publicar la trama JSON que contiene los datos de los sensores al tópico configurado inicialmente.

de las más utilizadas por el firmware, es la encargada de publicar la trama JSON que contiene los datos de los sensores al tópico configurado inicialmente.

```
//Funcion para conectarse al servidor MQTT
em_bg96_error_handling connect_server_mqtt(st_bg96_config *self)
  {
    self->ft_resp=FT_BG96_ERROR;
    char cmd[150]={0};
    sprintf(cmd,"AT+QMRCONN=%u,\"%s\",\"%s\",\"%s\"\r",self->self_mqtt.
    identifier_socket_mqtt,self->self_mqtt.mqtt_client_id,self->
    self_mqtt.mqtt_username,self->self_mqtt.mqtt_password);
    self->ft_resp=self->send_data_device(cmd,RS_BG96_CERO,self->
    buffer_resp,10000);
    if (self->ft_resp!=FT_BG96_OK)
    {
      self->last_error=BG96_ERROR_CONNECT_SERVER_MQTT;
    }
    return self->ft_resp;
  }
  //Funcion para publicar un mensaje al topico configurado
  em_bg96_error_handling publish_message(st_bg96_config *self,char *
    topic,char *data)
  {
    self->ft_resp=FT_BG96_ERROR;
    char cmd[50]={0};
    char buffer_data[220]={0};
    sprintf(buffer_data,"%s\x1a\r",data);
    sprintf(cmd,"AT+QMTPUB=%u,0,0,0,\"%s\"\r",self->self_mqtt.
    identifier_socket_mqtt,topic);
    self->ft_resp=self->send_data_device(cmd,RS_BG96_SIGNAL,self->
    buffer_resp,3000);
    if (FT_BG96_OK==self->ft_resp)
    {
      self->ft_resp=self->send_data_device(buffer_data,RS_BG96_CERO,self
    ->buffer_resp,15000);
      if (self->ft_resp!=FT_BG96_OK)
      {
        self->last_error=BG96_ERROR_PUBLISH_MESSAGE;
      }
   }
      else self->last_error=BG96_ERROR_PUBLISH_MESSAGE;
      return self->ft_resp;
  }
```

CÓDIGO 3.2. Función de conexión y publicación al broker MQTT.

Para el envío de mensajes de texto se desarrolló en el driver la función *send_sms_bg96()* que recibe como parámetros el número al que se desea mandar el mensaje y el texto del mensaje a enviar. El firmware utiliza esta función para mandar mensajes de alarma al usuario. El código 3.3 muestra la implementación de la función *send_sms_bg96()*

```
//Funcion para enviar mensaje de texto
em_bg96_error_handling send_sms_bg96(st_bg96_config *self,char*number,
    char*message)
{
  self->ft_resp=FT_BG96_OK;
  char buffer_message[256];
  char buffer_number[20];
  sprintf(buffer_number,"AT+CMGS=\"%s\"\r",number);
  sprintf(buffer_message,"%s\x1a\r",message);

```



```
//Funcion para conectarse al servidor MQTT
em_bg96_error_handling connect_server_mqtt(st_bg96_config *self)
  {
    self->ft_resp=FT_BG96_ERROR;
    char cmd[150]={0};
    sprintf(cmd,"AT+QMRCONN=%u,\"%s\",\"%s\",\"%s\"\r",self->self_mqtt.
    identifier_socket_mqtt,self->self_mqtt.mqtt_client_id,self->
    self_mqtt.mqtt_username,self->self_mqtt.mqtt_password);
    self->ft_resp=self->send_data_device(cmd,RS_BG96_CERO,self->
    buffer_resp,10000);
    if (self->ft_resp!=FT_BG96_OK)
    {
      self->last_error=BG96_ERROR_CONNECT_SERVER_MQTT;
    }
    return self->ft_resp;
  }
  //Funcion para publicar un mensaje al topico configurado
  em_bg96_error_handling publish_message(st_bg96_config *self,char *
    topic,char *data)
  {
    self->ft_resp=FT_BG96_ERROR;
    char cmd[50]={0};
    char buffer_data[220]={0};
    sprintf(buffer_data,"%s\x1a\r",data);
    sprintf(cmd,"AT+QMTPUB=%u,0,0,0,\"%s\"\r",self->self_mqtt.
    identifier_socket_mqtt,topic);
    self->ft_resp=self->send_data_device(cmd,RS_BG96_SIGNAL,self->
    buffer_resp,3000);
    if (FT_BG96_OK==self->ft_resp)
    {
      self->ft_resp=self->send_data_device(buffer_data,RS_BG96_CERO,self
    ->buffer_resp,15000);
      if (self->ft_resp!=FT_BG96_OK)
      {
        self->last_error=BG96_ERROR_PUBLISH_MESSAGE;
      }
   }
      else self->last_error=BG96_ERROR_PUBLISH_MESSAGE;
      return self->ft_resp;
  }
```


CÓDIGO 3.2. Función de conexión y publicación al broker MQTT.

Para el envío de mensajes de texto se desarrolló en el driver la función *send_sms_bg96()* que recibe como parámetros el número al que se desea mandar el mensaje y el texto del mensaje a enviar. El firmware utiliza esta función para mandar mensajes de alarma al usuario. El código 3.3 muestra la implementación de la función *send_sms_bg96()*

```
//Funcion para enviar mensaje de texto
em_bg96_error_handling send_sms_bg96(st_bg96_config *self,char*number,
    char*message)
{
  self->ft_resp=FT_BG96_OK;
  char buffer_message[256];
  char buffer_number[20];
  sprintf(buffer_number,"AT+CMGS=\"%s\"\r",number);
  sprintf(buffer_message,"%s\x1a\r",message);

  self->ft_resp=self->send_data_device(buffer_number,RS_BG96_SIGNAL,self
    ->buffer_resp,12000);
  if (FT_BG96_OK==self->ft_resp)
```

```
  self->ft_resp=self->send_data_device(buffer_number,RS_BG96_SIGNAL,self
    ->buffer_resp,12000);
  if (FT_BG96_OK==self->ft_resp)
  {
    self->ft_resp=self->send_data_device(buffer_message,RS_BG96_OK,self
    ->buffer_resp,12000);
    if (FT_BG96_OK!=self->ft_resp)
    {
      self->last_error=BG96_ERROR_SEND_SMS;
    }
  }
  return self->ft_resp;
}
```

CÓDIGO 3.3. Función para enviar sms.

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 .

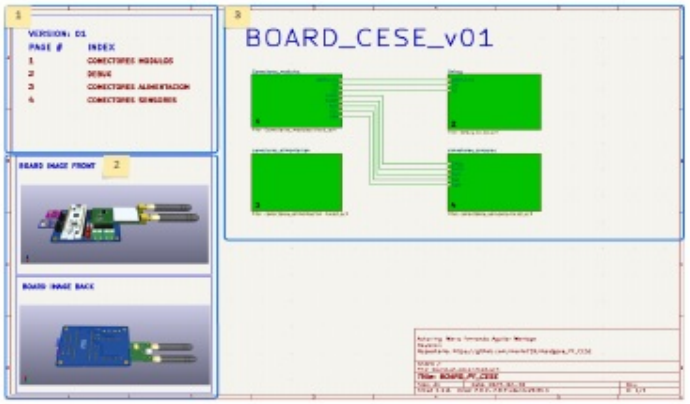


FIGURA 3.10. Esquemático página raíz.


```
  {
    self->ft_resp=self->send_data_device(buffer_message,RS_BG96_OK,self
    ->buffer_resp,12000);
    if (FT_BG96_OK!=self->ft_resp)
    {
      self->last_error=BG96_ERROR_SEND_SMS;
    }
  }
  return self->ft_resp;
}
```

CÓDIGO 3.3. Función para enviar SMS.

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

Dado que se trata de un prototipo, se diseñó una tarjeta que permite la integración y conexión de los componentes utilizados en el trabajo, incluyendo el módulo de comunicación celular, la 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 .

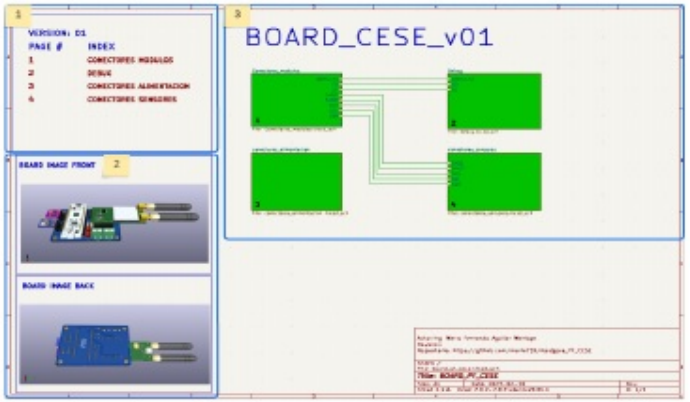


FIGURA 3.10. Esquemático página raíz.

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.

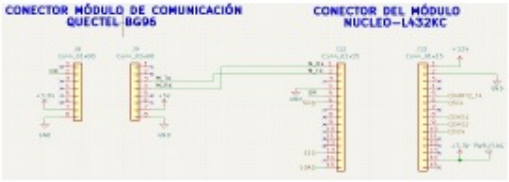


FIGURA 3.11. Conector módulo BG96 y NUCLEO-L432KC.

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.

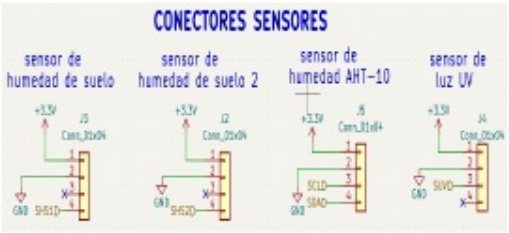


FIGURA 3.13. Esquemático conectores sensores.

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.

22 Capítulo 3. Diseño e implementación

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.

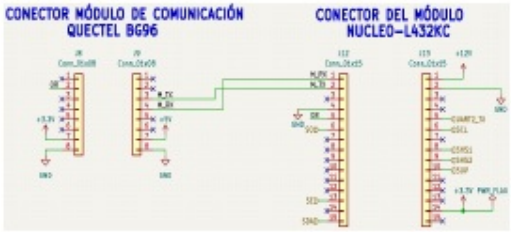


FIGURA 3.11. Conector módulo BG96 y NUCLEO-L432KC.

Se incorporaron dos leds con fines de depuración: uno para indicar si se logró la conexión al servidor MQTT y el otro para señalar cualquier estado de error del sistema. También se incluyó un conector para un puerto serial a través del cual el módulo transmite las secuencias de comandos enviadas y recibidas del módulo de comunicación. En la 3.12 se puede ver el circuito implementado.

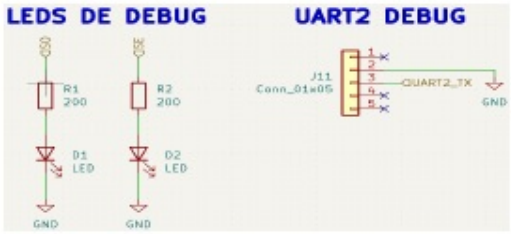


FIGURA 3.12. Esquemático interfaz de debug.

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.

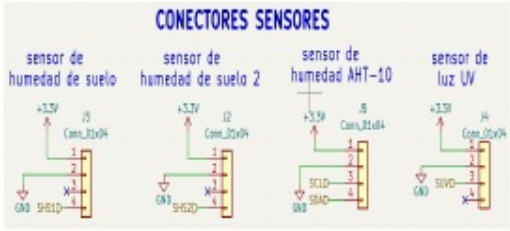
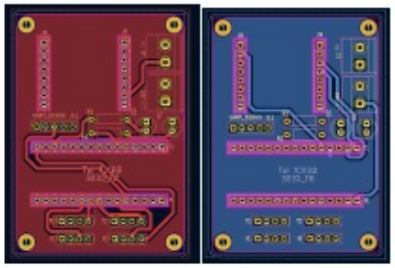


FIGURA 3.13. Esquemático conectores sensores.

FIGURA 3.14. Esquemático conectores alimentación.

3.5.2. PCB del hardware

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



(A) Capa top PCB (B) Capa bot PCB

FIGURA 3.15. PCB del proyecto.

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

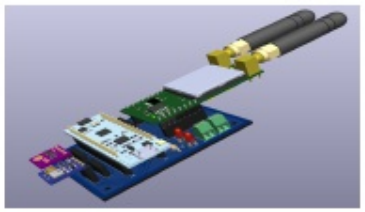


FIGURA 3.16. Modelo 3D de la tarjeta.

3.5.3. Fabricación circuito impreso

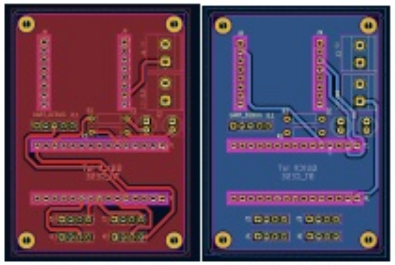
Una vez completado y validado el diseño, se generaron los archivos de fabricación y se mandaron a la empresa JLCPCB parar su producción. La figura 3.17 muestra la tarjeta ya ensamblada con los módulos.

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 de alimentación.

3.5.2. PCB del hardware

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



(A) Capa top PCB (B) Capa bot PCB

FIGURA 3.15. PCB del trabajo.

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

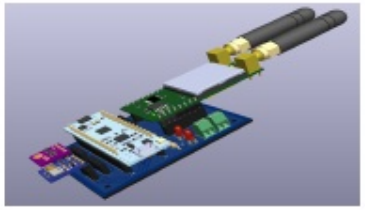


FIGURA 3.16. Modelo 3D de la tarjeta.

3.5.3. Fabricación del 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.

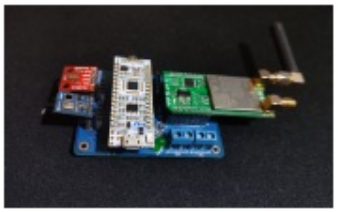


FIGURA 3.17. PCB ensamblado.

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.

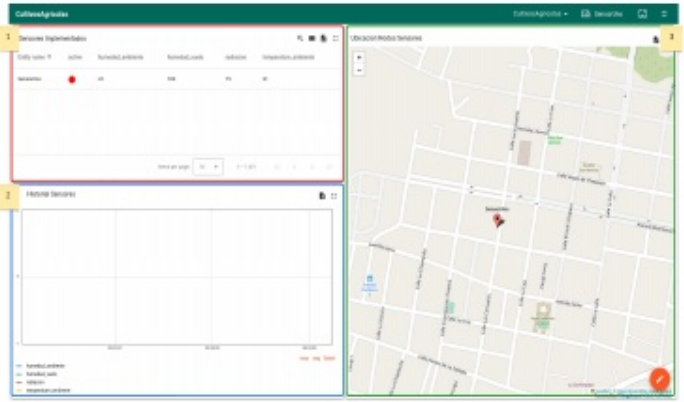


FIGURA 3.18. Panel principal de la interfaz gráfica.

24 Capítulo 3. Diseño e implementación

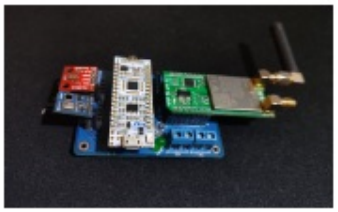


FIGURA 3.17. PCB ensamblado.

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 de gráfica, tablas, etc.

3.6.1. Panel principal

La Figura 3.18 muestra el panel principal de la interfaz gráfica. A continuación, se detallarán las distintas áreas que componen este panel. Este se divide 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 representan la evolución en el tiempo de los valores de las variables registradas por los sensores.
- Zona 3: mapa con la ubicación de los nodos sensores implementados.

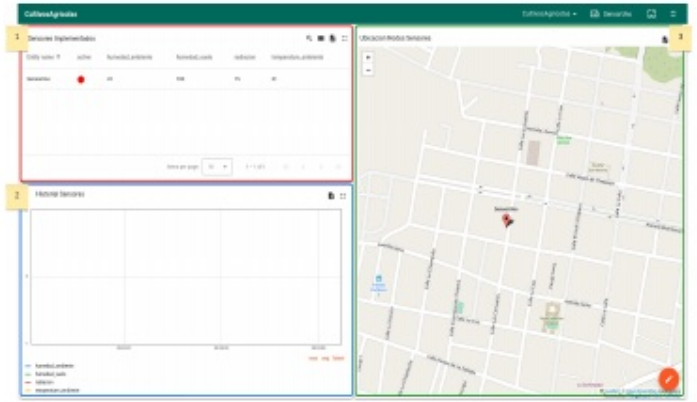


FIGURA 3.18. Panel principal de la interfaz gráfica.

Capítulo 4

Conclusiones

4.1. 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 puedo 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?

4.2. Próximos pasos

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

Capítulo 4

Ensayos y resultados

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

4.1. Pruebas unitarias

Para la implementación de los drivers se utilizó la metodología de desarrollo TDD. Esto implica que se escribieron pruebas unitarias para los drivers. Se utilizó ceedling como herramienta de desarrollo de pruebas automáticas.

En el código 4.1 se muestra la prueba implementada para la función de *aht10_get_status()* del driver del sensor AHT10. Internamente utiliza funciones mock para la lectura y escritura por protocolo I2C, líneas 6,11 y 15. Se prueban dos casos: cuando la lectura de los registros es 0 y cuando es 255.

```
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,
       AHT10_OK);
7    read_I2C_STM32L432_port_ReturnThruPtr_buffer(&data);
8    TEST_ASSERT_EQUAL(SENSOR_IDLE, aht10_get_status(&aht10config));
9
10   data=255;
11   read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE, buffer, 1,
       AHT10_OK);
12   read_I2C_STM32L432_port_ReturnThruPtr_buffer(&data);
13   TEST_ASSERT_EQUAL(SENSOR_BUSY, aht10_get_status(&aht10config));
14
15   read_I2C_STM32L432_port_ExpectAndReturn(AHT10_ADDRESS_SLAVE, buffer, 1,
       AHT10_ERROR);
16   TEST_ASSERT_EQUAL(SENSOR_BUSY, aht10_get_status(&aht10config));
17 }
```

CÓDIGO 4.1. Tests del driver del sensor AHT10.

En el código 4.2 se muestra la prueba desarrollada para la función *send_sms_bg96()* encargada de mandar los mensajes de texto del driver BG96. Se probaron varios casos, cuando la función de envío de comandos responde con un OK y cuando responde con errores.

```
1  //Prueba de la funcion para mandar sms
2  void test_send_sms_bg96(void)
3  {
```

```
4   char buffer_resp[30]={0};
5   send_data_ExpectAndReturn("AT+CMGS=\"72950576\"\r",RS_BG96_SIGNAL,
      buffer_resp,12000,FT_BG96_OK);
6   send_data_ExpectAndReturn("HOLA\x1a\r",RS_BG96_OK,buffer_resp,12000,
      FT_BG96_OK);
7   TEST_ASSERT_EQUAL(FT_BG96_OK,send_sms_bg96(&config_module,"72950576","
      HOLA"));
8
9   send_data_ExpectAndReturn("AT+CMGS=\"72950576\"\r",RS_BG96_SIGNAL,
      buffer_resp,12000,FT_BG96_ERROR);
10  TEST_ASSERT_EQUAL(FT_BG96_ERROR,send_sms_bg96(&config_module,"72950576
      ","HOLA"));
11
12  send_data_ExpectAndReturn("AT+CMGS=\"72950576\"\r",RS_BG96_SIGNAL,
      buffer_resp,12000,FT_BG96_OK);
13  send_data_ExpectAndReturn("HOLA\x1a\r",RS_BG96_OK,buffer_resp,12000,
      FT_BG96_ERROR);
14  TEST_ASSERT_EQUAL(FT_BG96_ERROR,send_sms_bg96(&config_module,"72950576
      ","HOLA"));
15 }
```

CÓDIGO 4.2. Tests del driver del modulo bg96.

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

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.

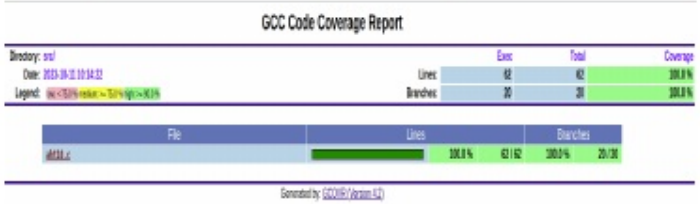


FIGURA 4.1. Informe de cobertura driver aht10.

FIGURA 4.2. Informe de cobertura driver BG96.

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://ssdielect.com/temperatura/3885-aht10.html.

[10] naylampmechatronics. *Especificacion de Producto,MÓDULO SENSOR DE LUZ ULTRAVIOLETA (UV) ML8511*. Visitado: 2023-05-27. URL: https://ssdielect.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] CEEDLING. *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: hhttps://learn.sparkfun.com/tutorials/i2c/all.

[17] MQTT. *Descripción del protocolo,Introduccion a MQTT*. Visitado: 2023-05-27. URL: https://mqtt.org/.

4.2. Pruebas de la plataforma IoT

4.2.1. Prueba de inyección de mensajes

El objetivo de la prueba de inyección de mensajes a la plataforma IoT, es verificar la llegada de los mensajes mandados por un cliente MQTT al broker de ThingsBoard. Para realizar el envío de datos al broker, se utilizó el programa mosquitto. Para realizar esta prueba se ejecutó el comando de mosquitto que se muestra en la figura 4.3.

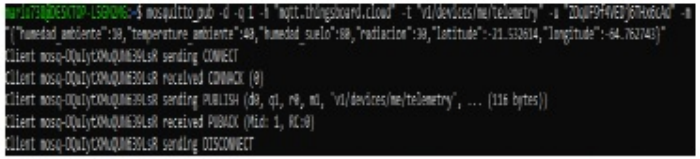


FIGURA 4.3. Envío de datos por el cliente MQTT de mosquitto.

Donde:

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

Al ejecutar el comando de la figura 4.3, el cliente de mosquitto primeramente se conecta al broker MQTT, luego publica el mensaje en el tópico y finalmente, se desconecta del servidor.

Para comprobar la llegada de los datos al broker de ThingsBoard, se tiene que ir a la sección dispositivos, seleccionar el dispositivo al que se le envio los datos y entrar a la pestaña de última telemetría. En la figura 4.4 podemos ver que los datos enviados llegan correctamente.

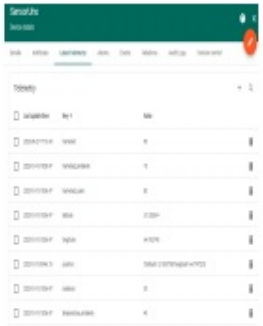


FIGURA 4.4. Recepción de datos en el broker MQTT.

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

4.2.2. Prueba de la tabla de alarmas del panel visualización

ThingsBoard permite configurar alarmas con respecto a las variables monitoreadas por el sistema. En el panel de visualización de cada sensor se tiene una tabla de alarmas, que muestra las notificaciones de las alarmas que se activaron. En la figura 4.5 podemos ver la notificación de una alarma cuando la temperatura ambiente sobrepasa los 43°C.

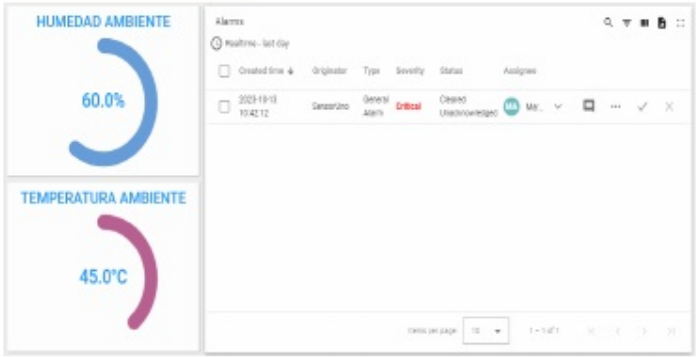


FIGURA 4.5. Tabla de alarmas activas.

4.2.3. Prueba del widget de mapa

En los paneles de visualización tenemos mapas que muestran la ubicación del dispositivo implementado. En la figura 4.6 podemos ver la ubicación del prototipo implementado para el trabajo.

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 las gráficas de los paneles de visualización con un entorno de tiempo más amplio. En las gráficas se