Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica.

Diseño y desarrollo de un sistema de precintado remoto sobre FreeRTOS para sistemas IoT.

Autor: Carmen Vivancos Porras
Tutor: Ramón González Carvajal
Eduardo Hidalgo Fort

Departamento de Ingniería Electrónico
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica.

Diseño y desarrollo de un sistema de precintado remoto sobre FreeRTOS para sistemas IoT.

Autor:
Carmen Vivancos Porras

Tutor:
Ramón González Carvajal
Catedrático
Eduardo Hidalgo Fort
Investigador Postdoctoral

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Grado: Diseño y desarrollo de un sistema de precintado remoto sobre FreeRTOS para sistemas IoT.

Autor: Carmen Vivancos Porras

Tutor: Ramón González Carvajal

Eduardo Hidalgo Fort

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mis padres

A mis abuelos

A mis amigos

AGRADECIMIENTOS

Son muchas las personas gracias a las cuales estoy donde estoy y que, cada una a su manera, me han ayudado a llegar hasta aquí, a todas ellas gracias.

Gracias a mis padres por darlo todo por mi, hasta cuando no tenían, y por no dejar que me rindiera nunca. A papá por enseñarme que la humildad es el mayor valor que puede tener una persona. A mamá por inculcarme que la constancia es la clave del éxito, y tenía razón. A mis abuelos porque verlos orgullosos de mi es mi mayor premio.

Gracias a mis amigos por siempre estar ahí, por aguantarme y darme ánimos a pesar de no saber (aún) de que va mi carrera. A Celia, Irene y Marta por ser mi norte, mi casa y donde acudir cuando la vida se tambalea.

Al voleibol, y en concreto a mi equipo, por ser siempre la mejor vía de escape.

A mis compañeros de clase, porque sin ellos, este camino no hubiera sido el mismo, ni tan divertido. A Juanfran y a Mari, por no solo ser compañeros, amigos y hermanos, sino por ser dos pilares fundamentales para mi en la escuela, y en mi vida.

Gracias a mis profesores, por los que me han hecho estar aquí gracias a que sus clases fueron un empujón hacia arriba, como por los que me enseñaron a mi misma lo que no era para mi. Y de todos los profesores que han pasado por mi vida estos años gracias en especial a mi tutor, Ramón, porque nunca me reí y aprendí tanto en una clase como en las suyas.

A Eduardo, y, sobre todo, a Jesús por agunatarme estos meses y ayudarme en absolutamente todo.

Por último, gracias a mi, por ser siempre una cabezona y saber que rendirse no era una opción.

Carmen Vivancos Porras

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sevilla, 2022

RESUMEN

Las comunicaciones inalámbricas, han experimentado una gran expansión en los últimos años en nuestras vidas y se hace casi imposible encontrar un producto o servicio donde no pueda ser implantado un transceptor inalambrico para comunicar información hacia o desde internet. Este es el propósito del Internet de las Cosas (IoT), el poder interconexionar dispositivos electrónicos de los que estamos rodeados a diario en cualquier momento, en cualquier lugar y sin que ningún ser humano tenga que intervenir.

Con casi 35 billones de dispositivos conectados a través de alguna red de IoT hay algunas funcionalidades que están en continua mejora como es el consumo. La idea de este trabajo es indagar en estos sistemas inalámbricos de bajo consumo que nos permiten enviar y, sobre todo, recibir información desde internet y enfocarlos hacia el desarrollo de aplicaciones para actuación remota, en concreto para la apertura y cierre de un actuador solenoide.

El objetivo final del proyecto no es el de hacer un sistema optimo en consumo o latencia sino, demostrar que las tecnologías inalámbricas actuales permiten cubrir estas funcionalidades. Se desarrollará así una aplicación con dos drivers, uno de la misma actuación del solenoide, y otro para el envio y recepción de datos (solicitud/respuesta) del backend, y se implementará en el sistema operativo FreeRTOS para validar su funcionalidad en un entorno real.

ABSTRACT

Wireless communications have experienced a great expansion in recent years in our lives and it is almost impossible to find a product or service where a wireless transceiver cannot be implemented to communicate information to or from the Internet. This is the purpose of the Internet of Things (IoT), the power to interconnect electronic devices that we are surrounded by every day at any time, anywhere and without any human being having to intervene.

With almost 35 billion devices connected through an IoT network, there are some functionalities that are constantly improving, such as consumption. The idea of this work is to investigate these low-consumption wireless systems that allow us to send and, above all, receive information from the Internet and focus them on the development of applications for remote actuation, specifically for the opening and closing of a solenoid actuator.

The final objective of the project is not to make an optimal system in terms of consumption or latency, but rather to demonstrate that current wireless technologies allow these functionalities to be covered. Thus, an application with two drivers will be developed, one for the same actuation of the solenoid, and another for sending and receiving data (request/response) from the backend, and it will be implemented in the FreeRTOS operating system to validate its functionality in a real environment.

ÍNDICE

Agradecimientos **29**

Resumen **31**

Abstract **32**

Índice **34**

Índice de Tablas **37**

Índice de Figuras **38**

Índicie de Ilustraciones **i**

Notación **ii**

1 Introducción **1**
1.1. Motivación *2*
1.2. Objetivos *2*
1.3. Metodología *3*

2 Estado del arte dispositivos de precintado remoto **4**

3 Estado del arte de la recepción de información en sistemas IoT de bajo consumo **7**
LTE-M *9*
NB-IoT *10*
LoRaWAN *11*
Sigfox *13*
Selección de la tecnología para la transmisión de datos *15*

4 Hardware **17**
4.1. DSML-0224-12 *18*
4.2. Placa Núcleo STM32 *20*
4.3. FT4232H-56Q Mini Module *24*
*4.4. Transceptor Sigfox AX-SIGFOX ANTSATMP ** *25*
4.5. NB-IoT HAT basado en SIM7080G *26*
4.6. Conexión del hardware *28*

5 Diseño y desarrollo del driver del solenoide **32**
5.1. Definición del problema y diseño de la solución *32*
5.2. Desarrollo de la solución *33*
uint8_t SOL_Write_PIN(GPIO_PIN configPIN, uint8_t val) *33*
uint8_t SOL_Read_PIN(GPIO_PIN configPIN) *33*
uint8_t SOL_Conf_Init() *33*
uint8_t SOL_Actuacion(uint8_t UART_PORT, GPIO_PIN configPIN, char esperado) *34*

6 Diseño y desarrollo de la recepción **36**
INICIALIZACIÓN: *36*
SIM7080_Init (uint8_t UART_Port, uint32_t BaudRate, GPIO_PIN PWRKEY_Pin, uint8_t Comissioning, uint8_t Echo_mode) 36
SIM7080_CheckNetworkConnection () 36
SIM7080_Comissioning (uint8_t Comissioning) 36
SIM7080_InitNetwork () 36
SIM7080_InitConnection (char *tp_mode, char *Port, char *Ip, uint8_t socket) 37
SOLICITUD: *37*
SIM7080_TransmitDatagram (uint8_t socket, char *msg) 37

RESPUESTA: *37*
_SIM7080_SendATCommand (char* at_command, const char* expected_response, char *response, uint16_t response_size, uint32_t response_timeout) 37
_SIM7080_GetSubstring (char* message, char* from_this_string, char* to_this_string, char* substring) 38

7 Sobre FreeRTOS **39**
7.1. Tareas *40*
7.1.1. xTaskCreate() 41
7.1.2. Prioridad de las Tareas 42
7.2. Colas y semáforos *42*
7.2.1. Colas: 42
7.2.1.1. xQueueCreate() 43
7.2.1.2. xQueueSendToBack(), xQueueSendToFront() y xQueueSend() 44
7.2.1.3. xQueueReceive() 44
7.2.2. Semáforos: 45
7.2.2.1. vSemaphoreCreateBinary() 46
7.2.2.2. xSemaphoreTake () 46
7.2.2.3. xSemaphoreGive () 46

8 Implementación en FreeRTOS **47**
vSolicitudApertura() *50*
vActuacionSOL() *52*

9 Pruebas de validación **54**

10 Conclusiones y líneas futuras **59**

Anexo **61**
¿Qué es un actuador solenoide y para qué sive? *61*
Código de las funciones *63*
Comandos AT usados *68*

Referencias **81**

ÍNDICE DE TABLAS

Tabla 1. Comparativa entre las tecnologías LTE-M, NB-IoT, LoRaWAN y Sigfox.
Tabla 2. Tabla comparativa entre modelos de actuadores solenoides.
Tabla 3. Tabla modelos actuador DSOL-0844.
Tabla 4. Definición de los pines de SIM7080G.
Tabla 5. Comandos AT usados para establecer el Baud Rate del módulo NBIoT.
Tabla 6. Valores definidos del comando AT+IPR.
Tabla 7. Comandos AT usado para resetear el módulo NBIoT.
Tabla 8. Comandos AT usado para reiniciar el módulo si se bloquea.
Tabla 9. Valores definidos del comando AT+CEDUMP.
Tabla 10. Comandos AT usado para muestrear el ICCID del módulo NBIoT.
Tabla 11. Comandos AT usado para seleccionar CAT-M o NB-IoT.
Tabla 12. Valores definidos del comando AT+CMNB.
Tabla 13. Comandos AT usados para establecer el modo Echo del módulo NBIoT.
Tabla 14. Valores definidos del comando AT+ATE.
Tabla 15. Comandos AT usado para cerrar una conexión TCP/UDP del módulo NBIoT.
Tabla 16. Comandos AT usados para la recepción a través del módulo NBIoT.
Tabla 17. Valores definidos del comando AT+CARECV.
Tabla 18. Comandos AT usados para el envío de datos a través del módulo NBIoT.
Tabla 19. Valores definidos del comando AT+CASEND.
Tabla 20. Comandos AT usados para seleccionar el operador del módulo NBIoT.
Tabla 21. Valores definidos del comando AT+COPS.
Tabla 22. Comandos AT usados para obtener red APN con el módulo NBIoT.
Tabla 23. Valores definidos del comando AT+CGNAPN.
Tabla 24. Comandos AT usado para establecer el estado de registro de red del módulo NBIoT.
Tabla 25. Valores definidos del comando AT+CGREG.
Tabla 26. Comandos AT usado para la configuración del modo de ahorro de energía del módulo NBIoT.
Tabla 27. Valores definidos del comando AT+CCPSMS.
Tabla 28. Comandos AT usado para consultar el envio de info de datos del módulo NBIoT.
Tabla 29. Valores definidos del comando AT+CAACK.
Tabla 30. Comandos AT usado para configurar la banda del módulo NBIoT.
Tabla 31. Valores definidos del comando AT+CBANDCFG.

ÍNDICE DE FIGURAS

Figura 1. Evolución del número de dispositivos conectados al IoT en todo el mundo. [2015-2025].
Figura 2. Ejemplo de “red estrella”.
Figura 3. Gráfica comparativa características NB-IoT.
Figura 4. Gráfica comparativa características LoRaWAN.
Figura 5. Arquitectura detallada de LoRaWAN.
Figura 6. Formato mensaje *uplink* LoRaWAN.
Figura 7. Formato mensaje *downlink* LoRaWAN.
Figura 8. Transmision LoRaWAN clase A.
Figura 9. Transmision LoRaWAN clase B.
Figura 10. Transmision LoRaWAN clase C.
Figura 11. Gráfica comparativa características Sigfox.
Figura 12. Formato mensajes *uplink* y *downlink* de Sigfox.
Figura 13. Esquema funcionamiento de la bidereccionalidad de la comunicación Sigfox.
Figura 14. Diagrama de la implementación general del proyecto.
Figura 15. Dimensiones mecánicas del actuador.
Figura 16. Vista isométrica del actuador.
Figura 17. Gráfica Fuerza vs Carrera (distancia recorrida por el émbolo).
Figura 18. Layout del Top STM32 Nucleo- L152RE.
Figura 19. Pinout del Núcleo-L152RE.
Figura 20. Leyenda de pines de la placa.
Figura 21. Pin Out CN6/CN8 placa STM32L152RE.
Figura 22. Pin Out CN10 placa STM32L152RE.
Figura 23. Conexiones eléctricas (pines) del FT4232H-56Q Mini Module.
Figura 24. Pines Módulo AX-SIGFOX ANTSTAMP.
Figura 25. Diagrama de bloques SIM7080G.
Figura 26. Esquema del setup.
Figura 27. Esquema montaje actuador.
Figura 28. Diagrama función *SOL_Config_Init()*.
Figura 29. Diagrama función *SOL_Actuacion()*.
Figura 30. Diagrama driver NB-IoT.

Figura 31. Esquema general de FreeRTOS.
Figura 32. Transiciones de estado de tareas FreeRTOS.
Figura 33. Ejemplo de tarea en estado Blocked por *vTaskDelay().*
Figura 34. Prototipo de la función *xTaskCreate()*.
Figura 35. Funcionamiento de una Cola.
Figura 36. Prototipo de la función *xQueueCreate().*
Figura 37. Prototipo de la función *xQueueSendToFront(),xQueueSendToBack() y xQueueSend().*
Figura 38. Prototipo de la función *xQueueReceive().*
Figura 39. Ejemplo de semáforo mutex.
Figura 40. Prototipo de la función *vSemaphoreCreateBinary().*
Figura 41. Prototipo de la función *xSemaphoreTake().*
Figura 42. Prototipo de la función *xSemaphoreGive().*
Figura 43. Representación de tareas en FreeRTOS.
Figura 44. Código de la tarea *vInicializacion().*
Figura 45. Código de la tarea *vSolicitudApertura().*
Figura 46. Código de la tarea *vActuacionSOL().*
Figura 47. Diagrama tarea vSolicitudApertura().
Figura 48. Diagrama tarea vActuacionSOL().
Figura 49. Comunicación Docklight con el módulo NB-IoT.
Figura 50. Comunicación Docklight con el microcontrolador.
Figura 51. Esquema mejora con App.
Figura 52. Esquema funcional actuador solenoide.
Figura 53. Esquema general actuador solenoide.
Figura 54. Ejemplos de Duty Cycle.

ÍNDICIE DE ILUSTRACIONES

Ilustración 1. Actuador DSOL-0844-05.
Ilustración 2. Placa STM32 Nucleo-L152RE.
Ilustración 3. FT4232H-56Q Mini Module.
Ilustración 4. Módulo AX-SIGFOX ANTSTAMP.
Ilustración 5. Modulo NB-IoT HAT basado en SIM7080G.
Ilustración 6. Placa SIM7080G de SIMCOM.
Ilustración 7. Identificación de los pines de los módulos (detalle).
Ilustración 8. Setup real del hardware.

NOTACIÓN

BW	Band Wide
CRC	Cyclic Redundancy Check
CSS	Chirp Spread Spectrum
GIE	Grupo de Ingeniería Electrónica (Universidad de Sevilla)
GPIO	General Purpose Input/Output, Entrada/Salida de Propósito General
GSM	Global System for Mobile communications
HATs	Hardware Attached on TOP
I2C	Inter-Integrated Circuit
ICCID	Integrated Circuit Card ID
IoT	Internet of Things
LED	Light Emitting Diode
LPWAN	Low Power Wide Area Network
LTE	Long Term Evolution
M2M	Machine to Machine
MCU	MicroController Unit
RTC	Real Time Clock
RTOS	Real Time Operating System
SPI	Serial Peripheral Interface
TTL	Time To Live
UART	Universal Asynchronous serieal Receiver and Transmiter
UNB	Ultra Narrow Band
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WPAN	Wireless Personal Area Network
WWAN	Wireless Wide Area Network

1 INTRODUCCIÓN

"Life can only be understood backwards; but it must be lived forwards."

- Søren Kierkegaard-

Las comunicaciones inalámbricas, desde hace ya bastantes años, han sufrido una gran expansión en la vida de todos los seres humanos y se han convertido en la mejor alternativa a los sistemas cableados. El 'Internet of Things' o Internet de las cosas (IoT, por sus siglas en inglés) es la capacidad de los objetos de transferir datos por internet sin necesidad de interacciones, es decir, sin que ningún usuario tenga que intervenir y, además, el ecosistema donde se desarrollan multitud de aplicaciones de interés. Desde el planeamiento del tráfico en la ciudad, pasando por el cuidado de grandes extensiones de cultivos hasta nuestros propios hogares. Por lo tanto, es necesario que la comunicación de estos sistemas inalámbricos se realice consiguiendo reducir costes y de manera energéticamente eficiente y robusta para afrontar los problemas del entorno.

La revolución del Internet de las Cosas es casi imparable y apunta alto para ser una tecnología imprescidible en ámbitos como la atención sanitaria, el comercio, el transporte e incluse los servicios públicos y para **2025**, se prevé, que haya cerca de **75.000 millones de dispositivos IoT en funcionamiento** en todo el mundo.

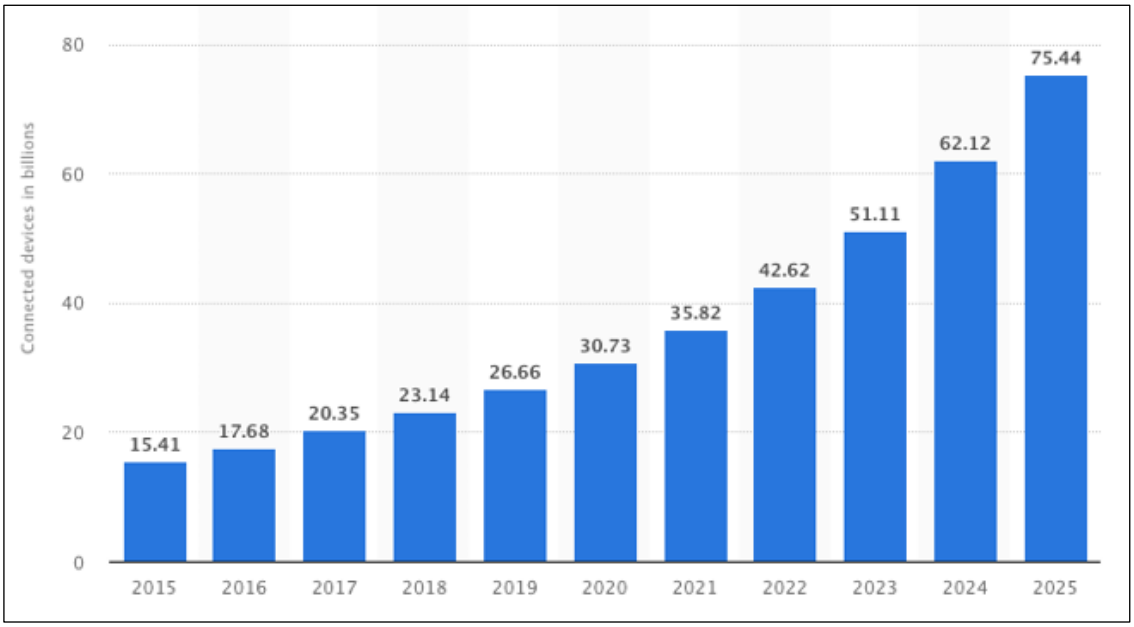

Figura 1. Evolución del número de dispositivos conectados al IoT en todo el mundo [2015-2025]. [1]

En este proyecto se busca el desarrollo de un driver para controlar un actuador solenoide (que podría ser una puerta, cerradura, cofre de moto…) remotamente con la ayuda de la tecnología NB-IoT. Además, se implementará en el sistema operativo FreeRTOS para comprobar que la aplicación es totalmente funcional y compatible con los RTOS.

1.1. Motivación

La motivación de este Trabajo de Fin de Grado es la curiosidad e interés por las comunicaciones inlambricas, de la cantidad de aplicaciones que ofrece dentro del ámbito IoT y, en concreto, por aquellas que proporcionan un bajo consumo.

Las soluciones IoT se encuentran cada vez en más actividades cotidianas, industriales y económicas. Desde los sensores intregrados en los edificios, que permiten recopilar datos sobre su uso y así regular automáticamente la calefacción e iluminación de este para así ahorrar energía, hasta electrodomésticos, como podría ser una nevera conectada a internet que puede avisarnos si se ha quedado abierte o si se ha terminado la leche, por ejemplo.

Como dentro de este ámbito de las redes inalámbricas y de las aplicaciones IoT existe una gran variedad de opciones y sería imposible abarcarlas todas ellas en un solo proyecto, se ha optado por un estudio de los dispositivos de precintado remoto y de la recepción de información en sistemas IoT de bajo consumo, y el desarrollo de una aplicación que, basada en una de estas redes inalámbricas, permita una solicitud/recepción de datos.

1.2. Objetivos

El objetivo de este proyecto, por un lado, es el de desarrollar un driver de control para un sistema de precintado remoto en un entorno IoT. Y por el otro el desarrollo de un segundo driver para la recepción de información de sistemas IoT.

La idea principal de este proyecto es la de poder abrir el actuador solenoide desde un servidor cuando sea necesario. Pero hay que dejar claro como va a ser este proceso debido a sus limitaciones.

Normalmente la solicitud de apertura, como, por ejemplo, el cofre de una moto de alquiler (Acciona, Muving, Yego…), se enviaría desde el usuario hacia la moto. Desde una aplicación clicaríamos en “Abrir cofre” por ejemplo y desde el servidor nos contestarían abriéndolo. En este caso no va a ser así y se realizará como se detalla a continuación:

como se trata de dispositivos de bajo consumo no va a ser posible enviarle una solicitud cuando nosotros queramos porque el microcontrolador va a estar “dormido”. Lo que va a hacer este es que, periódicamente, se va a “despertar” y le comunicará al servidor que los está y si necesita que haga algo. En este momento si será cuando podrá recibir la orden, abrir el solenoide en nuestro caso. Podría ser cualquier otra: encerder las luces, arrancar el motor, etc. Además, los nodos IoT no tienen por qué tener una IP fija (lo que supone un coste adicional importante), lo que también impide que la comunicación la inicie el servidor.

Esta es la razón por la que tenemos que enviar primero un mensaje hacia arriba, *uplink*, hacia el servidor, porque el micro siempre esta dormido. No tiene sentido enviar algo hacia abajo, *downlink*, desde el servidor hacia un módulo que no sabemos si estará despierto o no. Por esto, esa va a ser la clave, deberemos esperar a que el micro indique que está disponible (despierto) y así se podrá solicitar que haga lo que yo quiero. El nodo tiene que enviar primero, aunque la orden la queramos dar hacía abajo.

El primero de los drivers se encargará de todo lo relacionado con el control del actuador, lo necesario para la configuración de su pin GPIO, y las funciones que realicen las tareas de apertura y cierre del solenoide.

El segundo, será el que se encargue de la recepción de la información por el módulo NB-IoT a través de la UART.

Para poder llevar a cabo el desarrollo software es muy importante señalar que se ha hecho uso de un módulo de comunicación que opera en NB-IoT, el SIM7080G. Se ha usado este porque nos permitía enviar y, sobre todo, recibir datos desde el backend que era lo esencial para este proyecto. Por esto, aunque inicialmente el proyecto se planteó con el módulo AX.SF10-ANT21-868 de tecnología Sigfox, este fue descartado.

Como resultado, vamos a verificar el correcto funcionamiento del driver y comprobar que el solenoide actúa cuando el usuario se lo solicite.

Por último, se realizará la implementación de la aplicación en FreeRTOS, un sistema operativo en tiempo real, con el fin de comprobar que la funcionalidad de dicha aplicación dentro de un entorno real es compatible.

1.3. Metodología

La estructura de este proyecto tiene como fin ir, progresivamente, adquiriendo los conocimientos necesarios para poder desarrollar la solución que queremos obtener de la manera más eficiente y eficaz que podamos.

Para ello, en el desarrllo del proyecto, reflejado en esta memoria, trataremos los siguientes aspectos, secuencialmente:

- En un primero lugar, se hará un estudio del estado del arte sobre los dispositivos de precintado remoto, así como de la recepción de información en sistemas IoT de bajo consumo.
- A continuación, se detallan los componentes hardware usados, su arquitectura y las características que nos ofrecen para su implementación.
- El siguiente paso será el diseño y desarrollo de los *drivers*, tanto el que necesitamos para actuar con el solenoide, como el que necesitamos para la recepción.
- A continuación, se presentará el sistema operativo FreeRTOS y todas las funcionalidades que nos ofrece para desarrollar nuestra aplicación, como pueden ser las tareas, las colas, los semáforos….
- Por ultimo, implementaremos el driver en FreeRTOS y desarrollaremos tareas con él para comprobar su correcto funcionamiento en un entorno real.
- Integración en un marco de producción.

A continuación, se va a realizar el estudio del estado actual de los dispositivos de precintado remoto justificando así este Trabajo de Fin de Grado.

2 ESTADO DEL ARTE DISPOSITIVOS DE PRECINTADO REMOTO

Una vez que conocemos qué es el Internet de las Cosas, el siguiente paso es saber que nos puede ofrecer, y esta es la visión que pretende dar este capítulo.

La IoT ha evolucionado tanto en los últimos años que prácticamente hoy en día, abarca cualquier campo de la vida que podamos imaginar, tanto es así, que cada vez encontramos más dispositivos capaces de conectarse a Internet, comunicarse entre sí o que podamos controlar de forma remota, como las bombillas inteligentes o los interruptores.

Como es casi imposible abarcar todas las aplicaciones que es capaz de IoT, vamos a agruparlas en tres grandes grupos:

- Seguridad.
- Control de calidad y de gestión.
- Monitorizacion y seguimiento.

Aplicaciones como la industria energética, la domótica, la agricultura o la ganadería se sirven de todas estas áreas, cada una de ellas beneficiandose de las ventajas relevantes para su propósito. De las aplicaciones más importantes y que más protagonismo han adquirido estos últimos años podemos destacar:

- Smart Home o Casa Inteligente: Las aplicaciones en esta área son muy variadas desde controlar la temperatura de la vivienda a subir y bajar las persianas remotamente, pasando por frigoríficos inteligentes que nos avisan cuando se nos acaba algún alimento.
- Smart City o Ciudad Inteligente: Este concepto es muy simple, busca tener la mayor cantidad de datos sobre la ciudad para facilitar la vida del ciudadano. Desde Smart Parking que indica donde hay un hueco libre donde poder aparcar hasta el Smart Traficc que indica en tiempo real el tráfico en la ciudad y rutas alternativas.
- Agricultura inteligente: Monitoriza los cultivos, controlar automáticamente el riego, protección conta heladas, fertilización…
- Weareables: Son los dispositivos de consumo como las pulseras que nos monitorizan durante todo el día y nos dicen los pasos que hemos hecho y las calorías que hemos consumido, los relojes inteligentes que se conectan a nuestro smartphoe e incluso las camas inteligentes de los hospitales para observar signos vitales, presión sanguínea y temperatura corporal de los pacientes.

IoT utiliza dispositivos electrónicos que son capaces de medir magnitudes física o química y transformarlas en señales eléctricas como son los sensores y por otro lado utilizar esas señales eléctricas para activar un determinado proceso, como serían los actuadores. La combinación de estos dispositivos con la capacidad de conexión a internet constituiría el hardward de IoT.

Uno de los dispositivos inteligentes que están ganando más protagonismo son las *cerraduras inteligentes o de precintado remoto*, que nos aportan un sistema adicional para cerrar y abrir cualquier dispositivo o cosa como la puerta de nuestra casa, el garaje, una caja fuerte o los cofres de las motos de alquiler de nuestra ciudad.

Qué son

Las cerraduras inteligentes son aquellas que, como su propio nombre indica han sido diseñadas para ofrecer las funciones propias de una cerradura convencional pero que, además, añaden otro tipo de sistemas de acceso como pueden se huellas digitales, contraseñas o códigos PIN, mandos a distancia, etc. Cerraduras que además tienen la capacidad de conectarse a Internet y **ser controladas remotamente por otros dispositivos** como el propio smartphone.

Y es que, estos sistemas de precintado remoto se caracterizan por funcionar mediante una aplicación que se conecta directamente a nuestro teléfono móvil y desde donde podemos **abrir y cerrar el sistema a distancia.**

Aunque muchos piensan que han sido diseñadas para ofrecer una mayor seguridad a la hora de entrar en una casa, por ejemplo, son otros muchos los que piensan todo lo contrario, ya que exponen la seguridad de la casa a posibles ataques a través de Internet. Lo que sí está claro es que las cerraduras inteligentes nos facilitan el día a día, tal y como veremos a continuación. [2]

Cómo funcionan

Las cerraduras inteligentes pueden hacer uso de diferentes sistemas o métodos para la apertura y cierre de las puertas, por lo tanto, cada una tendrá un sistema o mecanismo distintos para funcionar. Sin embargo, el funcionamiento general es muy similar para todas ellas, ya que cuentan con un sistema eléctrico o electromagnético que se encarga de mantener la cerradura cerrada hasta que el controlador de acceso indique que se debe abrir.

En cuanto a seguridad, lo que hacen las cerraduras inteligentes, es gestionar un sistema de identificación para la persona que quiera abrirla. De esta manera, a través del sistema que incorpore cada cerradura, cada usuario deberá identificarse (o no) de manera correcta para que la puerta se abra o cierre.

Podríamos decir que se trata de un mecanismo similar al del bloqueo y desbloqueo de nuestros teléfonos móviles. Es decir, a través de un PIN o contraseña podemos desbloquear nuestro móvil o abrir nuestra puerta, mientras que también existen cerraduras que podemos abrir o cerrar con nuestra huella dactilar, por ejemplo. Estos sistemas de seguridad suelen ser bastante sofisticados para garantizar la seguridad de estas.

Además, el hecho de que puedan estar conectadas a Internet permite que podamos abrir o cerrar la puerta cuando queramos **de forma remota** desde otros dispositivos como el propio móvil.

Tipos de cerraduras inteligentes

Podemos clasificar estos dispositivos en diferentes tipos según el sistema que utilice para abrir o cerrar la puerta o según las tecnologías que intervienen.

- **Lectura de huella dactilar**: Es uno de los tipos de cerradura más avanzado, ya que cuenta con un lector de huellas dactilares para identificar a la persona que quiere abrir la puerta. Es el tipo de cerradura más utilizado para controlar el acceso a zonas restringidas como puede ser un banco. Lo primero que hay que hacer es registrar la huella de cada una de las personas que van a tener acceso y el sistema abrirá la puerta en el momento que reconozca cualquiera de ellas.
- **Con llave y contraseña**: Están diseñadas para aquellos que quieren seguir usando llaves para entrar en su casa, pero a su vez, quieren añadir seguridad extra. El funcionamiento es muy sencillo, es necesario meter la llave e introducir una contraseña para poder abrir la cerradura.
- **Con paneles táctiles y/o numéricos**: Estas cerraduras funcionan al introducir un pin o contraseña previamente configurados.
- **Con mando a distancia**: Este tipo cuentan con un sistema mecánico que se comunica vía WiFi con los dispositivos conectados a la red de nuestra casa. Es necesario instalar la app específica de la cerradura en nuestro móvil para poder realizar la apertura de la puerta. Suele usarse para abrir puertas de garaje.

- **Radiofrecuencia**: Son aquellas que nos permiten abrir la puerta con una tarjeta o un llavero haciendo uso de radiofrecuencia (RFID).
- **Bluetooth:** Funcionan con la conexión bluetooth del teléfono móvil. En el momento que acerquemos a la cerradura con el smartphone, la puerta se abre. Así mismo se puede configurar para que del mismo modo, se cierre cuando nos alejamos. Este tipo de cerraduras suelen contar con un sistema de apertura alternativo por si perdemos el móvil o nos quedamos sin batería antes de llegar a casa.
- **WiFi**: La conexión Wifi suele ser un complemento opcional en algunos modelos de cerraduras inteligentes. La posibilidad de usar esta tecnología nos permite conectar nuestra cerradura con otros dispositivos, como nuestros teléfonos móviles, y tener un control remoto de la cerradura desde cualquier lugar del mundo que tengamos conexión a Internet. En este caso, no es necesario ningún tipo de concentrador especial, ya que nuestro router de casa será el que proporcione a la cerradura la conexión a internet.

Ventajas y desventajas del uso de cerraduras inteligentes

El uso de las cerraduras inteligentes ofrece una serie de ventajas entre las que cabe destacar:

- Son muy cómodas de usar y fáciles de abrir, casi cualquier persona puede hacer uso de una cerradura inteligente, incluido niños o personas mayores.
- Podemos cerrar la puerta de una forma remota, aunque nos hayamos olvidado de hacerlo en el momento de salir de casa.
- Es posible abrir la cerradura a alguien o para algo cuando estamos lejos de casa.
- No requieren de una llave física para su apertura o cierre.
- Suelen permitir la programación de un horario para su apertura y cierre.
- No permite el forzado de la cerradura como ocurre con cerraduras convencionales.

Como cabe esperar, no todo son ventajas, ya que también podemos encontrar ciertos inconvenientes o desventajas a la hora de usar una cerradura inteligente.

- Cualquier problema con el mecanismo electrónico, las baterías o la electricidad, puede hacer que el sistema quede bloqueado o inactivo.
- Ofrecen menor fuerza mecánica a la hora de ser sometidas a cualquier golpe.
- Baja compatibilidad a la hora de instalarlas en ciertas puertas.
- Coste extra según el tipo de cerradura.
- Están expuestas a otras técnicas de hackeo para conseguir su apertura.

En el siguiente capítulo se realizará el estudio correspondiente al estado de las redes inalámbricas, en concreto de las de bajo consumo (LPWAN), centrándonos el modo que tienen de recibir la información de un servidor o de otros dispositivos.

3 ESTADO DEL ARTE DE LA RECEPCIÓN DE INFORMACIÓN EN SISTEMAS IOT DE BAJO CONSUMO

Como ya es sabido, la llegada del Internet de las Cosas (IoT) como evolución de las tecnologías inalámbricas, empieza resaltar el papel que juegan estas en sectores como el transporte, la industria, las comunicaciones e incluso la medicina. Pero esta oportunidad de introducirse en tales campos como la automatización de procesos, cuidades inteligentes o la conducciona autónoma no le llega gratuitamente, IoT tiene que enfrentarse a exigencias como la conexión con la nube, la seguidad, el mantenimiento de consumos de potencia bajos para que una batería pueda durar un periodo prolongado, la coexistencia de varias tecnologías con el fin de crear mejores aplicaciones IoT entre otras.

Esta gran cantidad de nuevas aplicaciones para los dispositivos inalámbricos que desarrollan IoT provoca una gran demanda de dispositivos con características hechas a la medida. Uno de los mayores problemas a los que se enfrenta el mercado de dispositivos IoT actualmente es la fragmentación, hay una gran multitud de dispositivos y protocolos que hacen difícil poder construir sistemas uniformes y funcionales si decididmos utilizar componentes de diferentes tecnologías. Por ello, la especificidad del dispositivo que se quiere diseñar obliga a los diseñadores a seleccionar la tecnología de comunicación inlambrica que más se adecue a los requisitos que necesitamos. De los más importantes que hay que tener en cuenta son, entre otros, la duración de la batería, el rango de comunicación y la cantidad de datos tranferidos.

Se ha creído conveniente en este estado del arte dar primero una visión sobre la clasificación que existe de las redes inalámbricas y después centrarnos, concretamente en las redes LPWAN y en tres ejemplos de ellas. Estas cuatro redes serán LTE-M, LoRaWAN, NB-IoT y Sigfox, y trataremos su forma de enviar y recibir la información desde su posición de tecnologías IoT de bajo consumo.

Comencemos con la clasificación de las redes inalámbricas según su cobertura: [3]

1. **WPAN (*Wireless Personal Area Network*):** Red Inalámbrica de Área Local. Es un tipo de red con cobertura personal. Bluetooth es un ejemplo de WPAN. Tiene un alcance pequeño, de unos 10 metros, por lo que la finalidad de estas redes es comunicar un dispositivo personal con sus periféricos o entre dos dispositivos permitiendo una comunicación a corta distancia entre ellos.
2. **WLAN (*Wireless Local Area Network*):** Red Inalámbrica de Área Local. En este tipo de red se encuentran las tecnologías basadas en Wi-Fi. Proporciona mayor alcance, de hasta 20 km si se usan repetidores.
3. **WMAN (*Wireless Metropolitan Area Network*):** Red Inalámbrica de Área Metropolitana. Son redes más complejas y con mayor alcance.
4. **WWAN (*Wireless Wide Area Network*):** Red Inalámbrica de Área Amplia. Necesita de una tecnología especial, red celular, para dar su servicio más eficientemente. Proporciona acceso a internet a través de una red de telefonía móvil. Tiene las ventajas de la banda ancha normal (ADSL) y las de la movilidad inalámbrica. Aquí se encontrarían 3G, 4G y 5G.

Y según el rango de frecuencias que utilicen para trasmitir:

1. Infrarrojos: Los datos se transmiten mediante ondas infrarrojas. Pueden alcanzar frecuencias de entre 300 GHz y 400 THz, pero los dispositivos deben estar muy cerca y alineados. Además, no pueden atravesar paredes.

2. Mircoondas: Aquí encontramos dos tipos. Las terrestres, que usan antenas parabólicas para la transmisión con una cobertura de kilómetros y un rango de frecuencia de 1 a 300 GHz. Y, por otro lado, las satelitales, que usan un satélite para enlazar varias estaciones y enviar señales con un alcance y velocidad mucho mayor que las terrestres.
3. Ondas de radio: Las señales se reciben y se emiten mediante ondas de diferentes frecuencias (AM, FM, TV…). Estas reducen su capacidad a medida que aumenta la distancia.

A la hora de elegir un tipo de red inalámbrica estas son las características diferenciadoras que habría que tener en cuenta:

- Niveles OSI de aplicación.
- Uso de espectro licenciado o no licenciado.
- Banda de frecuencia utilizada.
- BW de señal. (Determina la tasa de envío de datos)
- Modulación.
- Mecanismo de acceso al medio.
- Topología de red.
- Máxima tasa de transmisión.
- Latencia de las comunicaciones.
- Uso de energía.
- Coste.
- Seguridad.

Dentro de las tecnologías inalámbricas, existen también las denominadas **LPWAN** (*Low Power Wide Area Network*), en español, Red de Bajo Consumo y Área Extensa. Este tipo de tecnología permite transmitir poco volumen de datos entre un dispositivo y una estación base/getaway separados a kilómetros de distancia con muy bajo consumo energético.

Por sus características, esta tecnología esta haciendo posible el desarrollo de la gran mayoría de iniciativas IoT de hoy en día. Al ser diseñada para este entorno, permite instalar decenas o centenares de nodos distribuidos por un área muy extensa, alimentadas con baterías que duran años (hasta más de 10) y sin necesidad de grandes y costosas infraestructuras cableadas.

Las tecnologías de transmisión inalámbrica tienen en cuenta tres pilares, por así decirlo, fundamentales. Consumo energético, alcance o distancia entre nodo y antena, y por último capacidad de transmisión. No existe ninguna que cubra muy bien los tres por lo que depende de cual sacrifica uno de ellos para potenciar los demás. En este caso, las LPWAN, para tener un muy buen alcance y un bajo consumo sacrifican el numero de datos que pueden transmitir. [4]

Aunque los sistemas de IoT tienen muchas arquitecturas diferentes, la mayoría incluye los siguientes componentes para su comunicación:

1. **Dispositivo IoT**: un dispositivo IoT es, posiblemente, cualquier aparato electrónico que pueda comunicarse con Internet. Normalmente, nos centramos en dispositivos con una finalidad más limitada como un sensor para detectar sucesos físicos (movimiento, temperatura, tensión…) o actuadores que creen cambios físicos (como encender una luz o cerrar una puerta). Como dispositivo conectado, debe de tener al menos un elemento de comunicación (radio o método de comunicación por cable). Además, suelen funcionar con baterías por lo que gestionar esta característica es clave.
2. **Comunicaciones locales**: Los métodos y protocolos que utiliza el dispositivo para hablar con los dispositivos vecinos. Algunos sólo envían información y muchos otros envían y reciben.

Algunas comunicaciones con dispositivos similares son directas, pero en cambio las remotas a veces deben de pasar por una pasarela para llegar a su destino.

Un modelo de comunicación inalámbrica podría ser la "red estrella", en el que un módulo inalámbrico inteligente coordina las comunicaciones hacia los dispositivos que actúan como routers y éstos trasladan las comunicaciones a los dispositivos finales.

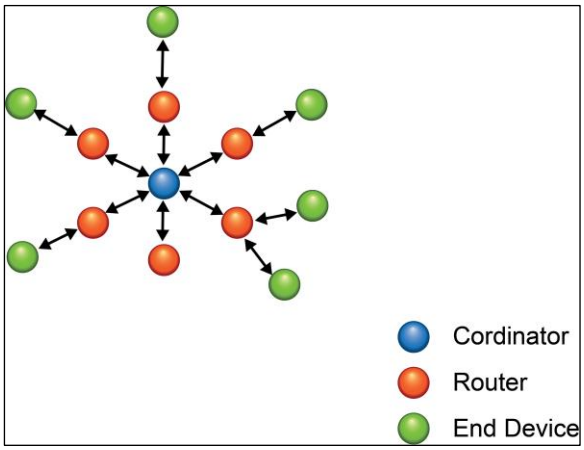

Figura 2. Ejemplo de "red estrella". [4]

3. **Protocolo de aplicación**: define cómo se transporta el contenido de la información.
4. **Pasarelas**: traducen y transmiten la información, normalmente enlazando las redes de dispositivos locales con internet. Estas pasarelas ayudan a los dispositivos IoT a que siguan siendo pequeños, funcionen con baterías y sean baratos. Por ejemplo, los dispositivos portátiles con Bluetooth suelen usar un teléfono móvil como puerta de entrada a internet.
5. **Servidores de red**: sistemas que gestionen la aceptación y la transmisión de los datos del IoT.
6. **Aplicaciones en la nube**: procesan los datos del IoT para convertirlos en información útil y presentarla a los usuarios.
7. **Interfaz de usuario**: donde el usuario ve la información de IoT, la manipula y emite órdenes a los dispositivos del IoT.

LTE-M

LTE-M es una tecnología de área amplia y de baja potencia (LPWAN) que permite que los dispositivos IoT puedan conectarse directamente a una red 4G, sin necesidad de una puerta de enlace. Por así decirlo, utiliza la autopista 4G ya implantada y dedica un carril exclusivamente a comunicaciones IoT. Al mismo tiempo, permite la reutilización de las instalaciones de LTE. Así se obtiene una mayor vida útil de la batería.

Esta tecnología requiere transmitir pequeñas cantidades de datos en largos periodos de tiempo, con bajo consumo de energía. Posee un ancho de banda mayor, lo cual favorece mayor velocidad de datos, menor latencia y mejor posicionamiento para aplicaciones de movilidad. Como añadido, permite envíos por voz.

El estándar de tecnología LTE-M, desarrollado por 3GPP, coexisten con todas las redes móviles 2G, 3G y 4G, además de beneficiarse de todas las características de seguridad y privacidad de las redes móviles como pueden ser el soporte para confidencialidad de la identidad del usuario.

NB-IoT

NB-IoT o *Narrow Band*, aparece de la mano de G3PP durante el auge de las LPWAN. Esta tecnología reutiliza varios principio y bloques de la capa física de LTE y esta diseñada para dar mas cobertura en comparación con las redes GSM tradicionales, además de mejorar la capacidad de la red en zonas de mala cobertura mediante tranmisiones de un único tono. Por esto NB-IoT puede coexistir con LTE y en su modo de operación autónomo con GSM.

Hay que destacar que NB-IoT es una tecnología que nos da fiabilidad porque garantiza la entrega de los datos. También decir, que el uso de una banda licenciada la hace más fiable puesto que no coexiste con otras tecnologías. Sin embargo, el hecho de usar una banda licenciada tiene implicaciones como el hecho de depender de un operador y por lo tanto estar sujetos a un modelo de servicio y cobertura fuera del control de la aplicación. Esto también tiene implicaciones en el consumo energético del dispositivo. Los nodos con peor cobertura gastarán más energía, porque deberán repetir mas las tramas y porque deberán transmitir a una potencia mayor. Finalmente, destaca el buen soporte al downlink de NB-IoT, habilita de forma sencilla aplicaciones que necesiten modelos push a diferencia de los modelos uplink tradicionales que ofrecen la mayoría de LPWANs.

NB-IoT permite la conexión de un numero masivo de dispositivos IoT y esta pensada para equipos fijos con bajos volúmenes de transferencia de datos y bajo consumo de energía, ya que utiliza poco ancho de banda. Es muy útil para ciertas aplicaciones en sectores como el agrícola y la industria por el bajo consumo de potencia de sus transceptores, que no consumen energía a lo largo del día hasta que existe la necesidad de hacer una tranmisión. Según sus desarrolladores, la duración de la batería de un dispositivo NB-IoT es de hasta 10 años cuando se transmiten 200 bytes al día de media. La velocidad de tranmision de datos entre los elementos de la red es de 200kbpd en dowlink y 20 kbps en uplink siendo el tamaño máximo de la caga útil de cada mensaje de 1600 bytes. Por último, tiene un alcance de hasta 3km en entornos urbanos. [5]

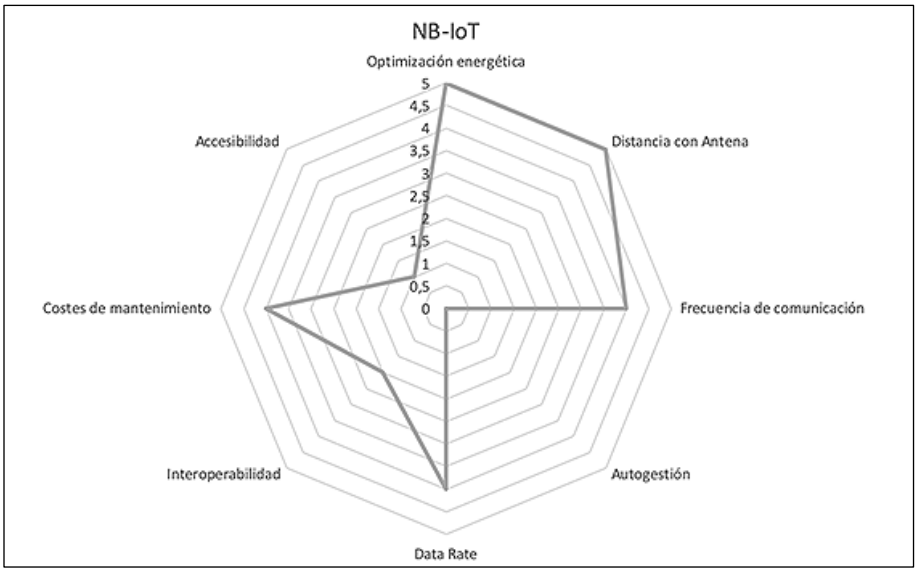

Figura 3. Gráfica comparativa características NB-IoT. [6]

LoRaWAN

LoRaWAN es, como indicamos anteriormente, otra de las llamadas LPWAN. Esta basada en LoRa, de LoRaAlliance, y utiliza una técnica derivada de la modulación Chirp Spread Spectrum (CSS). Esto hace que mantenga las mismas características de baja potencia, pero mejorando significativamente el alcance de la comunicación. Utiliza todo el ancho de banda del canal para enviar la señal, haciéndola resistente al ruido. Hay que destacar que LoRaWAN, utiliza un espectro sin licencia para comunicarse con la red, lo que hace que funcione bien en áreas rurales o remotas sin cobertura 4G. También funciona bien cuando está en movimiento (en un camión, avión o barco).

La velocidad de Lora oscila entre 300 bps y 50 kbps, dependiendo del factor de dispersión y del ancho de banda del canal que se use. Además, las estaciones base de LoRa pueden recibir simultáneamente mensajes transmitidos con diferentes factores de propagación y la comunicación bidireccional la proporciona la modulación de espectro disperso chirp (CSS) mencionado anteriormente.

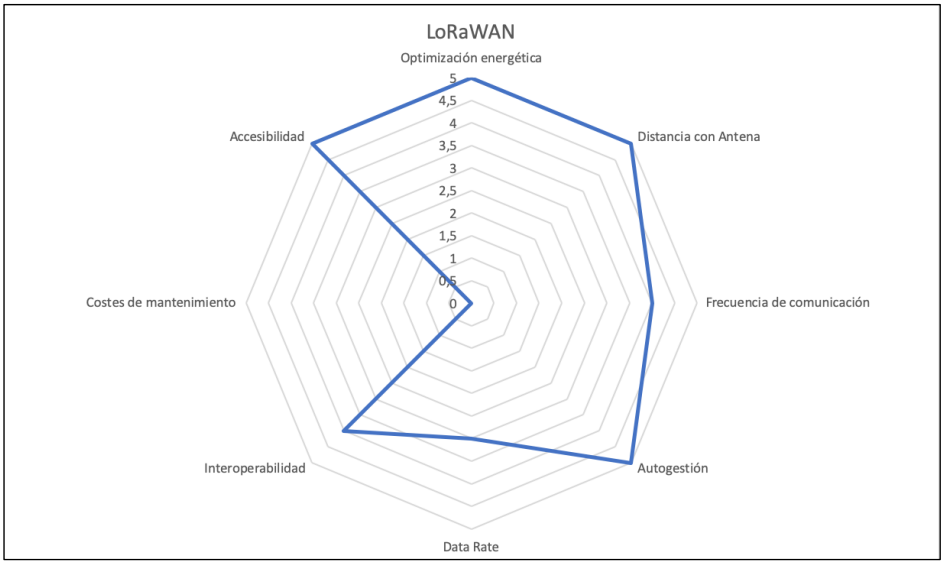

Figura 4. Gráfica comparativa características LoRaWAN. [6]

LoRaWAN presenta una topología de estrella con las getaways enlazando los nodos finales con un servidor. La comunicación, como hemos mencionado es bidireccional, si bien lo habitual es que el nodo final mande datos al servidor (*uplink*). Este puede transmitir a cualquier frecuencia disponible a la tasa de datos requerida, pero respetando las reglas de ciclo de trabajo máximo, la duración de la transmisión y los cambios aleatorios de canal que hace para mejorar la robustez ante interferencias.

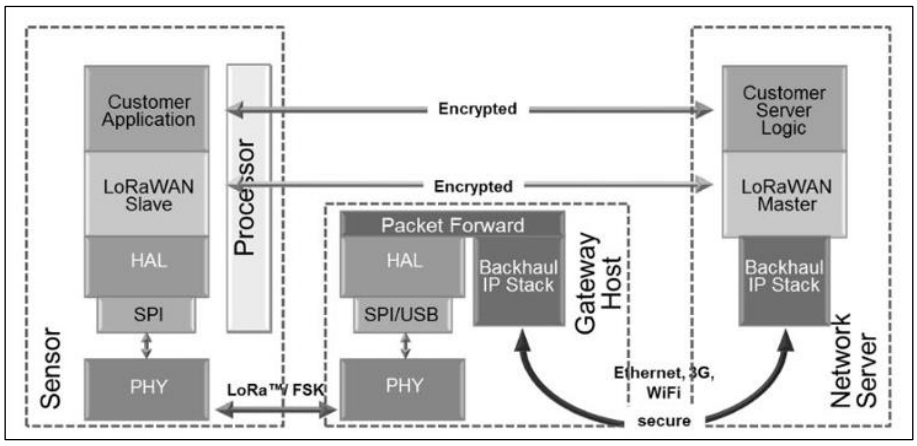

Figura 5. Arquitectura detallada de LoRaWAN. [7]

Mensajes de uplink: son los que se envían desde el nodo al servidor. Constan de un preámbulo, encabezado y payload, y finalmente un CRC para asegurar la transmisión.

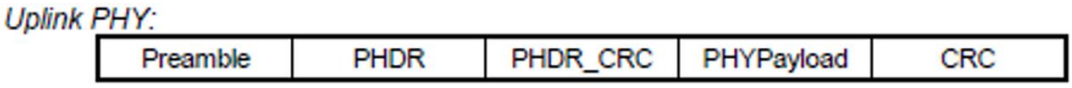

Figura 6. Formato mensaje *uplink* LoRaWAN. [8]

Mensajes de downlink: son los que se envían desde el servidor a un nodo a través de una única getaway. Contiene preámbulo, encabezado y payload, pero solo tienen CRC en el encabezado.

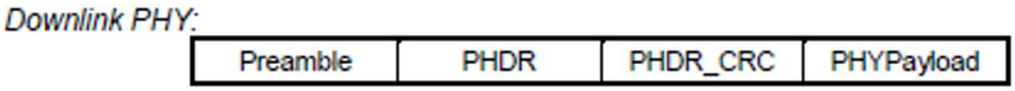

Figura 7. Formato mensaje *downlink* LoRaWAN. [8]

Los dispositivos finales de la red no tienen porqué tener las mismas características y no se van a utilizar con el mismo propósito ni para las mismas aplicaciones, por eso LoRaWAN ha diseñado diferentes clases de dispositivos:

- **Dispositivos de clase A:** los dispositivos finales de clase A permiten comunicaciones bidireccionales, donde una transmisión viene seguida de dos ventanas para la recepción. La primera (RX1) se abre después de un delay de +- 20 microsegundos después de acabar el uplink y la segunda (RX2), +- 20 microsegundos después de crearse RX1. RX1 usa el mismo canal de frecuencia que el ultimo mensaje de uplink y RX2 se abre en un canal configurable con comandos MAC. Estas ventanas deben de durar como mínimo lo suficiente para detectar el preámbulo de un mensaje de downlink y, si detecta uno se queda esperando hasta recibir toda la trama. Si se interpreta la trama en RX1 no se abre RX2. Este tipo de operación de clase A es la que menor energía consume ya que solo espera una comunicación del servidor si el dispositivo ha transmitido antes.

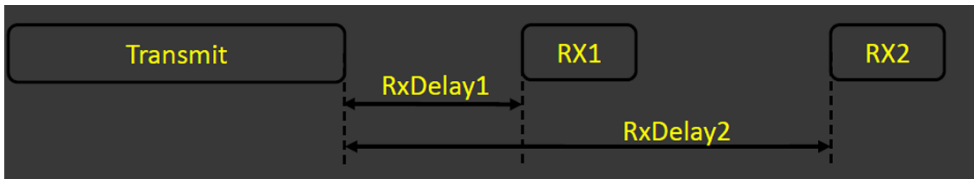

Figura 8. Transmision LoRaWAN clase A. [8]

- **Dispositivos de clase B:** los dispositivos finales de clase B permiten la recepción de datos sin la necesidad de transmitir con anterioridad, lo que se consigue abriendo ventanas de recepción extra en tiempos programados. Para que el dispositivo abra esta ventana en el tiempo programado, este recibe una bandera sincronizada en el tiempo del Getaway, indicándole que lo haga. Esto permite que el servidor sepa en qué momento el dispositivo final está escuchando. Como consecuencia de esto, estos dispositivos consumen más energía que los de clase A.

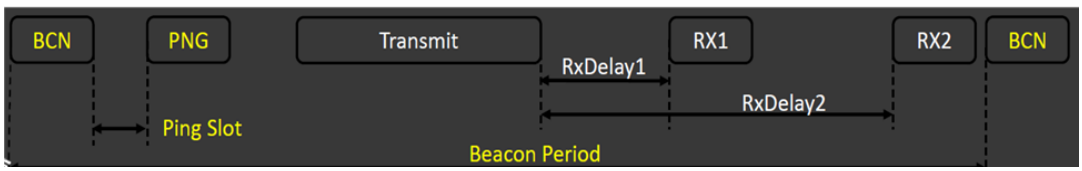

Figura 9. Transmision LoRaWAN clase B. [8]

- **Dispositivos de clase C:** estos dispositivos se encuentran permanentemente escuchando, es decir, sus ventanas de recepción están siempre abiertas, excepto que el dispositivo esté transmitiendo. Esta clase proporciona los mejores tiempos de respuestas y capacidad de envío desde el servidor a los nodos, pero en consecuencia consume mucha más energía que los de las otras dos clases.

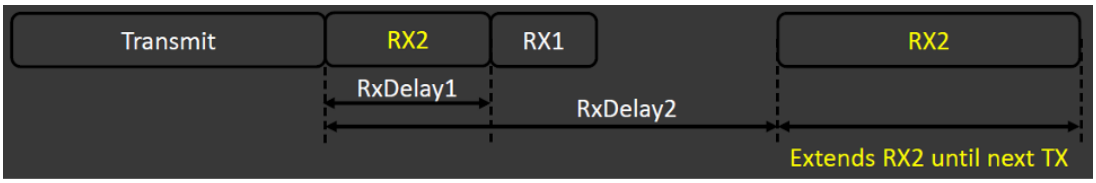

Figura 10. Transmision LoRaWAN clase C. [8]

Hasta el momento LoRa ha sido el estándar líder en LPWAN del sector público y ha sido uno de los mayores impulsadores de las Smart cities en varios países del mundo, desplegando redes para las interconexion es M2M (Parquímetros, Señales de tránsito etc…) sin necesidad de usar bases para controlar el tráfico de la información.

Sigfox

Sigfox es otro de los operadores de redes LPWAN, pensado para tener bajo consumo y ofrecer una solución de conectividad IoT de extremo a extremo basada en su propia tecnología. Cuenta con sus propias redes desplegadas y las conecta a los servidores backend, y presta así, servicio a las compañías que así lo requieran.

Utiliza 200kHz de las bandas ISM sin licencia, como LoRaWAN, para las tranmisiones de datos. Usa el ancho de manda de una forma muy eficiente y sufre niveles de ruido muy bajos, lo que permite que el consumo de energía sea muy bajo, la sesibilidad del receptor sea alta y el diseño de la antena sea de bajo coste.

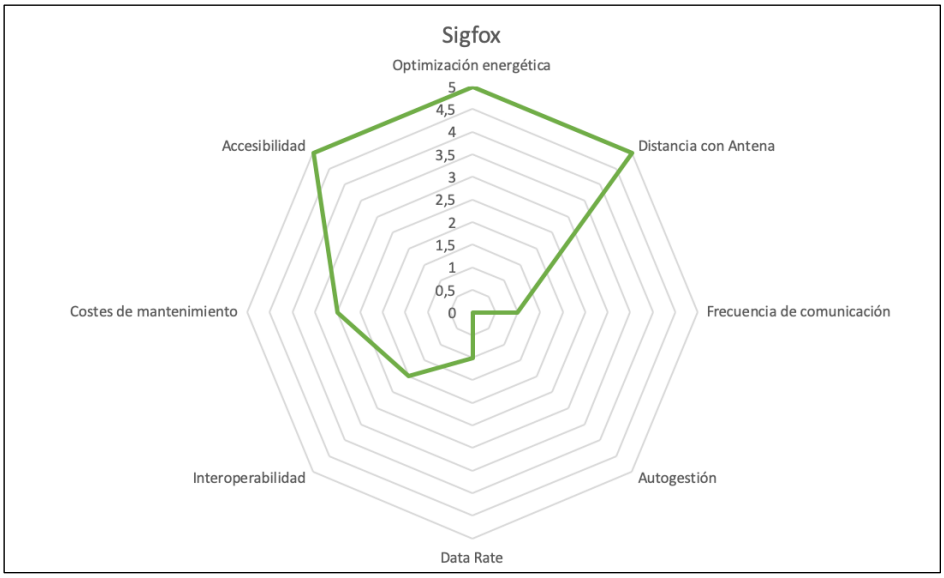

Figura 11. Gráfica comparativa características Sigfox. [6]

Los sistemas inalámbricos de Sigfox envían una cantidad muy pequeña de datos (12 bytes de carga útil en mensajes *uplink* y 8 bytes de carga útil en mensajes *downlink*) a baja velocidad (100 bit/s). Los mensajes escendentes (*uplink*) permiten enviar la información recogida por los dispositivos al backend y los

descendentes (*downlink*) sirven para configurarlos y enviarles mensajes de control. Las características de los tranmisiones de Sigfox son las siguientes:

- La transmisión entre los dispositivos y la red es asíncrona.
- Un dispositivo puede tranmisitir a otro siempre y cuando estén en la misma banda de frecuencia.
- Cada dispositivo emite los mensajes tres veces, usando tres frecuencias distintas de la misma banda. Lo que garantiza la fiabilidad de la comunicación.
- La demodulación de señales se lleva a cabo por las estaciones base.

Asimismo, Sigfox presenta cuatro modalidades de transmisión:

- One: de 1 a 2 mensajes al día sin descarga.
- Silver: de 3 a 50 mensajes al día con 1 enlace de descarga.
- Gold: de 51 a 100 mensajes al día con 2 enlaces de descarga.
- Platinum: de 101 a 140 mensajes al día con 4 enlaces de descarga.

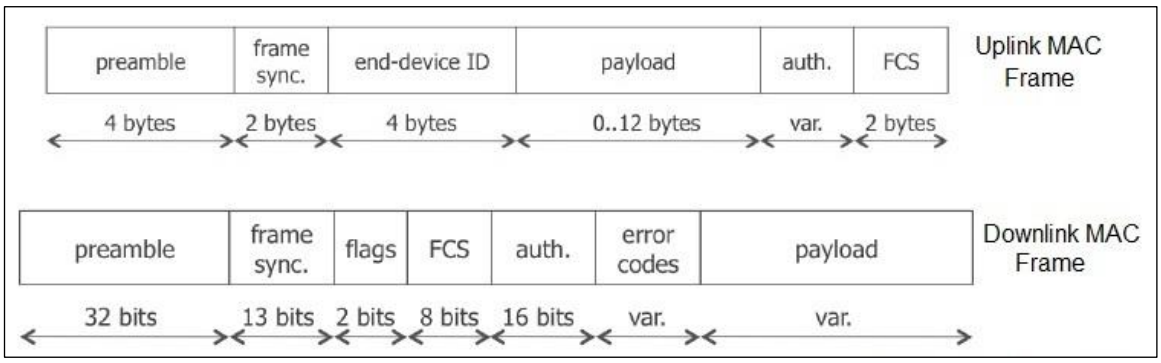

Figura 12. Formato mensajes *uplink* y *downlink* de Sigfox. [9]

Inicialmente, Sigfox solo admitía la comunicación *uplink*, pero ha evolucionado a una tecnología bidireccional. El inconveniente, que es comunicación *downlink*, es decir, los datos son enviados desde el backend, está optimizada para minimizar el consumo de energía, por lo que no se pueden transmitir mensajes en cualquier momento. Solo se recibe una respuesta downlink si se ha producido primero una comunicación uplink. El modulo manda un mensaje al servidor de Sigfox que contenga un *uplink request flag* solicitando un mensaje de downlink. Este mensaje se tranmite a la nube donde se pasa el mensaje a la plataforma del cliente que decide si transmitir un mensaje o no de downlink. Si decide transmitirlo, lo envia a la nube y esta selecciona una estación para que lo transmita al objeto en cuestión.

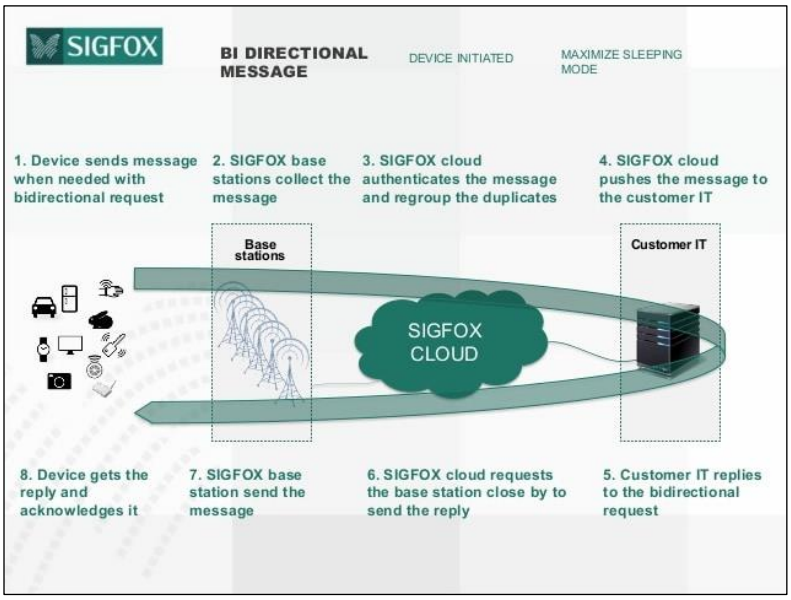

Figura 13. Esquema funcionamiento de la bidereccionalidad de la comunicación Sigfox. [9]

Selección de la tecnología para la transmisión de datos

	LTE-M	**NB-IoT**	**LoRaWAN**	**Sigfox**
Ancho de banda	1. MHz	180 kHz	600 kHz	100 Hz
Bandas de frecuanencias tipicas	450 MHz - 3.5 GHz	450 MHz - 3.5 GHz	Por debajo de 1 GHz	Por debajo de 1 GHz
Máxima tasa de datos descendente	1 Mbps	250 kbps	50 kbps	100 bps
Maxima tasa de datos ascendente	1 Mbps	250 kbps	50 kbps	100 bps
Duración de la batería	Hasta 10 años	Hasta 10 años	Hasta 10 años	
Coste típico de los módulos	Medio	Bajo	Muy Bajo	Muy bajo
Protección de la identidad	Si	Si	Parcial (dirección del dispositivo)	No
Confidencialidad de datos	Si	Si	Si	No
Bidireccionalidad	Si	Si	Si	Limitada
Movilidad	Si	No	Si	
Alcance		2km (entorno urbano), 15 km (entorno rural)	Hasta 5 km en entorno urbano y hasta 20 km en entorno rural.	40 km (ambientes rurales), 10 km (ambientes urbanos)
Latencia	Baja.	Muy baja. De 1,5 a 10 segundos. (Más frecuente la transferencia de datos)	Alta.	

Tabla 1. Comparativa entre las tecnologías LTE-M, NB-IoT, LoRaWAN y Sigfox.

En un primer momento se planteó el proyecto utilizando la tecnología Sigfox, ya que se pretendía validar la posibilidad de enviar mensajes *downlink* como continuación del TFG de otra compañera. La

imposibilidad de ello nos llevo a utilizar la tecnología NB-IoT, la cual nos ofrecía las mejores prestaciones para este tipo de sistmas.

La principal característica que nos hizo decantaranos por esta tecnología fue que, como ya hemos dicho, si nos permitía enviar mensajes de downlink, lo cual es imprescindible para el desarrollo de este proyecto, además de su bajo coste.

Aunque tiene un alcance menor que la demás, proporciona una buena tasa de datos tanto ascendente como descendente, y una latencia muy baja. Además, es una tecnología conocida y usada por el GIE, que nos ha permitido tener mucha información sobre ella y proporcionado un driver para su implementación en el proyecto.

A continuación, se detallan los módulos que se han usado para llevar a cabo la implementación del proyecto, la conexión entre ellos, la funcionalidad que cada uno desemperña y sus características.

4 HARDWARE

Las cosas no se hacen siguiendo caminos distintos para que no sean iguales, sino para que sean mejores.

- Elon Musk -.

Los elementos Hardware usados en este proyecyo son 4 el actuador solenoide *DSML-0224-12*, la placa Núcleo-L15RE, el FT4232H-56Q Mini Module, que convierte la lógica TTL a UART y el módulo de NBIoT. Por ello, consideramos oportuno profundizar en las características de cada uno de ellos, sus conexiones y la función que desempeñan en este proyecto.

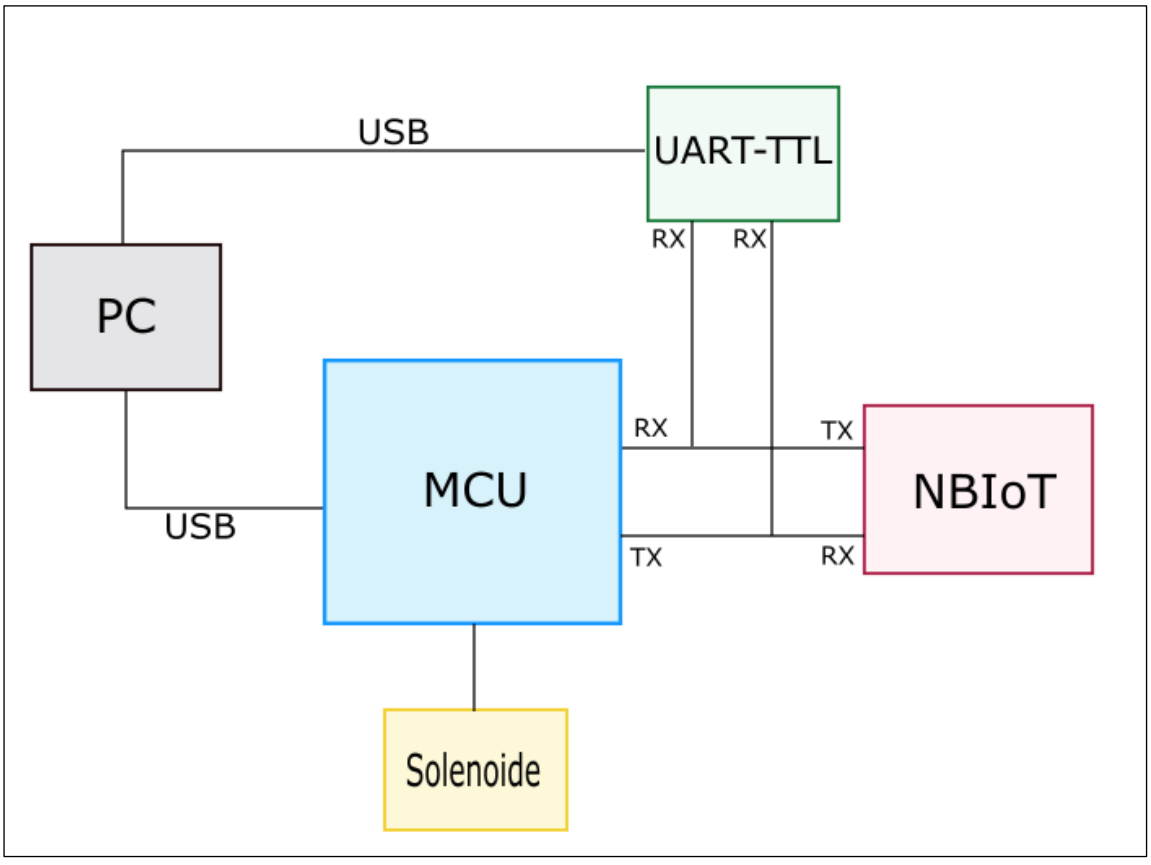

Figura 14. Diagrama de la implementación general del proyecto.

4.1. DSML-0224-12

Hemos realizado una búsqueda de distintos actuadores solenoides en el proveedor *Mouser Electronics.* Vamos a hacer una breve comparación entre tres de ellos.

Nombre	*DSML-0224-12*	*DSOL-0844-05*	*DSOL-0844-12*
Consumo	1 A	3,85 A	1,55 A
Voltaje	12 V	5 V	12 V
Potencia	12 W	20 W	20 W
Peso	5,5 g	95 g	95 g
Duty Cycle	0,2%, 20mseg On	100 %	100 %
Precio aprox	7,41 €	13,92 €	13,92 €

Figura 2. Tabla comparativa entre modelos de actuadores solenoides.

Teniendo en cuenta la tabla anterior, el actuador elegido ha sido el *DSML-0224-12.* Por su tamaño y consumo es el mád indicado para este proyecto, además de ser el más barato.

Ilustración 1. Actuador DSML-0224-12. [10]

El DSML-0224-12 es un actuador de tipo solenoico que podemos encontrar en el mercado con varios voltajes de alimentación. Siempre actúan con el mismo duty cycle, solo varía su consumo y alimentación.

En nuestro caso nos hemos quedado con el correspondiente a 12V. Lo ideal hubiera sido disponer de el de 5V y que compartiera tensión con la placa núcleo, pero no se disponía de existencia. Como el objetivo del proyecto es demostrar la viabilidad tecnologíca se ha optado por el de 12 V alimentándolo con un batería externa de 12V.

PART No.	VOLTAGE	DUTY CYCLE	POWER (W)	CURRENT (A)	DCR (Ω) *
DSML-0224-24	24 VDC	0.2%, 20 msec On.	13	0.54	44
DSML-0224-18	18 VDC	0.2%, 20 msec On.	13	0.73	24.6
DSML-0224-12	12 VDC	0.2%, 20 msec On.	12	1.0	12
DSML-0224-09	9 VDC	0.2%, 20 msec On.	11.2	1.22	7.4
DSML-0224-05	5 VDC	0.2%, 20 msec On.	13	2.62	1.9

Tabla 3. Tabla modelos actuador DSOL-0844. [10]

Vamos a mostrar a continuación una serie de características del actuador.

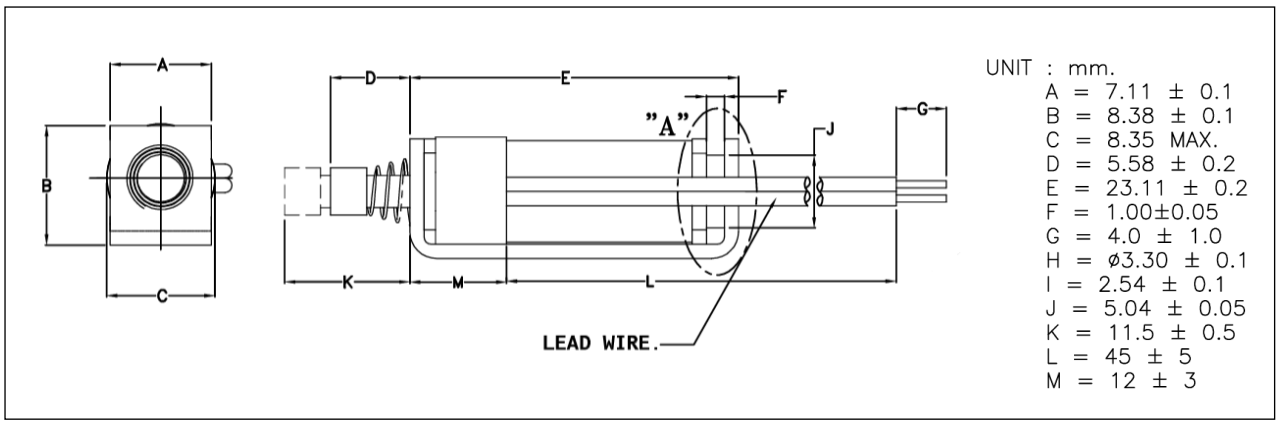

Figura 15. Dimensiones mecánicas del actuador. [10]

Nota 1: el datasheet especifica que el actuador tiene una esperanza de vida de unos 100.000 ciclos de operación.

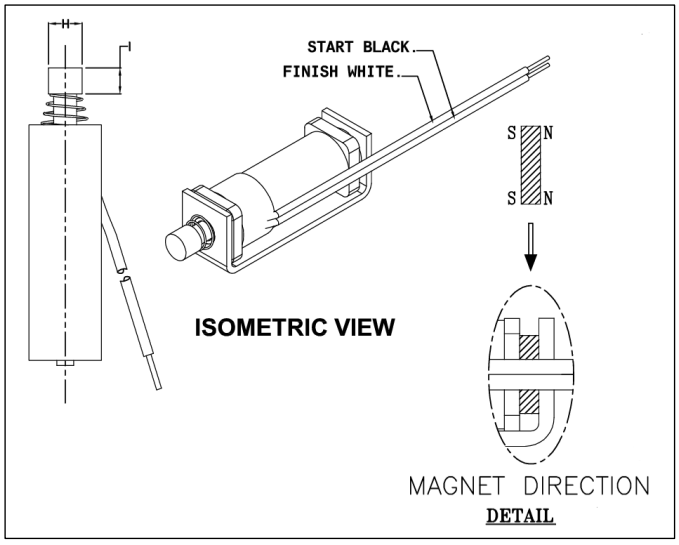

Figura 16. Vista isométrica del actuador. [10]

Que el duty cycle sea del 0.2% significa que el pulso está alto 20 milisegundos. En este caso, cuando se alimente, empujará el embolo hacia fuera 20 milisegundos y se desativará del tiempo hasta completar un ciclo. (20 ms On, 40 ms OFF). Para devolverlo a su posición inicial hay que hacerlo manualmente.

A continuación, se muestra una gráfica que muestra la relación que existe entre la fuerza que ejerce el actuador con la distancia que ha recorrido el émbolo. Esto viene explicado en el primer punto del *Anexo* que encontraremos al final de esta memoria, que nos viene a decir que la fuerza que ejerce el actuador solenoide no es lineal, es decir, ejerce menos fuerza cuando el émbolo está extendido, que cuando está cerca de su posición final, como bien se puede apreciar en la grafica.

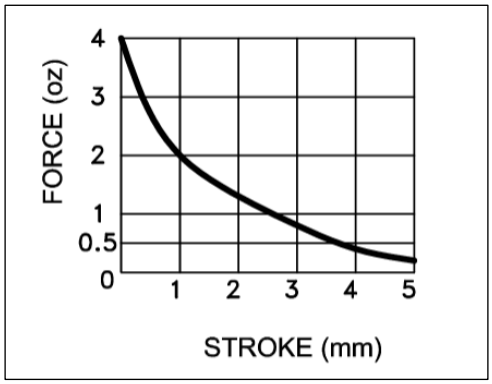

Figura 17. Gráfica Fuerza vs Carrera (distancia recorrida por el émbolo). [10]

4.2. Placa Núcleo STM32

El segundo elemento de este proyecto, y el principal, sin el este que no sería posible, es la placa de STM, NUCLEO-L152RE. Es la que posee el microcontrolador (STM321L52RE) que vamos a programar para controlar el actuador.

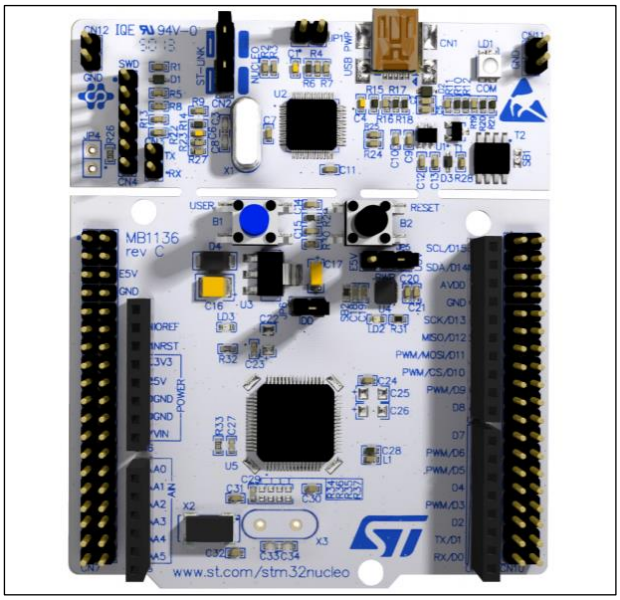

Ilustración 2. Placa STM32 Nucleo-L152RE. [11]

La placa ofrece las siguientes características:

- Microcontrolador STM32L152RET6 en encapsulado LQFP64.
- Tres LED:
  - Comunicación USB (LD1), LED de usuario (LD2), LED de alimentación (LD3).
- Dos pulsadores: USER y RESET.
- Dos tipos de conectores de extensión:
  - Conectividad ARDUINO® Uno V3.
  - Cabeceras de pin de extensión ST morpho para acceso completo a todas las E/S STM32.
- Fuente de alimentación:
  - USB VBUS o fuente externa (3.3 V, 5 V, 7-12 V).
  - Punto de acceso de administración de energía.
- Programador y depurador ST-LINK / V2-1 integrado con conector SWD:
  - Interruptor de modo de selección utilizando el kit como ST-LINK / V2-1 independiente.
- Tres interfaces diferentes compatibles con USB:
  - Puerto COM virtual.
  - Almacenamiento masivo.
  - Puerto de depuración.
- Amplias bibliotecas de software gratuito y ejemplos disponibles con STM32Cube

  Paquete MCU.

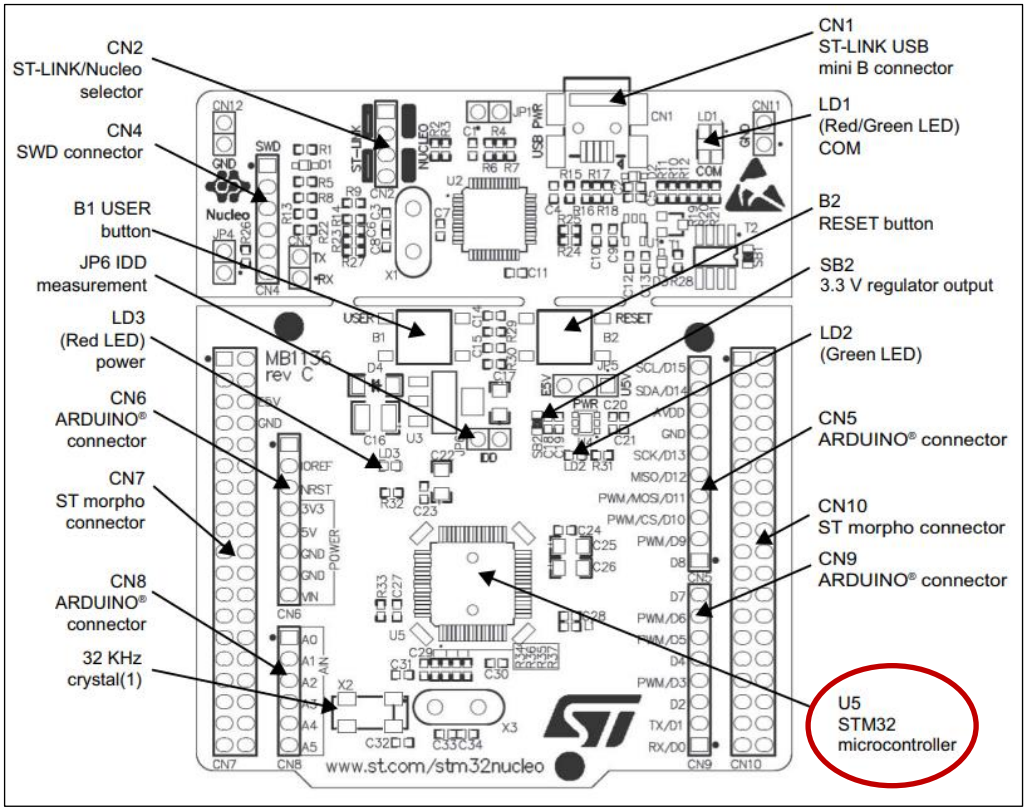

Figura 18. Layout del Top STM32 Nucleo- L152RE. [12]

Asimismo, del microcontrolador STM32152RE, el cual nos interesa de esta placa, tiene como características destacables:

- Consumo ultra-bajo con alimentación entre 1.65 V y 3.6 V. (Varios modos de bajo consumo).
- Núcleo: Core Arm Cortex-M3 de 32-bit. Desde 32 kHz hasta 32 MHz máx. 1.25 DMIPS/MHz y unidad de protección de memoria.
- Hasta 34 canales de detección capacitiva.
- Unidad de cálculo del CRC.
- Hasta 116 puertos de E/S, todos asignables en 16 vectores de interrupción externos.
- Seis fuentes de reloj.
- Tres resets: Browout reset, POR (power-on reset), PDR (power-down reset). Y un detector de tensión programable.
- 512 Kbytes de memoria Flash, 80 Kbytes de RAM, 16 Kbytes de EEPROM y un registro de respaldo de 128 bytes.
- Once interfaces de comunicación: 1 USB, 5 USARTs, 8 SPIs Y 2 I2Cs.
- Un driver LCD.
- Dos amplificadores operacionales, un ADC y DAC de 12-bit y dos comparadores de potencia ultrabaja.

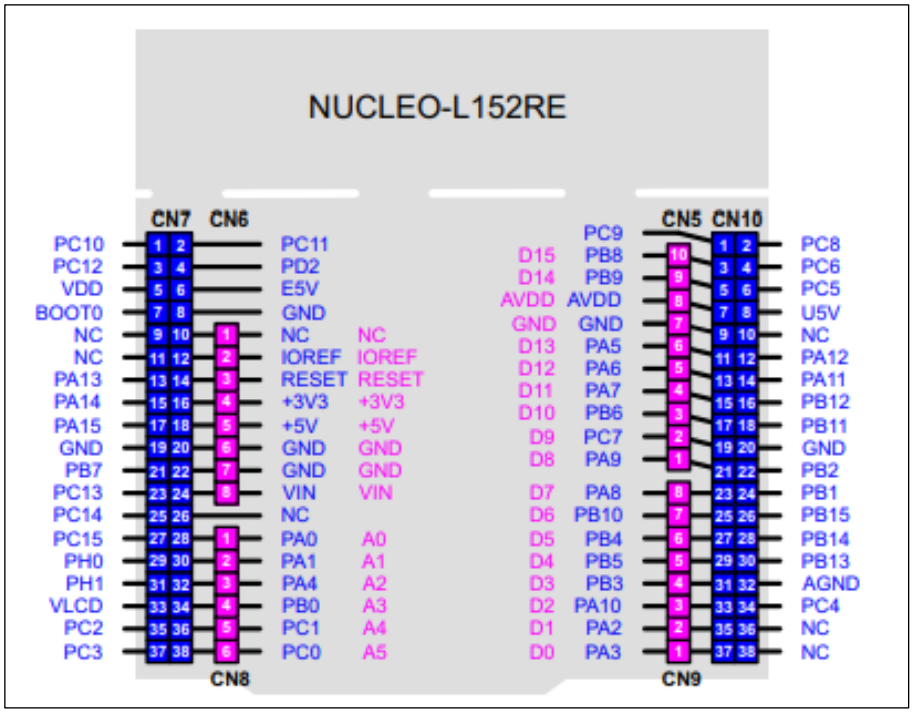

Figura 19. Pinout del Núcleo-L152RE. [12]

En las figuras mostradas a continuación se observan los canales 8 y 9, compatibles con Arduino, y los canales 7 y 10, compatibles con Morpho. Todos ellos con sus respectivos pines y las funcionalidades que soportan cada uno de ellos.

Para nuestro proyecto son de interés los canales CN6 y CN10, el resto, se pueden consultar en el manual de la placa Núcleo-L152RE. [12]

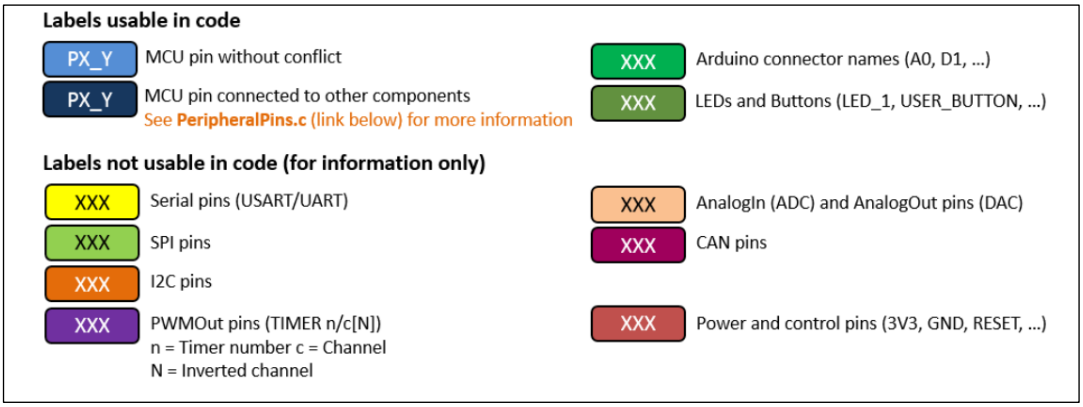

Figura 20. Leyenda de pines de la placa. [11]

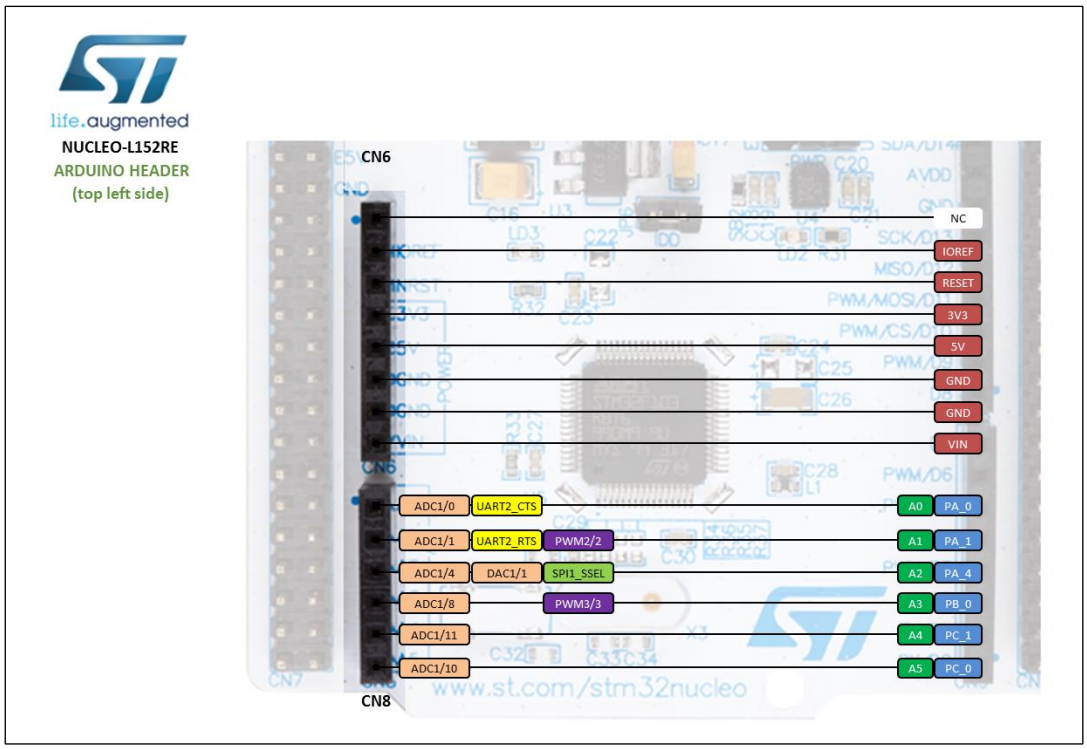

Figura 21. Pin Out CN6/CN8 placa STM32L152RE. [11]

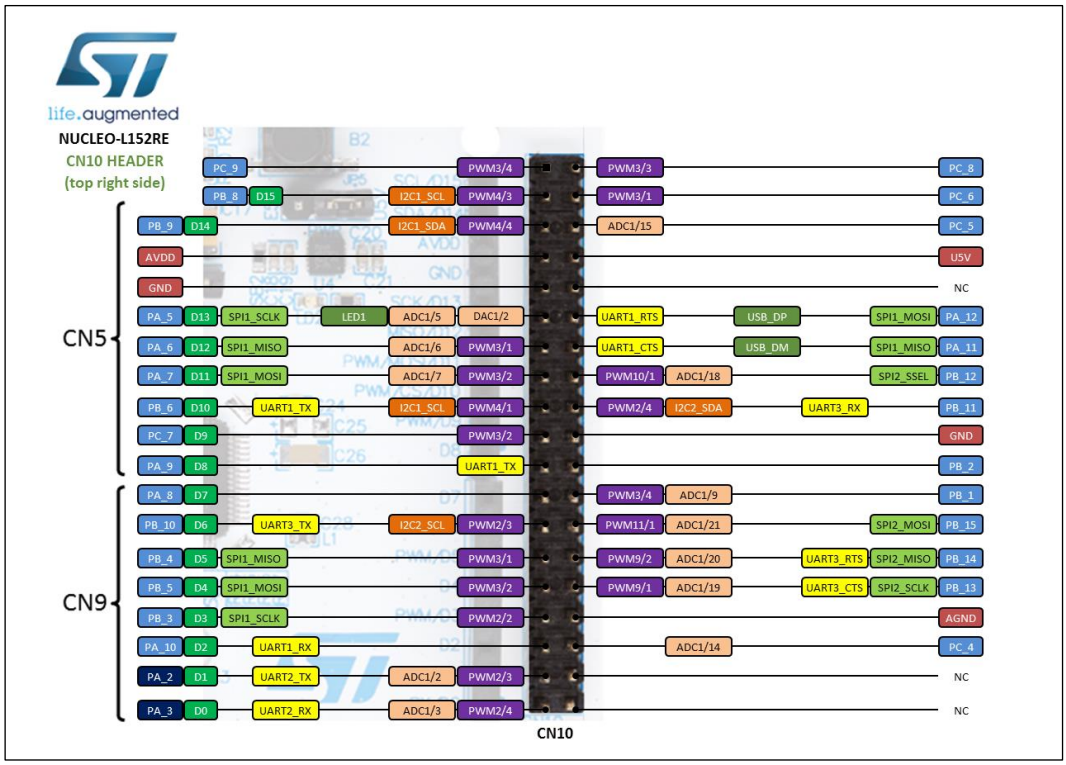

Figura 22. Pin Out CN10 placa STM32L152RE. [11]

Como se ha dicho previamente, el microcontrolador es el elemento indispensable para desarrollar este proyecto.

En nuestro caso, se ha desarrollado la aplicación en el entorno IAR Embedded Workbench IDE en lenguaje C.

4.3. FT4232H-56Q Mini Module

El siguiente componente del que vamos a hablar es el módulo FT4232H-EQ mini. El propósito principal de este módulo es la visualización de la comunicación existente entre los componentes. Nos permite enviar y leer los datos a través de una UART, y así poder ver dicha comunicación en el PC.

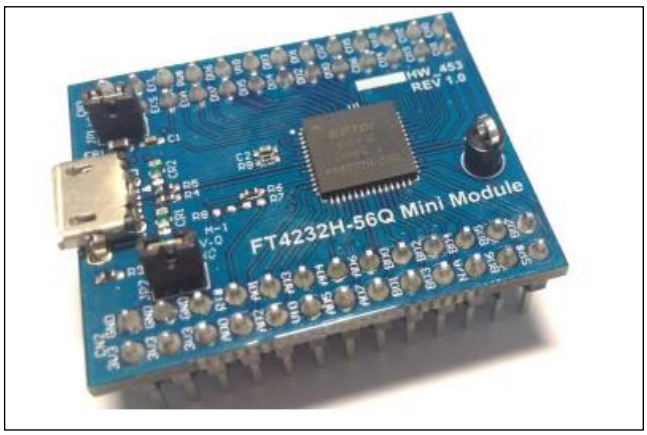

Ilustración 3. FT4232H-56Q Mini Module. [13]

Para ver estos mensajes vamos a usar el software Docklight. Consiste en una herramienta de prueba, análisis y simulación de protocolos de comunicación serie. Así nos permite monitorizar la comunicación entre dos dispositivos serie o de un único dispositivo.

El FT4232H-56Q tiene, entre otras, las siguientes características:

- Basado en el dispositivo USB de alta velocidad FT4232H-56Q.
- Compatible con USB 2.0 de alta velocidad.
- Alimentado por USB: no se necesita fuente de alimentación externa.
- Protocolo USB manejado completamente por el módulo USB.
- Velocidades de datos en serie síncrona (MPSSE) de hasta 30 Mbps en JTAG, SPI e I2C.
- Rango de temperatura de funcionamiento de -40 ° C a + 85 ° C.

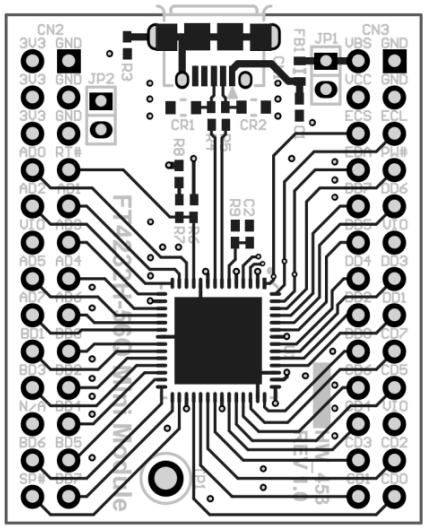

Figura 23. Conexiones eléctricas (pines) del FT4232H-56Q Mini Module.[13]

4.4. Transceptor Sigfox AX-SIGFOX ANTSATMP *

El siguiente componente es el módulo de Sigfox AX.SF10-ANT21-868, en concreto, el AX-SIGFOX ANTSTAMP, que tiene integrado un chip antena de -5dB. No ha sido mencionado al principio de este capítulo ya que, en un principio era el módulo elegido para la comunicación, pero finalmente se descartó ya que no nos permitía la recepción de datos desde el backend. Esto se especifica más en el capítulo 6.

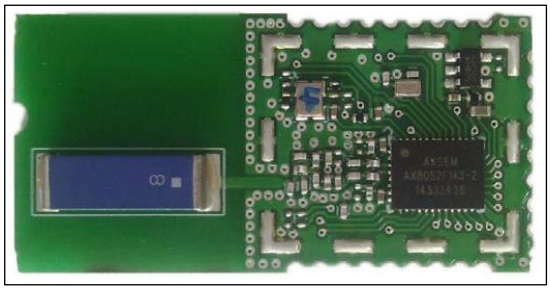

Ilustración 4. Módulo AX-SIGFOX ANTSTAMP.[14]

A pesar de no usar este módulo finalmente, se van a detallar algunas de sus características por las que en un principio era el mejor candidato.

Las características más destacables del módulo son:

- Control mediante comandos AT, que permiten la comunicación entre el usuario y el módulo.
- Consumo ultra-bajo: Modo *DeepSleep* 500nA, modo *Sleep* 1.6 µA y modo *Standby* 0.5mA.
- Comunicación bidireccional, que permite enviar y recibir datos hacia y desde el backend.
- Como medidas: 18.2 x 39.7 x 3 mm^3 (incluyendo el chip antena).
- Alimentación de 1.8V a 3.3V.
- Sensores de temperatura y de tensión.
- Suscripción Sigfox de dos años.
- 10 pines GPIO.
- Velocidad de transmisión de datos de 100bps.

También destacar que los mensajes que puede soportar Sigfox son de hasta un máximo de 12 byes, y solo permite que se transmitan 140 mensajes al día.

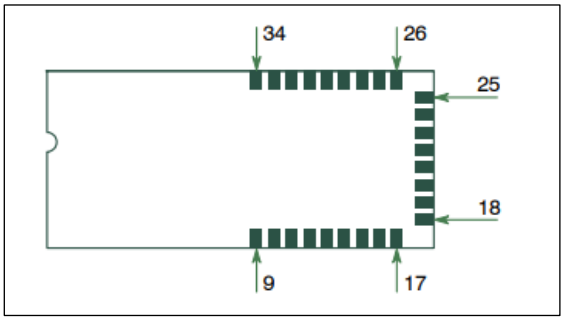

Figura 24. Pines Módulo AX-SIGFOX ANTSTAMP. [14]

4.5. NB-IoT HAT basado en SIM7080G

Por último, tenemos el módulo basado en SIM7080G de SIMCom, el componente que nos va a permitir la comunicación entre el actuador y el servidor. Este es un HAT de telecomunicaciones que cuenta con múltiples funcionalidades de comunicación, entre ellas **NB-IoT** (Internet de las cosas de banda estrecha) además de tener una gran capacidad de extensión con interfaces que incluyen UART, GPIO, PCM, SPI, I2C, etc.

Este módulo proporciona flexibilidad y facilidad de integración para la aplicación del cliente. Debido a las ventajas que presenta de bajo retardo, bajo consumo de energía y amplia cobertura, es ideal para aplicaciones IoT. En nuestro caso, monitorización y actuación remota.

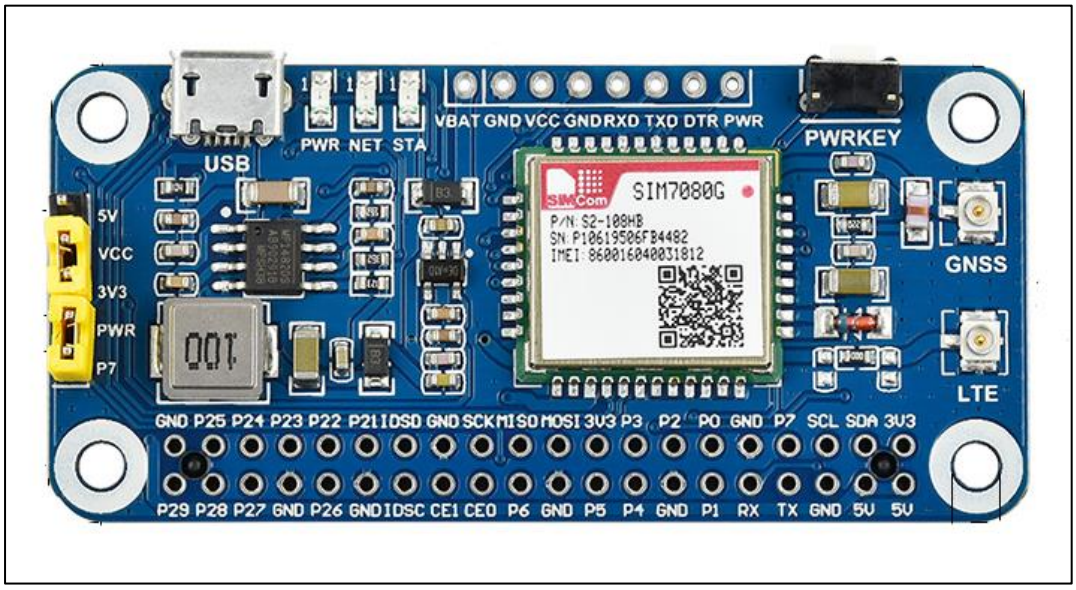

Ilustración 5. Modulo NB-IoT HAT basado en SIM7080G. [15]

Las características de este HAT son:

- Encabezado de extensión estándar Raspberry Pi 40PIN GPIO, compatible con placas de la serie Raspberry Pi.
- Admite protocolos de comunicación como TCP/UDP/HTTP/HTTPS/TLS/DTLS/PING/LWM2M/COAP/MQTT.
- Admite posicionamiento GNSS.
- Interfaz USB integrada, para probar los comandos AT, obtener datos de posicionamiento GPS, etc.
- Pines de control Breakout UART, para conectar con placas anfitrionas como Arduino/**STM32**.
- Traductor de voltaje integrado, 3,3 V por defecto, permite cambiar a 5 V a través de un puente integrado.
- Ranura para tarjeta SIM, solo admite tarjeta SIM de 1,8 V.
- 3 indicadores LED, fácil de controlar el estado de funcionamiento.
- Velocidad de transmisión: 300 ~ 3686400 bps
- Negociación automática de velocidad en baudios común: 9600/19200/38400/57600/115200 bps.
- Corriete general (modo inactivo) de 39 mA.

A continuación, se detallan algunas características de la placa SIM7080G de SIMCom usada, así como de su diagrama de bloques.

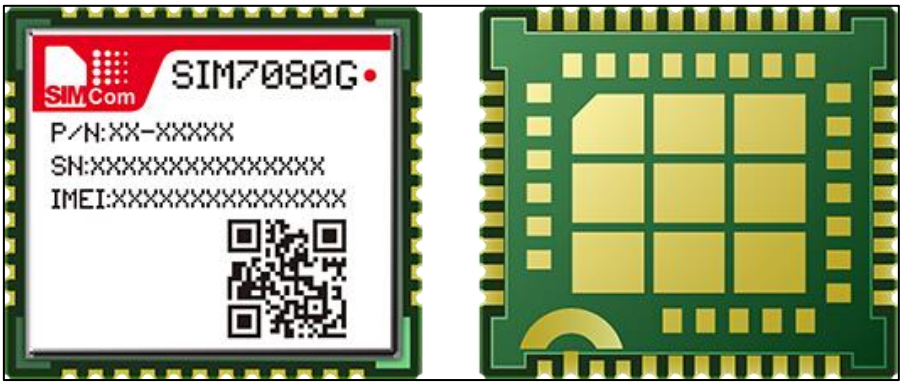

Ilustración 6. Placa SIM7080G de SIMCOM. [16]

- Rango de alimentación entre 2.7V y 4.8V.
- Consumo en sleep mode: 0.6mA y en PSM mode: 3uA.
- Peso de 1.4g.
- Interfaces: UART, SIM card, PCM/I2S/SPI, I2C, GPIO, ADC, USB.
- Control mediante commandos AT.
- Rango de temperature entre -40ºC y 90ºC.
- Transmisión de datos:
    - **Cat-M:** Uplink 1119Kbps, Downlink 589Kbps.
    - **NB-IoT:** Uplink 150Kbps, Downlink 136Kbps.

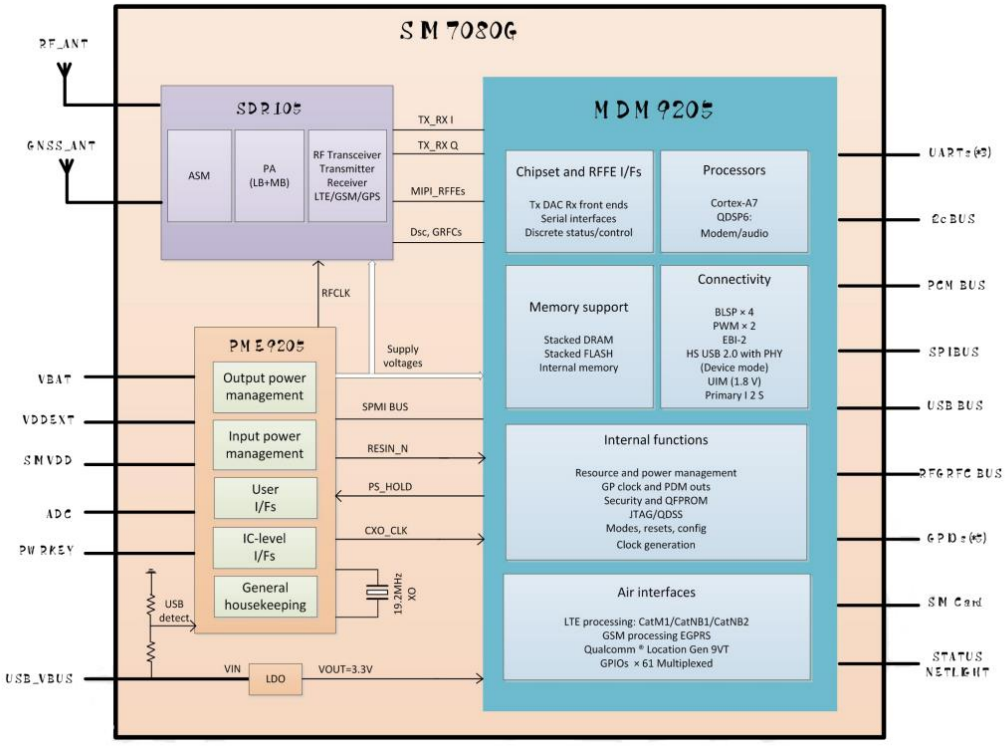

Figura 25. Diagrama de bloques SIM7080G. [15]

4.6. Conexión del hardware

En este capítulo se va a realizar una breve explicación de como se ha realizado la conexión entre los distintos módulos para su comunicación.

El montaje que se ha utilizado para este proyecto está compuesto por las tres placas y el actuador. El microcontrolador es el núcleo que se encarga de procesar todo lo que se envía a y se recibe del modulo de NB-IoT, el cual se encarga de recibir y enviar datos al backend, y por último el TTL que lo usaremos a modo de depuración.

En la siguiente imagen se muestran en detalle los pines de los diferentes módulos para la conexión entre ellos.

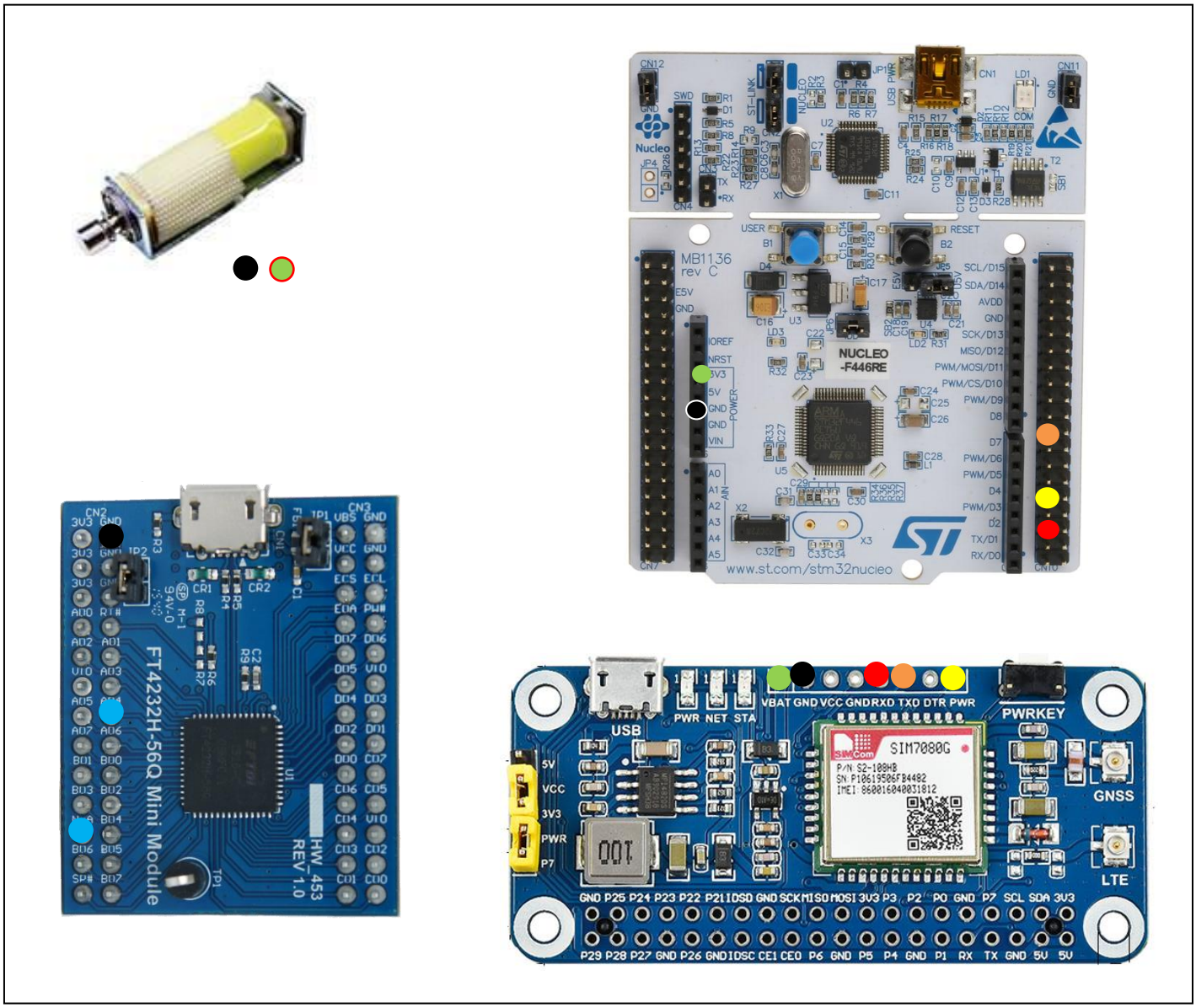

Ilustración 7. Identificación de los pines de los módulos (detalle).

Leyenda de pines:

- PWR
- TX
- RX
- GND
- Alimentación
- RX del módulo TTL
- Alimentación actuador

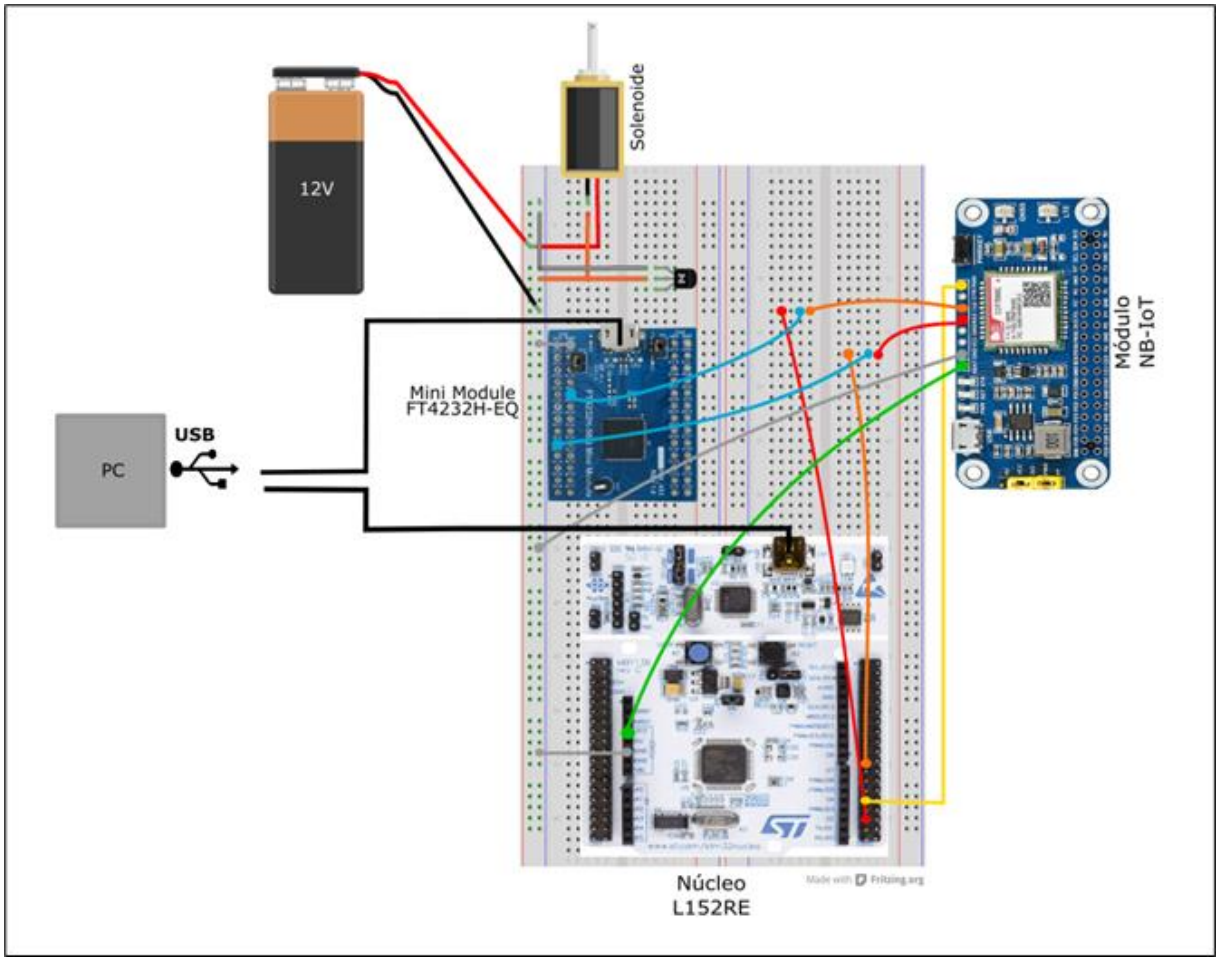

Figura 26. Esquema del setup.

Con la distribución de la figura anterior vemos que tanto la núcleo como la TTL están conectadas al PC mediante USB (quedando así alimentadas). Mientras que el módulo de NB-IoT comparte la alimentación con la núcleo de 3.3V. Asimismo, las tre placas y el actuador comparten tierra.

El pin PWR key de la placa de NB-IoT, está conectado a el pin PB5 GPIO del microcontrolador.

Por otro lado, los canales de tranmisión de módulo de NB-IoT se conectan de manera cruzada a los de el microcontrolador.

TX NB-IoT RX micro

RX NB-IoT TX micro

Con respecto al módulo TTL, este se conecta en paralelo a la conexión (RX-TX) entre el microcontrolador y el módulo de NB-IoT ya que este intenta recibir todos los mensajes, tanto los enviados por el micro como por los enviados por el módulo de NB-IoT.

Por último, en cuanto al montaje del actuador hemos usado el siguiente:

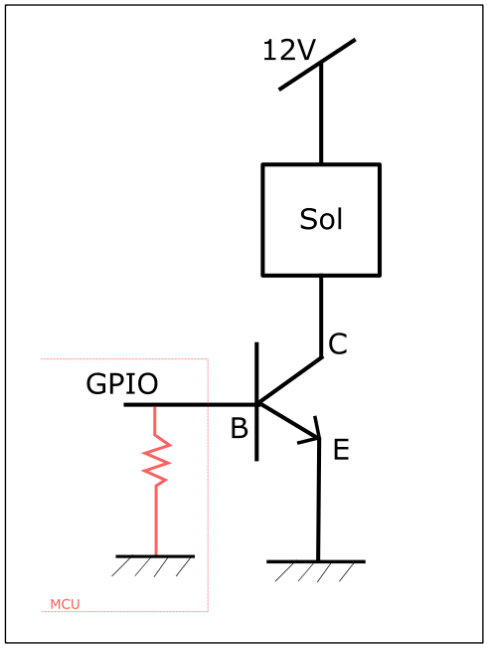

Figura 27. Esquema montaje actuador.

Usando un transistor NPN (BD125G) y alimentándolo con una batería de 12 V conectada al cable blanco del actuador, y a tierra. Antes de conectarlo al sistema ha sido probado con una fuente externa para comprobar su funcionamiento y su consumo. La resistencia de color rojo corresponde con la resistencia de pull-down del GPIO interna al microcontrolador.

El montaje que hemos expuesto en este capítulo no es óptimo, solo pretende demostrar la viabilidad que tiene el uso de estas tecnologías con la finalidad que nos ocupa, el control de un sistema de precintado remoto. El objetivo de este proyecto es demostrar que es posible, siendo tanto el esquema, como el actuador, como la forma de controlarlo muy optimizable.

A continuación, se muestra una imagen del setup real del proyecto.

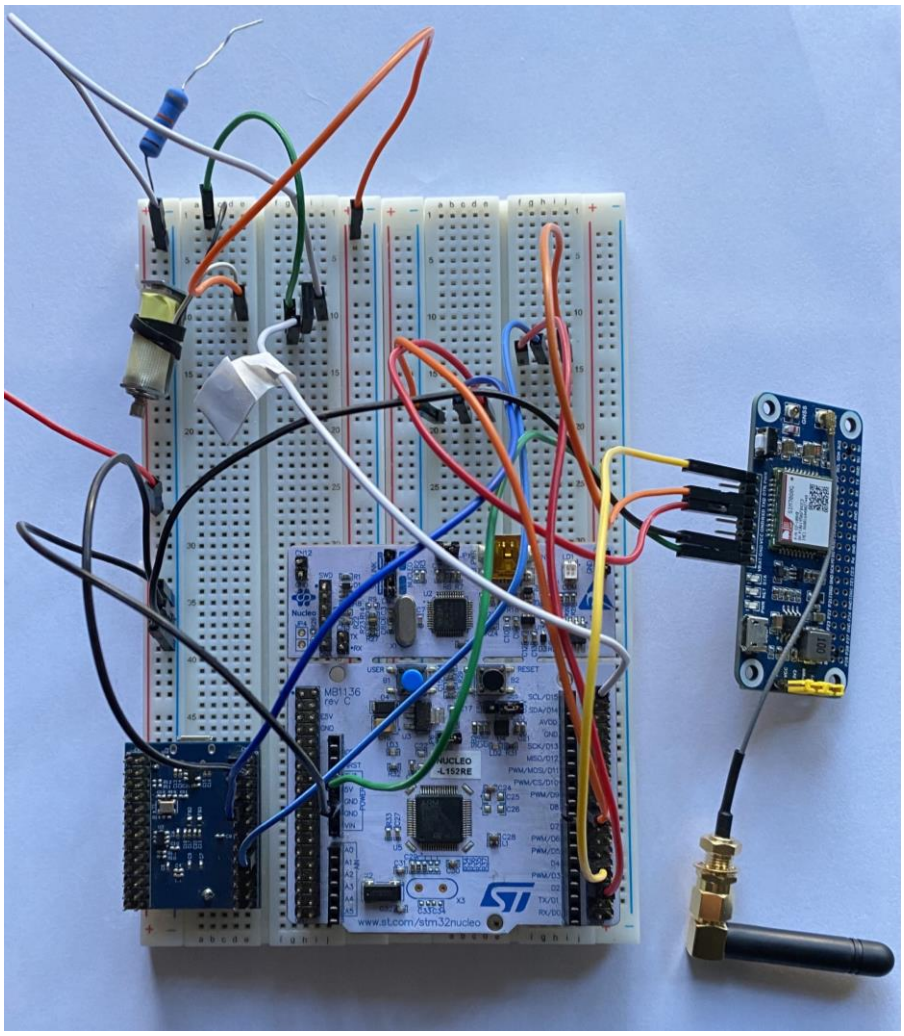

Ilustración 8. Setup real del hardware.

En el siguiente capítulo, se va ha desarrollado el driver que controla el actuador y sus funcionalidades como: configurar el pin GPIO al que corresponderá el mismo, leer del pin, escribir en el pin y actuar según nos indique la información que recibamos de la UART.

5 DISEÑO Y DESARROLLO DEL DRIVER DEL SOLENOIDE

En este capitulo se plantea la metodología seguida para desarrollar el driver que implica al solenoide. Vamos a centrarnos, en primer lugar, en definir, tanto el problema, como las especificaciones que queremos que posea la solución. Y en segurndo lugar, en el diseño de dicha solución, teniendo en cuenta las características de los módulos que hemos descrito anteriormente.

5.1. Definición del problema y diseño de la solución

Se desea conseguir un *driver* que nos permita controlar el actuador DSOL-0844-12. Esto nos presenta el siguiente problema. Al tratarse de un dispositivo de bajo consumo, el usuario no va a poder enviarle una solicitud, en este caso, de apertura del solenoide, cuando el quiera ya que el micro va a estar "dormido" y no la va a escuchar. Por esto, lo que va a hacer el microcontrolador es que periódicamente se va a "despertar" y va a avisar de esto al servidor a la espera de que este le indique que hacer, abrir el actuador concretamente.

Este es el principal problema, o la principal especificación que debemos tener en cuenta a la hora de desarrollar este proyecto. Esta es la razón por la que tenemos que enviar algo hacía arriba, uplink, hacia el servidor, porque como el micro esta dormido. No es posible que se envie algo hacia abajo, desde el servidor porque no sabemos si esta dormido o despierto. Por ello vamos a esperar a que se nos idique que esta disponible o "despierto" y esa será la señal para yo poder decirle que hacer. Es importante remarcar que el nodo tiene que enviar primero un mensaje hacia arriba, aunque queramos dar la orden hacia abajo.

Se va a recibir una respuesta a la solicitud de apertura a través del módulo de NBIoT, y en función de esta respuesta se abrirá o no el solenoide.

Durante el Desarrollo del software se tienen en cuenta algunas especificaciones entorno al método de desarrollo:

- El código está escrito en lenguaje C e implementado en el entorno de desarrollo IAR Embedded Workbench.
- Se hará uso de las librerías proporcionadas por el GIE, como son las librerías de la UART y los GPIO.
- El código estará integrado en un fichero .c en el que se desarrollan íntegramente las funciones y un fichero .h en el que se declaran las funciones, mensajes de error y estructuras.

Para el diseño de la solución se tiene en cuenta lo planteado en la definición del problema. Por ello se ha realizado un *driver GIE_SOLENOIDE_DRIVER* que nos permite, por un lado, configurar la UART y el pin GPIO que vamos a usar.

Por otro lado, se usarán dos funciones, definidas en el *driver* "GIE_UART_DRIVER" que se usarán para, por un lado, leer de el pin GPIO (solo lee valores 0 o 1) y por otro, que es la acción de verdaderamente nos interesa que es escribir en el pin.

El usuario enviará a través del módulo de NBIoT una solicitud de apertura del actuador, y según nos responda, enviaremos por el pin un 1 (apertura) o un 0 (no apertura).

5.2. Desarrollo de la solución

A continuación, se procede a desarrollar todas las funciones que proporcionan una solución al proyecto y que permiten la implementación del driver. [17]

- uint8_t SOL_CONFG_INIT()
- uint8_t SOL_Write_PIN(GPIO_PIN configPIN, uint8_t val)
- uint8_t SOL_Read_PIN(GPIO_PIN configPIN)
- uint8_t SOL_Actuacion(uint8_t UART_PORT, GPIO_PIN configPIN, char esperado)

uint8_t SOL_Write_PIN(GPIO_PIN configPIN, uint8_t val)

Con esta función vamos a realizar lo que de verdad nos interesa de este driver que es, escribir a través del pin para la actuación del solenoide.

Se le pasa como parámetros, la configuración del pin por el que vamos a mandar el mensaje y un valor (0 o 1) según lo que queramos hacer.

Para ello usamos la función *GPIO_Write* también definida en el driver "GIE_GPIO_DRIVER". Esta función escribe lo que hay en el espacio de memoria que indicamos con un puntero en el pin definido en la configuración del pin, además devuelve un "OK" o un error dependiendo de si ha sido posible la escritura o no.

uint8_t SOL_Read_PIN(GPIO_PIN configPIN)

Esta función sirve para leer del pin. Hace uso de la función *GPIO_Read* del driver "GIE_GPIO_DRIVER", proporcionada por el GIE. Por ello le pasamos como parámetro la configuración del pin.

Destacar que también se ha creado una variable global para esta función llamada *valor* ya que la función de *GPIO_Read* necesita guardar lo que lee en la memoria.

uint8_t SOL_Conf_Init()

Esta función nos sirve para configurar el GPIO que vamos a usar.

Por ellos hacemos uso de dos estructuras ya definidas en el *driver* "GIE_GPIO_DRIVER" que son _GPIO_PIN Y _GPIO_Struct_Config. La primera nos sirve para definir el pin que vamos a usar indicando su número y su puerto. La segunda, define la configuración del GPIO, su modo (entrada,salida…) y su pull (pull up, pull down…).

En nuestro caso las hemos configurado del siguiente modo: ***Número de pin* 15, *puerto de pin* A, *modo de gpio* GPIO_MODE_OUTPUT_PP y *pull de gpio* GPIO_PULLDOWN**.

Por último, usamos la función *GPIO_Init* definida en el *driver* "GIE_GPIO_DRIVER" para confirmar si se ha inicializado bien el pin.

A continuación se muestra el diagrama de flujo de la función *SOL_Conf_Init().*

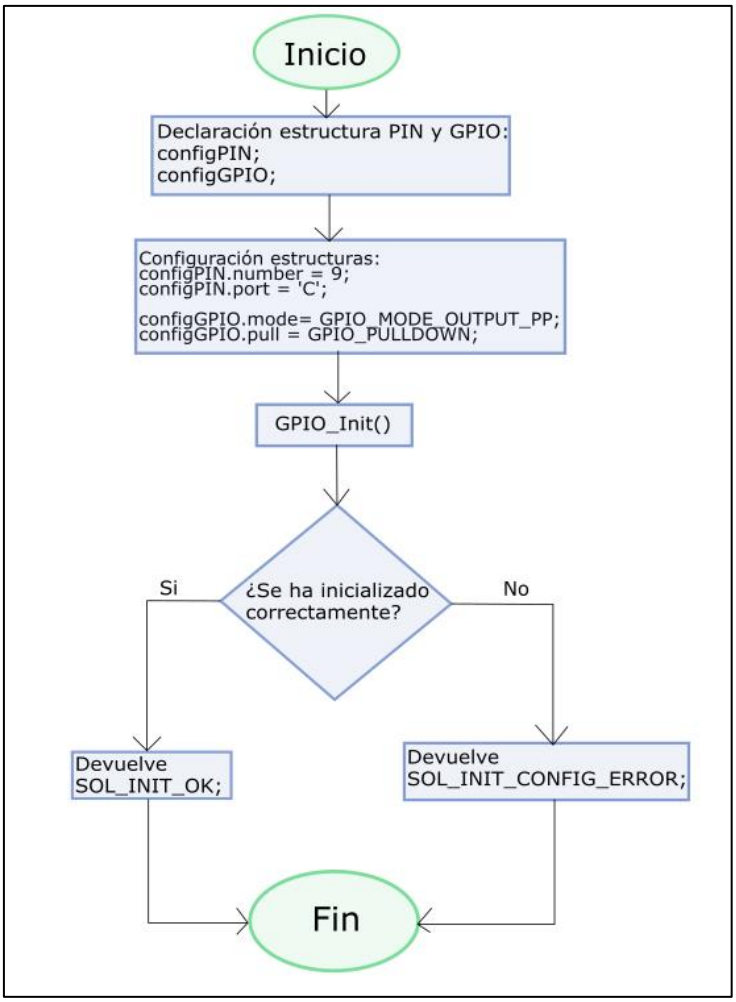

Figura 28. Diagrama función *SOL_Config_Init()*.

uint8_t SOL_Actuacion(uint8_t UART_PORT, GPIO_PIN configPIN, char esperado)

Usaremos esta función cuando necesitemos leer de la UART, un mensaje de confirmación o solicitud de actuación, por ejemplo, y activar o desactivar el pin del solenoide en función de lo leído.

En primer lugar, leerá por la UART (con la ayuda de la función *UART_ReadValue()*, la cual está conectada al módulo NB-IoT y transporta la información del servidor. Lo guardará en la variable *recibido*. Normalmente lo que reciba por UART será un dato a modo de confimacion o de señal de que esta todo Ok para que actue el solenoide.

Después se hará una comparación entre el mensaje recibido y el que debería de ser. Esta comparación se realizará mediante la función *strncmp* de la biblioteca de <string.h>, que compara dos cadenas de caracteres antes de n bytes. Si devuelve un numero entero < 0, la cadena 1 es menor que la cadena 2. Si devuelve un numero entero > 0, la cadena 1 es mayor que la cadena dos. Y finalmente, si devuleve un 0, las cadenas son iguales.

Si la función nos devuleve un 0 entonces escribiremos por el pin un ‘1’ para activar el solenoide. Usando para esto la función *SOL_Write_PIN()* que hemos descrito anteriormente.

Si además queremos dejar el pin desactivado para una nueva secuencia habría que añadir de nuevo la función *SOL_Write_PIN()* y en este caso, escribir un ‘0’ por el pin para desactivar el solenoide.

A continuación se muestra el diagrama de flujo de la función *SOL_Actuacion()*.

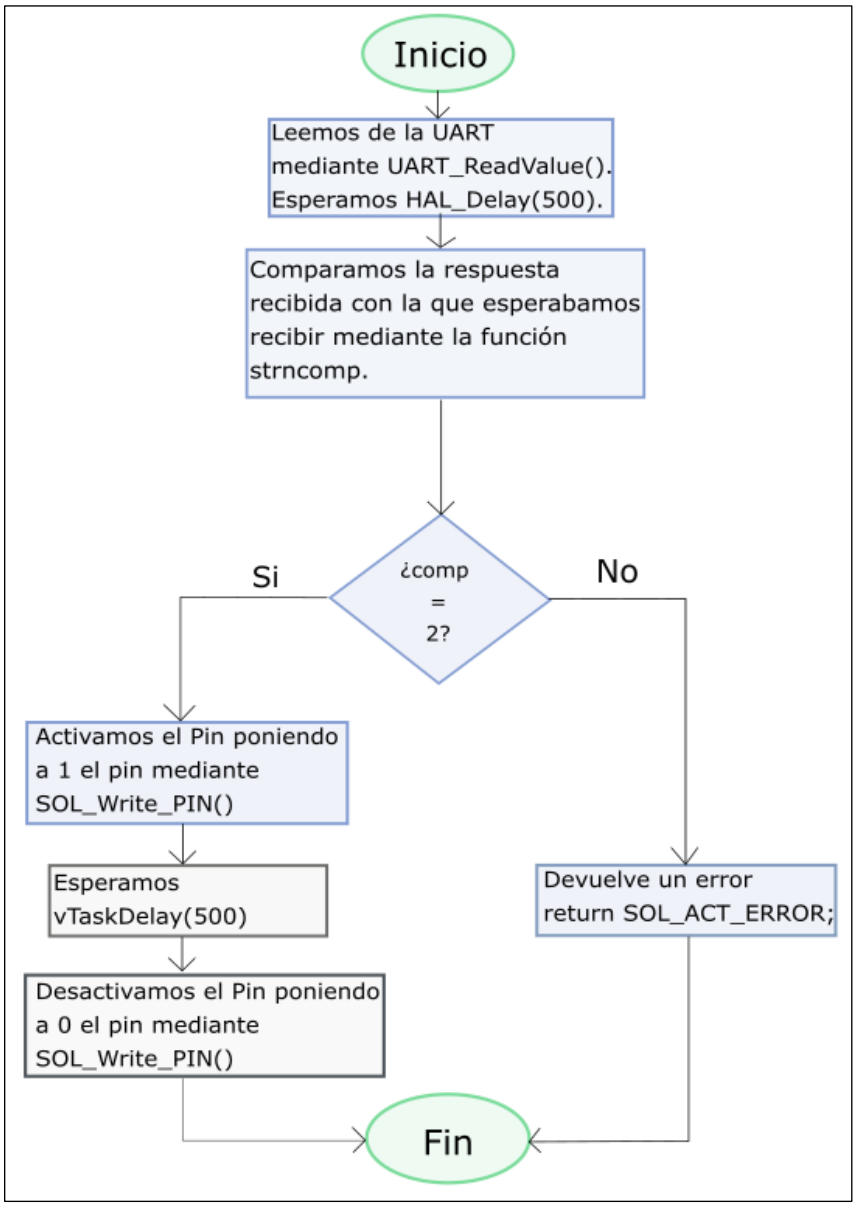

Figura 29. Diagrama función *SOL_Actucaion()*.

A continuación, se describa todo lo relacionado con el módulo de NB-IoT. Lo que en nuestro proyecto corrresponde a la información que manda el nodo al servidor para avisarle de que esta activo, y de la respuesta que este le da si desea que haga algo.

6 DISEÑO Y DESARROLLO DE LA RECEPCIÓN

En este capítulo se va a desarrollar, por un lado, todo lo relacionado con el envío que hace el nodo de la notificación de que se encuentra activo y si el usuario desea realizar alguna operación sobre él, y por otro, la recepción de dicha respuesta del usuario.

Por ello, el módulo que finalmente se usa para poder recibir datos desde el servidor va a ser el SIM7080G de NB-IoT. Haremos uso de un *driver* de NB-IoT (proporcionador por el GIE) y de sus funciones, y las agruparemos en tres bloques, que serán: Incialización, Solicitud y Respuesta.

INCIALIZACIÓN:

Este bloque se encargará de la inicialización tanto del módulo como de su conexión a un servidor. Para ello usaremos las siguientes funciones.

SIM7080_Init (uint8_t UART_Port, uint32_t BaudRate, GPIO_PIN PWRKEY_Pin, uint8_t Comissioning, uint8_t Echo_mode)

Esta función inicializa el módulo SIM7080:

- Inicializa la UART con la función *UART_Init ()*, que en este caso será la UART_1.
- Inicializa el nodo a usar a través del power key pin con la función *GPIO_Init ()*.
- Configura el baudrate del módulo, en este caso a 9600.
- Desactiva o activa el modo echo, en nuestro caso lo desactivará.
- Se utiliza para forzar la conexión con el operador móvil seleccionado. En este caso, durante la inicialización no se fuerza la conexión (Comissioning_Off).

SIM7080_CheckNetworkConnection ()

Esta función la usamos para comprobar mediante comandos AT si el módulo este o no conectado a la red, y el estado de esta conexión.

SIM7080_Comissioning (uint8_t Comissioning)

Se utiliza para forzar la conexión con el operador móvil seleccionado. En este caso, se intenta conectar a Vodafone, debido a que la tarjeta SIM de la que se dispone es de dicho operador. Si no consigue conectar, se establece el modo automático.

La primera vez que se conecta una tarjeta SIM al módulo NBIoT puede tardar unos minutos en conectarse. Una vez que se consiga conectar por primera vez, esa red guarda el operador en el módulo NBIoT y, a partir de ahí, se conectará en unos 10 segundos cada vez que se despierte, siempre que disponga de buena cobertura. En caso de que no sea así, este tiempo puede prolongarse hasta 5 minutos, intante en el que la conexión se cancelaría hasta la siguiente notificación al servidor programada.

SIM7080_InitNetwork ()

Esta función inicia la red del módulo SIM7080 para que se pueda usar en apliaciones como envío de datagramas.

SIM7080_InitConnection (char *tp_mode, char *Port, char *Ip, uint8_t socket)

Se utiliza para iniciar conexión con la apertura del socket.

- Tp_mode: determina si la conexión se realiza mediante UDP o TCP.
- Port: Número del puerto.
- IP: IP del servidor.
- Socket: Parámetro que identifica el socket (número del 0 al 12). Usa la función *SIM7080_CheckSocketStatus (uint8_t socket)* para comprobar el estado del socket.

En nuestro caso la conexión se realiza mediante UDP, en la IP del puerto 86.109.110.101, con puerto 7013 y socket número 0.

SOLICITUD:

SIM7080_TransmitDatagram (uint8_t socket, char *msg)

Esta función envía datos a través de la conexión establecida.

- Socket: Socket al que queremos enviar los datos.
- *msg: Trama que se quiere enviar.

En nuestro caso enviaremos un "1" a través del socket número cero. Esta función hace uso del comando AT, **"AT+CASEND"** para enviar datos a través de una conexión establecida por UDP.

Con esta función pretendemos enviar el mensaje uplink del que hablamos en el punto de *Objetivos* en la *Introducción*. Como señalamos, al tratar con dispositivos de bajo consumo consumo no va a ser posible enviarle una solicitud cuando nosotros queramos porque el microcontrolador va a estar "dormido". Por ello enviamos primero este mensaje hacia arriba para indicar que estamos disponibles y así ya poder solicitar que haga lo que nosotros queramos. Esto lo haremos con el driver del solenoide del punto anterior.

RESPUESTA:

_SIM7080_SendATCommand (char* at_command, const char* expected_response, char *response, uint16_t response_size, uint32_t response_timeout)

Con esta función mandamos comandos AT y verificamos que la respuesta es la esperada.

- At_comand: Comando AT que queremos enviar.
- Expected_response: Extracto de la respuesta esperada.
- Response: Cadena con la respuesta recibida.
- Response_size: Tamaño de la respuesta recibida.
- Response_timeout: Timeout de la respuesta recibida.

Vamos a usar el comando AT, **"AT+CARECV"** y esperamos recibir un "OK" como confirmación de que su envío.

En el *anexo* se puede ver los comandos AT que hemos usado en este proyecto y su explicación.

_SIM7080_GetSubstring (char* message, char* from_this_string, char* to_this_string, char* substring)

Esta función permite obtener un substring dentro de un string inicial. En nuestro caso la vamos a usar para obtener de la respuesta recibida los datos que nos interesan, que serán los correspondientes entre "+CARECV:1," y "\r\n" que debería ser un "2".

- Message: Cadena de caracteres inicial. En nuestro caso la cadena recibida de la función *_SIM7080_SendATCommand ()*.
- From_this_string: Cadena de caracteres justo anterior a la substring deseada. "+CARECV: 1,".
- To_this_string: Cadena de caracteres posterior a la substring deseada. "\r\n".
- Substring: Substring obtenida.

Por último, hemos usado otras dos funciones del driver de NBIoT para cuando ya hemos realizado la petición y queremos cerrar la conexión y poner el módulo en bajo consumo a la espera de una nueva conexión.

SIM7080_CloseConnection (uint8_t socket): Cierra todos los sockets activos y termina la conexión.

SIM7080_SetPowerMode (uint8_t power_mode): Establece el modo de consumo del módulo, en nuestro caso un modo de bajo consumo para la espera.

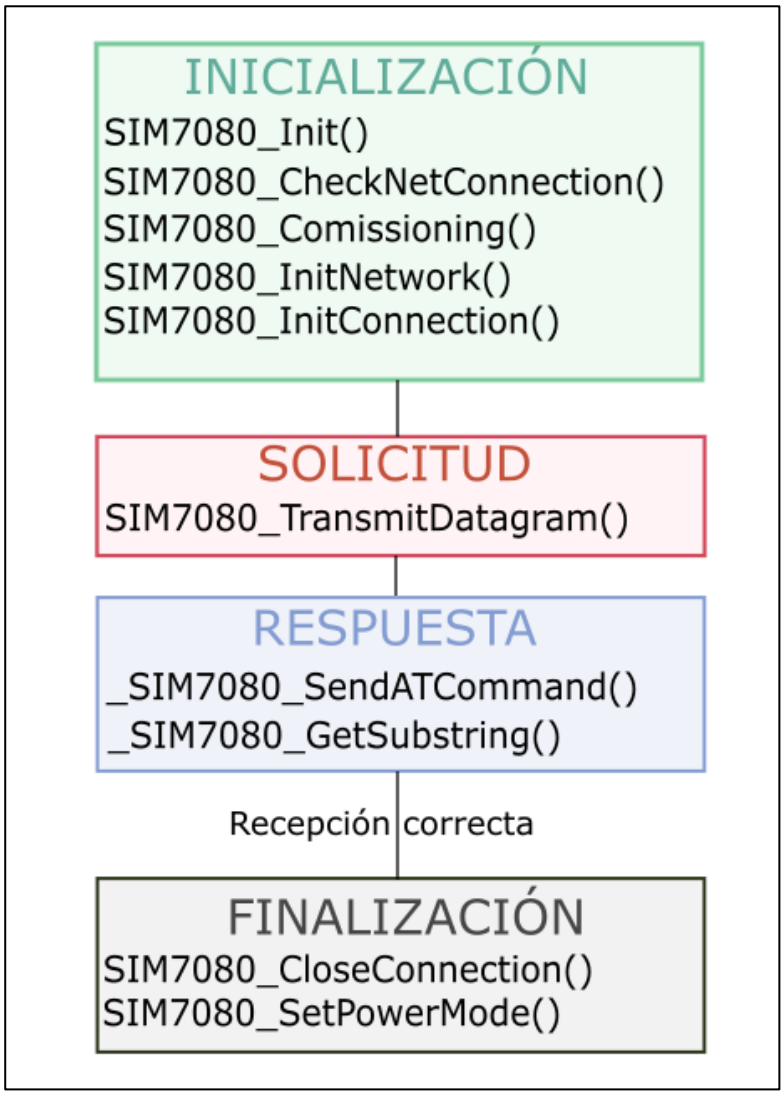

Figura 30. Diagrama driver NB-IoT.

7 SOBRE FREERTOS

El propósito de este capítulo es hacer una pequeña introducción sobre qué es FreeRTOS y sus recursos más importantes para así facilitar la comprensión de capítulos posteriores.

FreeRTOS es un sistema operativo en tiempo real líder en el mercado para microcontroladores y pequeños microprocesadores que facilita la programación, la implementación, la protección, la conexión y la administración de estos dispositivos pequeños y de bajo consumo. Desarrollado en colaboración con las principales empresas fabricantes de chips del mundo y distribuido libre y gratuitamente bajo la licencia de open-source del MIT, FreeRTOS incluye un núcleo y un conjunto creciente de bibliotecas de IoT adecuadas para su uso en todos los sectores industriales y sus aplicaciones.

FreeRTOS brinda todo lo que necesita para programar fácilmente dispositivos conectados basados en microcontroladores y recopilar datos de dichos dispositivos para las aplicaciones de IoT. Puede conectar de forma segura dispositivos de FreeRTOS a servicios en la nube, como AWS IoT Core, a un dispositivo de borde local o a un dispositivo móvil mediante Bluetooth de bajo consumo, y actualizarlos de manera remota mediante la característica de actualización inalámbrica.

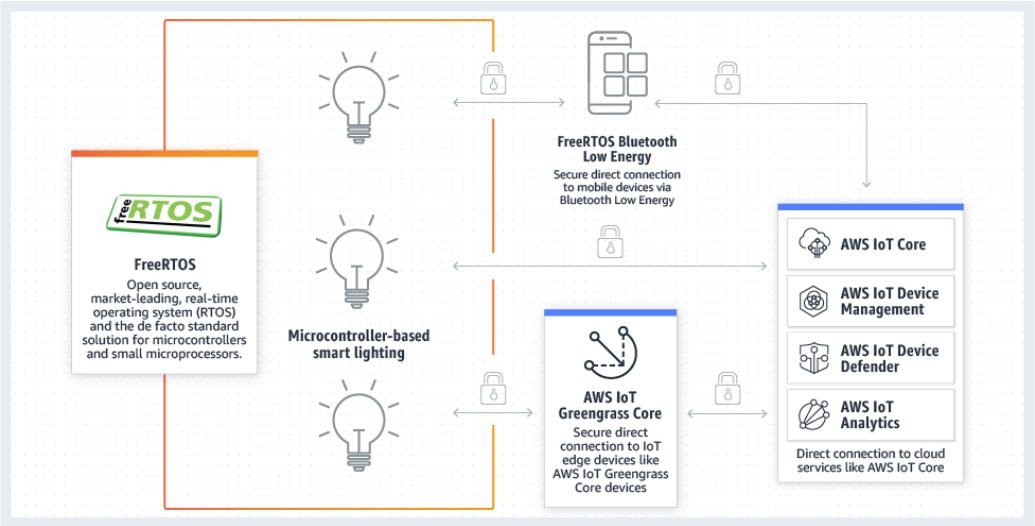

Figura 31. Esquema general de FreeRTOS. [19]

Entre las principales ventajas que ofrece FreeRTOS tenemos:

- La multitarea. Es decir, una tarea puede suspender su ejecución para que otra tarea se ejecute sin que perder su contexto de ejecución. Por eso se dice que usar un OS aumenta la confiabilidad del sistema.
- Aumento del determinismo del sistema. Existen aplicaciones donde el tiempo de respuesta del sistema es un parámetro crítico. Por ellos, dicha respuesta debe ubicarse dentro de una ventana de tiempo, que la respuesta no sea ni demasiado temprana ni demasiado tardía.
- Recursos para asegurar la respuesta del sistema con cierta tolerancia como son las tareas con prioridad definida, los semáforos, las colas, mutex….
- Es un kernel de tiempo real: simple, portable y libre. Mayormente escrito en C.
- Distribución. FreeRTOS puede ser compilado por más de veinte compiladores diferentes y puede ejecutarse en más de 30 arquitecturas de procesadores diferentes.

7.1. Tareas

Cada aplicación consiste en varias Task (Tareas). Cada tarea se ejecuta dentro de su propio contexto, sin dependencia de otras tareas. Como el microcontrolador (normalmente) sólo tiene un núcleo, sólo se ejecuta una tarea en cada momento. Por lo que las tareas pueden estar "funcionado"(running) o "no funcionando"(not running). El programador de RTOS en tiempo real determina cuándo debe ejecutarse cada tarea. Cada tarea se proporciona con su propia pila. Cuando una tarea se intercambia para poder ejecutar otra tarea, el contexto de ejecución de la tarea se guarda en la pila de la tarea de modo que puede restaurarse cuando esa misma tarea vuelva reanudar su ejecución más adelante. [20]

Hay que tener en cuenta que el estado "no funcionando" posee varios subestados. En el estado funcionando, se está ejecutando código. Cuando está en no funcionando, la tarea está inactiva esperando a que el programador la active.La transición de no funcionando a funcionado se denomina "switched in" o "swapped in" y, al contrario, "switched out" o "swapped out". Estos cambios sólo los puede realizar el scheduler (programador de tareas) del FreeRTOS.

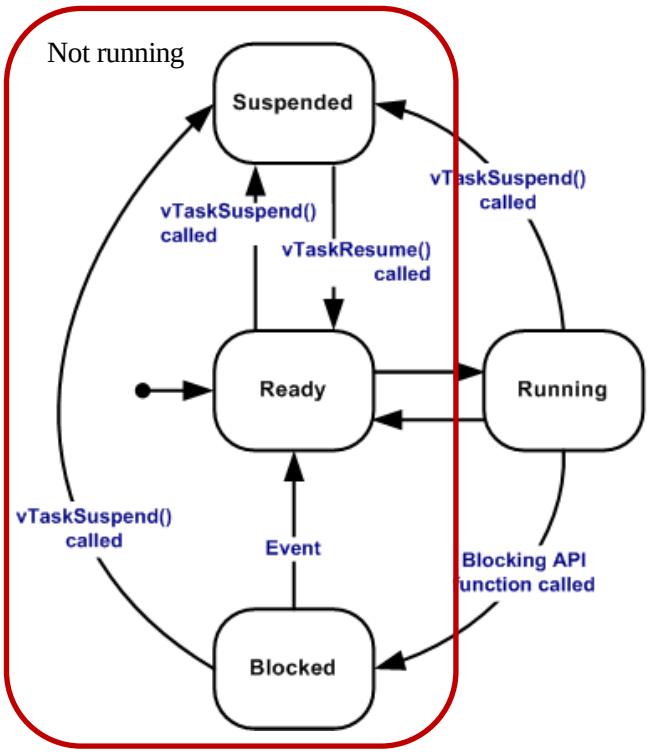

Figura 32. Transiciones de estado de tareas FreeRTOS. [20]

Estado Running: se está ejecutando código.

Estado No Running:

- **Blocked:** Tareas que están esperando un evento. Hay dos tipos de eventos por los que pueden entrar aquí: Eventos de sincronización (esperar que termine otra tarea o a una interrupción) y eventos temporales (esperar a que pase un periodo de tiempo determinado). Una manera eficiente de hacer que una tarea esepere por una cantidad de tiempo es usar la función *vTaskDelay()*, que pone a la tarea en estado blocked por un número de Ticks.
- **Suspended:** en este estado, el planificador de tareas no toma en cuenta dicha tarea. Para poner una tarea en este estado se usa la función *vTaskSuspend()* y para salir del mismo *vTaskResume().* Este estado no se suele utilizar.
- **Ready:** las tareas que están en not running pero no están Blocked ni Suspended se encuentran en Ready. Están esperando que el *Schedule* las active.

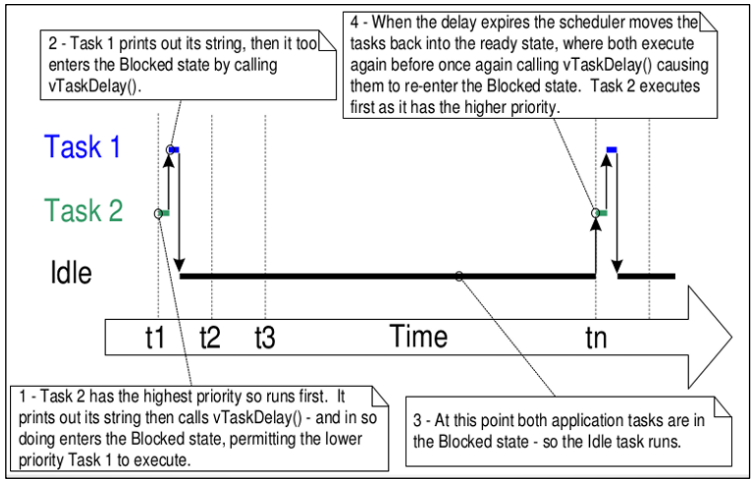

Figura 33. Ejemplo de tarea en estado Blocked por *vTaskDelay()*. [21]

Para proporcionar comportamiento determinista en tiempo real, el programador de tareas de FreeRTOS permite asignar prioridades estrictas a las tareas. RTOS garantiza que la tarea de máxima prioridad que pueda ejecutar reciba tiempo de procesamiento. Esto requiere compartir el tiempo de procesamiento entre tareas de la misma prioridad si están listas para ejecutarse en el mismo momento. FreeRTOS también crea una tarea de inactividad que ejecuta solo cuando no hay otras tareas listas para ejecutarse.

7.1.1. xTaskCreate()

Esta función es sirve para crear tareas. [22]

```
xTaskCreate(TaskFunctiont_y  pvTaskCode, const char * const pcName, uint16_t usStackDepth, void *pvParameters, UBaseType_t uxPriority, TaskHandle_t *pxCreatedTask);
```

Figura 34. Prototipo de la función *xTaskCreate()*.

- **pvTaskCode:** es un puntero a la función que implementa la tarea.
- **pcName:** un nombre descriptivo de la tarea. Sólo sirver para debug.
- **usStackDepth:** cada tarea tiene una pila de programa propia en el kernel, este parámetro determina el tamaño de la pila (en words (ancho del stack del sistema)).
- **pvParameters:** son los parámetros que necesita la tarea.
- **uxPriority:** define la prioridad de la tarea de 0 a configMAX_PRIORITIRES-1.
- **pxCreatedTask:** para pasar un argumento a la tarea. Si no se desea utilizar, se pone NULL.

Hay dos posibles valores de retorno:

1. **pdPASS:** la tarea se ha creado con éxito.
2. **errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY:** no hay suficiente memoria para crear la tarea.

7.1.2. Prioridad de las Tareas

El parámetro uxPriority de la función *vTaskDelay()* asigna la prioridad inicial al crearse la tarea, pero esta se puede cambiar llamando a la función de la API *vTaskPrioritySet().*

El valor máximo de prioridades disponible es el definido por la constante configMAX_PRIORITIRES en el archivo FreeRTOSConfig.h. FreeRTOS no tiene un límite máximo de valores de prioridad pero sí está limitado por la RAM del microcontrolador.

No hay resticciones en la definición de la prioridad se puede compartir el nivel de prioridad o poner un único nivel a todo. Eso sí, el planificador de tareas activará las tareas con un mayor grado de prioridad y en el caso de que vaya a activar una tarea prioritaria y hay varias con la misma prioridad, el planificador actiavará una y desactivará la otra sucesivamente.

Para poderse ejecutar la siguiente tarea, el planificador se activa al finalizar cada periodo de las tareas. Este periodo, se determina con la constante configTICK_RATE_HZ en el archivo FreeRTOSConfig.h.

Como la API del FreeRTOS trabaja con tick, la constante portTICK_RATE_MS proporciona el ratio de conversión de ticks a milisegundos y “Tick Count" determina el total de ticks que han ocurrido desde que se inicialializó el planificador teniendo en cuenta que no se haya saturado. [18]

7.2. Colas y semáforos

7.2.1. Colas:

Las colas son la principal forma de comunicación entre tareas. Pueden utilizarse para enviar mensajes entre tareas o entre interrupciones y tareas. En la mayoría de los casos, se utilizan como búferes FIFO (primero en entrar, primero en salir) seguros para subprocesos y los datos nuevos se envían al final de la cola. (Los datos también se puede enviar al principio de la cola). Los mensajes se envían a través de colas mediante copia, lo que significa que los datos (que pueden ser un puntero a búferes de mayor tamaño) en sí se copian en la cola, en lugar de limitarse a almacenar una referencia a los datos.

Las API de cola permiten especificar un tiempo de bloqueo. Cuando una tarea intenta leer una cola vacía, la tarea se coloca en estado bloqueado hasta que haya datos disponibles en la cola o hasta que transcurra el tiempo de bloqueo. Las tareas en estado bloqueado no consumen tiempo de CPU, lo que permite ejecutar otras tareas. Asimismo, cuando una tarea intenta escribir en una cola vacía, la tarea se coloca en estado bloqueado hasta que haya espacio disponible en la cola o hasta que transcurra el tiempo de bloqueo. Si hay más de una tarea bloqueada en la misma cola, se desbloquea primero la tarea con la prioridad más alta.

Varias tareas pueden leer o escribir en la misma cola.

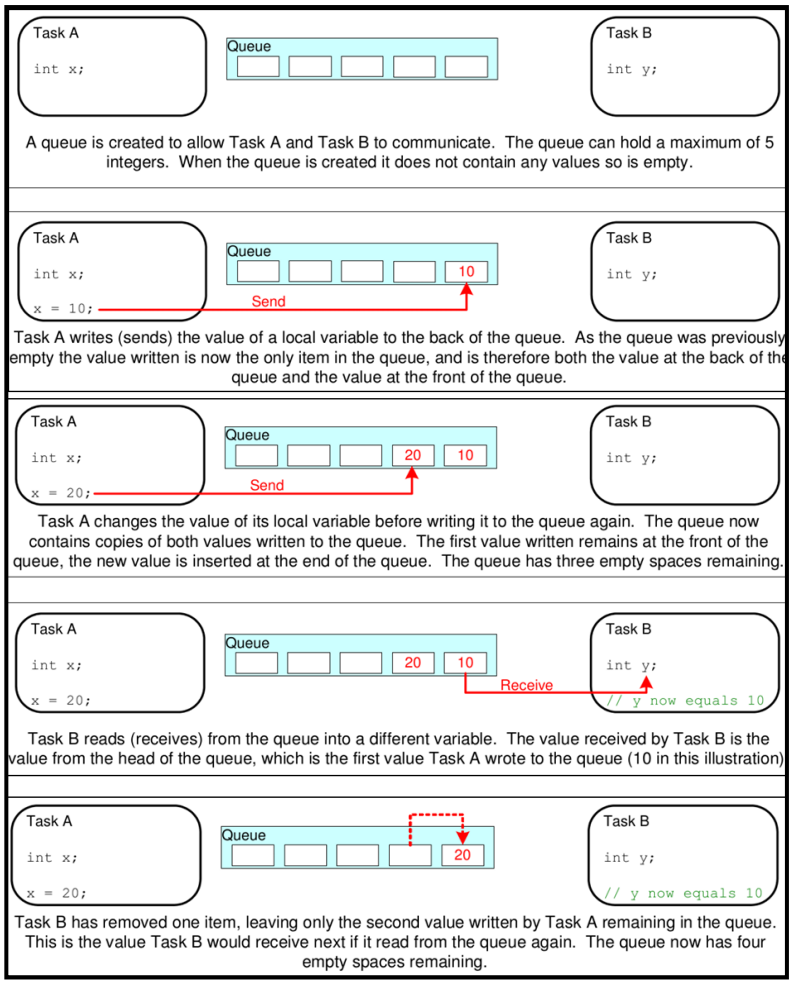

Figura 35. Funcionamiento de una Cola.[21]

7.2.1.1. xQueueCreate()

Esta función sirve para crear una cola.

Las colas se identifican por el tipo de variable xQueueHandle y a odnde se devuelve la referencia de esta. [22]

xQueueCreate(usigned portBASE_TYPE uxQueueLength, unsigned portBASE_TYPE uxItemSize);

Figura 36. Prototipo de la función *xQueueCreate()*.

- **uxQueueLength:** el número máximo de datos que la cola puede almacenar.
- **uxItemSize:** el tamaño en bytes de cada item de la cola.

Devuelve NULL si no hay suficiente RAM. Un valor distinto de NULL indica que se ha creado la cola.

7.2.1.2. xQueueSendToBack(), xQueueSendToFront() y xQueueSend()

xQueueSendToBack() se usa para mandar el dato al final de la cola y *xQueueSendToFront()* para mandarlo al principio de la cola. *xQueueSend()* es lo mimos que *xQueueSendToBack()*. [22]

```
portBASE_TYPE xQueueSendToFront ( xQueueHandle xQueue, const void * pvItemToQueue, portTickType xTicksToWait);
portBASE_TYPE xQueueSendToBack ( xQueueHandle xQueue, const void * pvItemToQueue, portTickType xTicksToWait);
portBASE_TYPE xQueueSend ( xQueueHandle xQueue, const void * pvItemToQueue, portTickType xTicksToWait);
```

Figura 37. Prototipo de la función *xQueueSendToFront(),xQueueSendToBack() y xQueueSend()*.

- **xQueue** el enlace de la cola. Es el valor que devuelve *xQueueCreate()*.
- **pvItemToQueue** puntero al dato que vamos a copiar.
- **xTicksToWait** el número de tiempo máximo que la tarea tiene que estar en estado Blocked para que la cola esté disponible.

El tiempo se da en ticks, si se quiere en ms hay que dividir por portTICK_RATE_MS.

Hay dos posibles valores de retorno:

1. **pdPASS:** si el dato se ha mandado satisfactoriamente a la cola.
2. **ErrQUEUE_FULL:** el dato no se ha podido almacenar en la cola porque está llena.

7.2.1.3. xQueueReceive()

Usamos esta función para recibir un item de una cola y se elimina una vez recibido. [22]

```
xQueueRecieve( xQueueHandle xQueue, const void * pvBuffer, portTickType xTicksToWait);
```

Figura 38. Prototipo de la función *xQueueReceive()*.

- **xQueue:** el enlace de la cola. Es el valor que devuelve xQueueCreate().
- **pvBuffer:** puntero a la memoria donde se va a copiar el dato.
- **xTicksToWait:** el número de tiempo máximo que la tarea tiene que estar en estado Blocked para que la cola esté disponible.

Hay dos posibles valores de retorno:

1. **pdPASS:** si el dato se ha leido satisfactoriamente de la cola.
2. **ErrQUEUE_FULL:** el dato no se ha podido leer porque la cola está vacía.

7.2.2. Semáforos:

El kernel de FreeRTOS proporciona semáforos binarios, semáforos de recuento y exclusiones mutuas con fines de exclusión mutua y sincronización.

Los semáforos binarios solo pueden tener dos valores. Son una buena opción para implementaciones de sincronización (ya sea entre tareas o entre tareas e interrupciones). Los semáforos de recuento tienen más de dos valores. Permiten compartir recursos entre muchas tareas o realizar las operaciones de sincronización más complejas.

Las exclusiones mutuas son semáforos binarios que incluyen un mecanismo de herencia de prioridades. Esto significa que, si una tarea de alta prioridad se bloquea al intentar obtener una exclusión mutua que actualmente está en manos de una tarea de menor prioridad, la prioridad de la tarea que tiene el token se eleva temporalmente a la de la tarea bloqueada. Este mecanismo está diseñado para garantizar que la tarea de mayor prioridad se mantenga en estado bloqueado el menor tiempo posible, a fin de minimizar la inversión de prioridades que ha tenido lugar.[20]

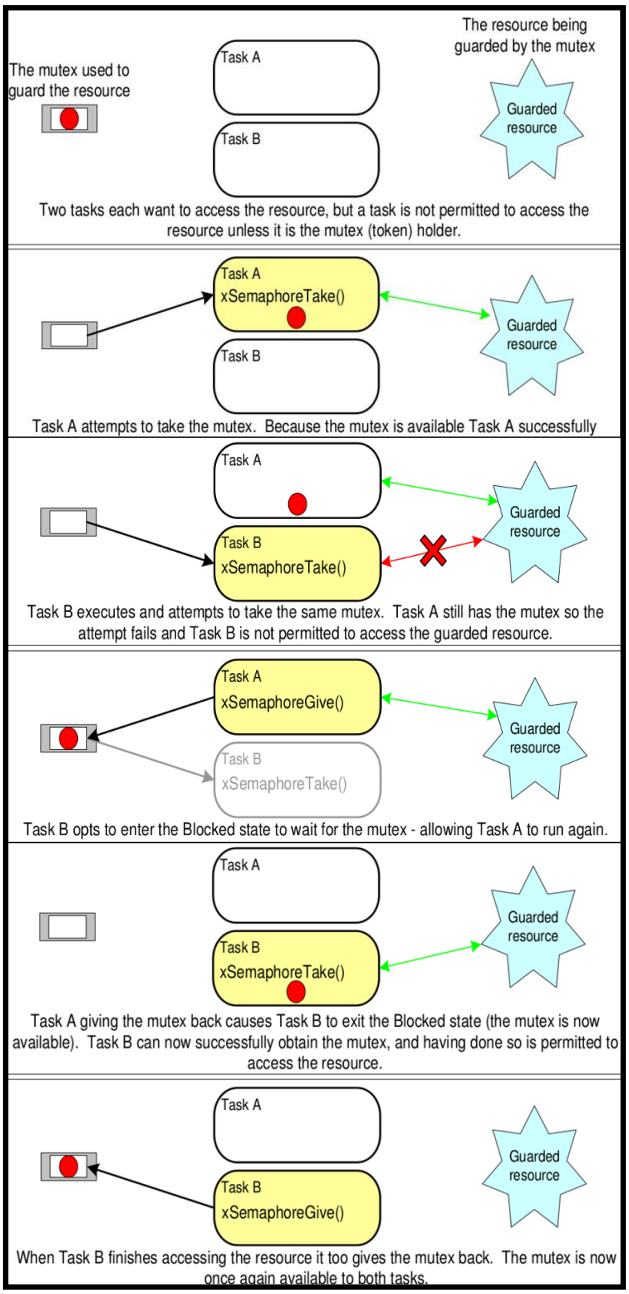

Figura 39. Ejemplo de semáforo mutex. [21]

7.2.2.1. vSemaphoreCreateBinary()

Esta función se utiliza para crear un semáforo. [22]

vSemaphoreCreateBinary(xSemaphoreHandle xSemaphore);

Figura 40. Prototipo de la función *vSemaphoreCreateBinary().*

- **xSemaphore:** Variable de tipo xSemaphoreHandle_t que almacenará el manejador del semáforo que se está creando.

Esta función no devuleve nada. (tipo void)

7.2.2.2. xSemaphoreTake ()

Esta función "toma" u obtiene un semáforo que ha sido previamente creado con éxito.[22]

xSemaphoreTake (*xSemaphoreHandle xSemaphore, TickType_t xTicksToWait*);

Figura 41. Prototipo de la función *xSemaphoreTake ().*

- **xSemaphore:** Variable de tipo xSemaphoreHandle_t que almacenará el manejador del semáforo que se está creando.
- **xTicksToWait:** La cantidad máxima de tiempo que la tarea debe permanecer en el estado Bloqueado para esperar a que el semáforo esté disponible, si el semáforo no está disponible de inmediato.

Hay dos posibles valores de retorno:

1. **pdPASS:** si la operación de "tomar" el semáforo se ha realizado con éxito. Si se especificó un tiempo de bloqueo (xTicksToWait no era cero), entonces es posible que la tarea llamada se coloque en estado Bloqueado para esperar al semáforo si no estaba disponible de inmediato, pero el semáforo estuvo disponible antes de que expirara el tiempo de bloqueo.
2. **pdFAIL:** si la operación de "tomar" el semáforo NO se ha realizado con éxito.

7.2.2.3. xSemaphoreGive ()

Esta función "da" o libera un semáforo que ha sido previamente "tomado" con éxito. [22]

xSemaphoreGive (*xSemaphoreHandle xSemaphore*);

Figura 42. Prototipo de la función *xSemaphoreGive ().*

- **xSemaphore:** Variable de tipo xSemaphoreHandle_t que almacenará el manejador del semáforo que se está creando.

Hay dos posibles valores de retorno:

1. **pdPASS:** si la operación de "dar" el semáforo se ha realizado con éxito.
2. **pdFAIL:** si la operación de "dar" el semáforo NO se ha realizado con éxito.

En el siguiente capítulo se desarrolla la implementación de nuestro proyecto en el sistema operativo anteriormente descrito, FreeRTOS.

8 IMPLEMENTACIÓN EN FREERTOS

En concreto, en este proyecto se han implementado dos tareas, *vSolicitudApertura* y *vActuacionSOL*, y una cola que permite la comunicación entre ellas, *colaActuacion*. Además de un semáforo (*xSemIniciarActuacion*) para la sincronización entre las dos tareas.

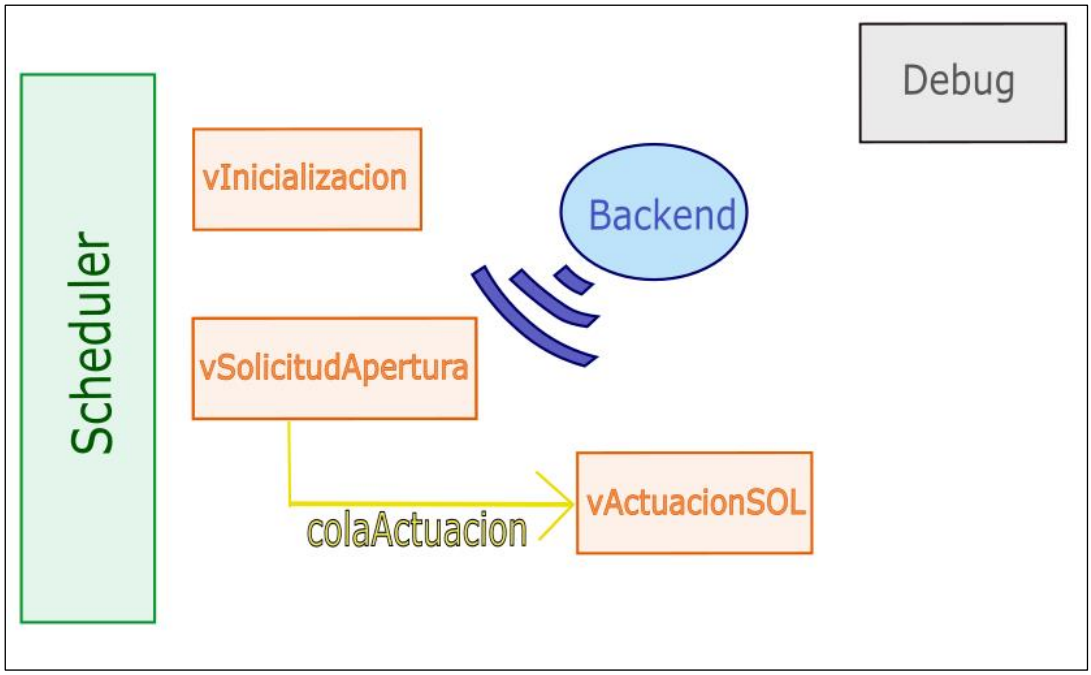

Figura 43. Representación de tareas en FreeRTOS.

Las tareas *vTaskStartScheduler()* y *vDepuracion()* son tareas que se usan en el desarrollo del funcionamiento del RTOS y que son proporcionadas por el propio RTOS y el GIE. El scheduler iniciará todas las tareas creadas, que se ejecutarán según su prioridad y si están listas para ejecutarse o no, y el Debug es la tarea que gestiona el envío de mensajes de depuración vía UART.

Asimismo, en la tarea *vInicializacion ()*, dedicada a iniciar fecha RTC, alarmas RTC y periféricos del microprocesador hemos introducido las funciones descritas en el bloque Inicialización del driver del NBIoT (*Capítulo 6*).

```
/*****************************************************************************
* Task : vInicializacion()
* \b Description:
* @brief Tarea dedicada a iniciar fecha RTC, alarmas RTC y periféricos generales
* del microprocesador.
*****************************************************************************/
void vInicializacion(){
  /* Variables de debug */
  uint8_t NIVEL = 0;
  uint8_t status = 0;
  char CHAR[100] = {0};

  /* Variable de reintentos de inicializacion */
 uint8_t SIM7080_retries = 0;

 /* Variables necesarias para el driver del NBIoT */
 char puerto[] = "7013";
 char ip[] = "86.109.110.101";
 uint8_t OPERADOR = COMISSIONING_VODAFONE;

  /* Variables para control de memoria (en words (4B)) */
  uint16_t uxHighWaterMark = 0;
  uint16_t freeHeap = (uint16_t)xPortGetFreeHeapSize();
  MandaColaDep(VERBOSE_DEBUG, "vInicializacion: Heap disponible: %d \n", freeHeap);
```

```
  /* Estructura de fecha RTC */
  RTC_Struct_Date getConfigDate;
  RTC_Struct_Timer RTC_alarm;

  /* Se accede y obtiene el semáforo de inicio de medida durante la configuración
  inicial, de manera que ninguna otra tarea de medida pueda ejecutarse durante esta
  primera configuración */
  xSemaphoreTake(xSemIniciarActuacion, portMAX_DELAY);

  /* ============================================================================ */
  /* CONFIGURACION FECHA RTC */
  /* ============================================================================ */

  /* Se introduce la fecha si está definido SET_RTC_DATE */
#ifdef SET_RTC_DATE
  /* Configuración fecha RTC    y  m  d  wd  h   m  s  ms */
  RTC_Struct_Date configDate ={RTC_YEAR, RTC_MONTH, RTC_DAY, RTC_WEEKDAY, RTC_HOUR, RTC_MINUTE,
RTC_SECOND, RTC_MILISECOND};
  /* Establece fecha RTC */
  id = RTC_SetDate(configDate);
  MandaColaDep(RTC_DEBUG_VERBOSE[id], RTC_DEBUG_MESSAGE[id], configDate.WeekDay,  configDate.Year,
configDate.Month, configDate.Day,
               configDate.Hours, configDate.Minutes, configDate.Seconds, configDate.Miliseconds);
#endif

  /* ============================================================================ */
  /* CONFIGURA ALARMAS RTC */
  /* ============================================================================ */

  /* Recoge fecha RTC */
  RTC_GetDate(&getConfigDate);

  /* Activa la alarma A */
  id = RTC_SetAlarm(RTC_alarmA, RTC_ALARM_A);
  Debug_SendMessageRTC(getConfigDate, RTC_DEBUG_VERBOSE[id], RTC_DEBUG_MESSAGE[id], RTC_alarmA.Hours,
RTC_alarmA.Minutes, RTC_alarmA.Seconds);

  id = RTC_GetAlarmTime(&RTC_alarm, RTC_ALARM_A);
  Debug_SendMessageRTC(getConfigDate, RTC_DEBUG_VERBOSE[id], RTC_DEBUG_MESSAGE[id], RTC_alarm.Hours,
RTC_alarm.Minutes, RTC_alarm.Seconds);

  /* Activa la alarma B */
  id = RTC_SetAlarm(RTC_alarmB, RTC_ALARM_B);
  Debug_SendMessageRTC(getConfigDate, RTC_DEBUG_VERBOSE[id], RTC_DEBUG_MESSAGE[id], RTC_alarmB.Hours,
RTC_alarmB.Minutes, RTC_alarmB.Seconds);

  id = RTC_GetAlarmTime(&RTC_alarm, RTC_ALARM_B);
  Debug_SendMessageRTC(getConfigDate, RTC_DEBUG_VERBOSE[id], RTC_DEBUG_MESSAGE[id], RTC_alarm.Hours,
RTC_alarm.Minutes, RTC_alarm.Seconds);

  /* ============================================================================ */
  /* CONFIGURACION INICIAL MODULO NB-IoT */
  /* ============================================================================ */

 while (status != SIM7080_INIT_OK){
    status = SIM7080_Init(UART_1, 9600, PWRKEY_Pin1, COMISSIONING_OFF, 0);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }

 while(status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED && SIM7080_retries <= 12){ //12 reintentos
= 2 minutos
    SIM7080_retries++;
    status = SIM7080_CheckNetworkConnection();
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
    vTaskDelay(10000);
  }
  SIM7080_retries = 0;

 /* Comprueba si hay conexion forzando operador Vodafone */
 if(status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED){
    status = SIM7080_Comissioning(OPERADOR);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
    vTaskDelay(3000/portTICK_PERIOD_MS);

    /* Si no ha cogido conexion forzando operador, establece busqueda automatica */
    if(status != SIM7080_COMISSIONING_OK && status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED){
      status = SIM7080_Comissioning(COMISSIONING_AUTO);
      SIM7080_Debug(status,&NIVEL, CHAR);
      MandaColaDep(NIVEL, CHAR);
```

```
      vTaskDelay(3000/portTICK_PERIOD_MS);

      /* Comprueba conexión */
      while(SIM7080_retries <= 30 && status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED){
        status = SIM7080_CheckNetworkConnection();
        SIM7080_Debug(status,&NIVEL, CHAR);
        MandaColaDep(NIVEL, CHAR);
        SIM7080_retries++;
        vTaskDelay(10000);
        if(SIM7080_retries >= 30)
        {
          MandaColaDep(VERBOSE_FATAL, "No se pudo conectar el modulo SIM7080 \n");
        }
      }
    }
  }

  /* Inicia red */
  SIM7080_retries = 0;
  while(status != SIM7080_INIT_NETWORK_CONNECTED_ROAMMING && status !=
SIM7080_INIT_NETWORK_CONNECTED_HOME_NETWORK ){
    status = SIM7080_InitNetwork();
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
    SIM7080_retries++;
    if (SIM7080_retries >= 9){
      SIM7080_Reboot();
      _SIM7080_Delay(2000);
      SIM7080_WakeUp();
    }
    vTaskDelay(3000);
  }


  /* Comprueba si esta conectado a red */
  SIM7080_retries = 0;
  status = SIM7080_CHECK_NETWORK_CONNECTION_NOT_CONNECTED;
  while (status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED && SIM7080_retries <= 20  )
  {
    /* Damos 1 min de margen para que se conecte */
    if (SIM7080_retries >= 20){
      MandaColaDep(VERBOSE_ERROR, "vInicializacion: No se pudo conectar a la red.\n");
    }

    /* Comprobamos conexion cada 3 segundos */
    status = SIM7080_CheckNetworkConnection();
    SIM7080_retries++;
    vTaskDelay(3000);
  }

  MandaColaDep(VERBOSE_SUCCESS, "vInicializacion: Conectado a la red\n");

  /* Inicia conexion TCP */
  SIM7080_retries = 0;
  while ((status != SIM7080_OPEN_CONNECTION_OK) & SIM7080_retries <= 3){
    /* Damos 1 reintento mas para que se conecte al topic */
    if (SIM7080_retries >= 3)
    {
      MandaColaDep(VERBOSE_ERROR, "No se ha podido iniciar la transmision.\n");
    }
    /* Reintentamos cada 3 segundos */
    vTaskDelay(3000);
    SIM7080_retries++;
    status = SIM7080_InitConnection(SIM7080_UDP, puerto, ip, 0);
    CHAR[0]='\0';
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }

  /* Se devuelve el semáforo de manera que las demás tareas obtendrían acceso para
  ejecutar su función */
  xSemaphoreGive(xSemIniciarActuacion);

  /* Suspende la tarea, no queremos ejecutar nada cíclicamente (no hay código en
  el bucle for(;;) */
  vTaskSuspend(NULL);

  for(;;){} // Nunca debería llegar aquí
}
```

Figura 44. Código de la tarea *vInicializacion()*.

vSolicitudApertura()

La tarea *vSolicitudApertura ()* se encarga de mandar el mensaje uplink a través del módulo NBIoT a su backend mediante la función *SIM7080_TransmitDatagram ()*. A continuación, recibimos la respuesta del servidor por el socket 0 con la función *_SIM7080_SendATCommand ()*. Por ultimo, mediante la función *_SIM7080_GetSubstring ()* obtenemos de la respuesta la información que nos interesa y comprobamos si corresponde con un '2' o no. Si es así, mandaremos un mensaje de éxito a la cola de depuración y la respuesta por la cola *colaActuacion* mediante *xQueueSendToBack*.

Esta tarea corresponde con los bloques de Solicitud y Respuesta del driver de NBIoT descritos en el *capítulo 6*.

A continuación se muestra el código de la tarea *vSolicitudApertura()*.

```
void vSolicitudApertura(){
 /* Variables de debug */
  uint8_t NIVEL = 0;
  uint8_t status = 0;
  char CHAR[100] = {0};
 /* Cadena de caracteres para respuesta NBIoT */
  char respuesta[100];
  char substring[10];

  RTC_Struct_Date getConfigDate;
  xSemaphoreTake(xSemIniciarActuacion, portMAX_DELAY);
  RTC_GetDate(&getConfigDate);

  /* Envia trama en socket 0 */
  SIM7080_TransmitDatagram(0, "1");
  /* Recibe respuesta del servidor en socket 0 */
  _SIM7080_SendATCommand("AT+CARECV=0,1", "OK", respuesta, 100, 1000);
  /* Comprobacion de la substring, comprobando si el mensaje recibido es 2.*/
  _SIM7080_GetSubstring(respuesta, "+CARECV: 1,", "\r\n", substring);
  if (substring[0] == '2'){
    Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Respuesta servidor correcta.\n");
    MandaColaDep(VERBOSE_SUCCESS, "Respuesta servidor correcta.\n");

    id = xQueueSendToBack(colaActuacion, ( void * ) &substring, ( TickType_t )
portMAX_DELAY);
    if( id /*xQueueSend(colaActuacion,( void * ) &substring,( TickType_t ) portMAX_DELAY */
!= pdPASS )
       {
         Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Error al enviar por cola");
       }
    else Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Envio por cola correcto");
    }

  else{
    Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Respuesta servidor erronea.\n");
    MandaColaDep(VERBOSE_ERROR, "Respuesta servidor erronea.\n");
  }

 while (status != SIM7080_CLOSE_NETWORK_OK && status !=
SIM7080_CLOSE_CONNECTION_SOCKET_WAS_CLOSED){
    status=SIM7080_CloseConnection(0);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }

 while (status != SIM7080_CLOSE_NETWORK_OK && status != SM7080_SET_POWER_MODE_PSM){
    status=SIM7080_SetPowerMode(SIM7080_PSM_MODE);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }
  vTaskSuspend(NULL);

  for(;;){}
}
```

Figura 45. Código de la tarea *vSolicitudApertura()*.

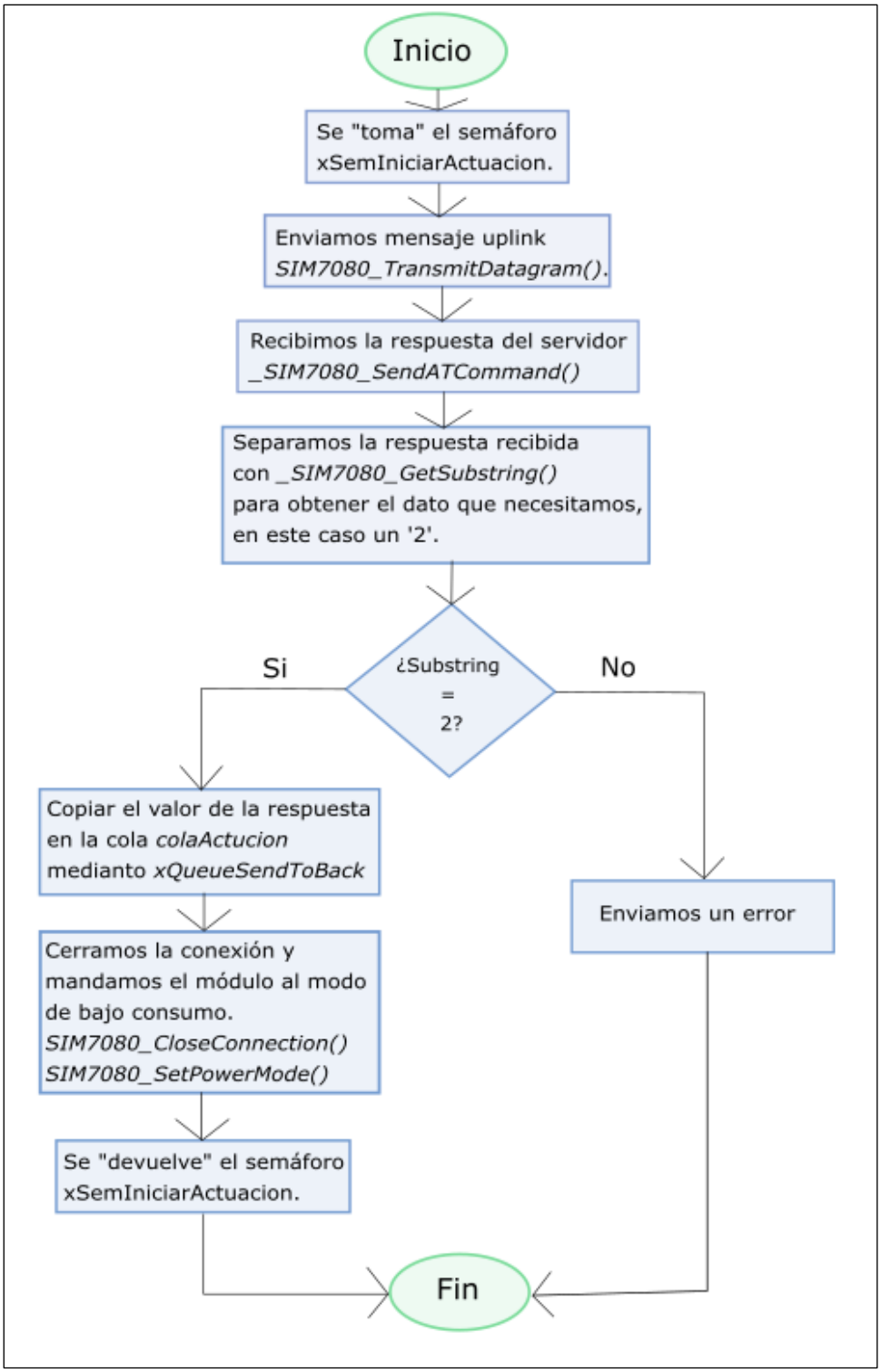

Figura 46. Diagrama tarea *vSolicitudApertura()*.

vActuacionSOL()

Esta tarea se encargará de todo lo relacionado con la actuación del solenoide. Va a recibir por la *colaActuacion* la respuesta que previamente ha mandado a dicha cola la tarea *vSolicitudApertura()*. Va a comprobar que el mensaje recibido por la cola es efectivamente un '2'. Si esto así ocurre, y es correcto, procederá a activar el solenoide escribiendo un uno por el pin con la función *SOL_Write_PIN ()*.

Después de esto, esperará un delay de medio segundo y pondrá a 0 el pin para así desactivar el actuador.

Asimismo, se enviarán los correspondientes mensajes de depuración "Pin activado con éxito", "Pin desactivado con éxito" y "Respuesta errónea" según corresponda y se hará a través de la función *MandaColaDep*. Cuando se quiere enviar un mensaje de depuración desde dentro de una tarea, no se puede enviar directamente a través de la función *Debug_SendMessage*, sino que se envían a la cola *colaDepuracion* para ser enviados por la tarea *vDepuracion*.

A continuación se muestra el código de la tarea *vActuacionSOL()*.

```
void vActuacionSOL(){

  char correcto[10]="";

  for(;;){
    if (xQueueReceive(colaActuacion, &correcto, 2000 )  == pdPASS)
    {
      if (correcto[0] == '2'){
        id=SOL_Write_PIN(configPIN,1); //ponemos pin a 1
        if(id==SOL_WRITE_OK){
          //Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Pin activado con éxito.\n");
          MandaColaDep(VERBOSE_SUCCESS, "Pin activado con éxito\n");
        }

        vTaskDelay(500); //esperamos
        id=SOL_Write_PIN(configPIN, 0); //apagamos (pin a 0)
        if(id==SOL_WRITE_OK){
          //Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Pin desactivado con
éxito.\n");
          MandaColaDep(VERBOSE_SUCCESS, "Pin desactivado con éxito\n");
        }
      }
      else{
        //Debug_SendMessageRTC(getConfigDate, VERBOSE_INFO ,"Respuesta incorrecta.\n");
        MandaColaDep(VERBOSE_ERROR, "Respuesta erronea.\n");
      }
    }
  }
}
```

Figura 47. Código de la tarea *vActuacionSOL()*.

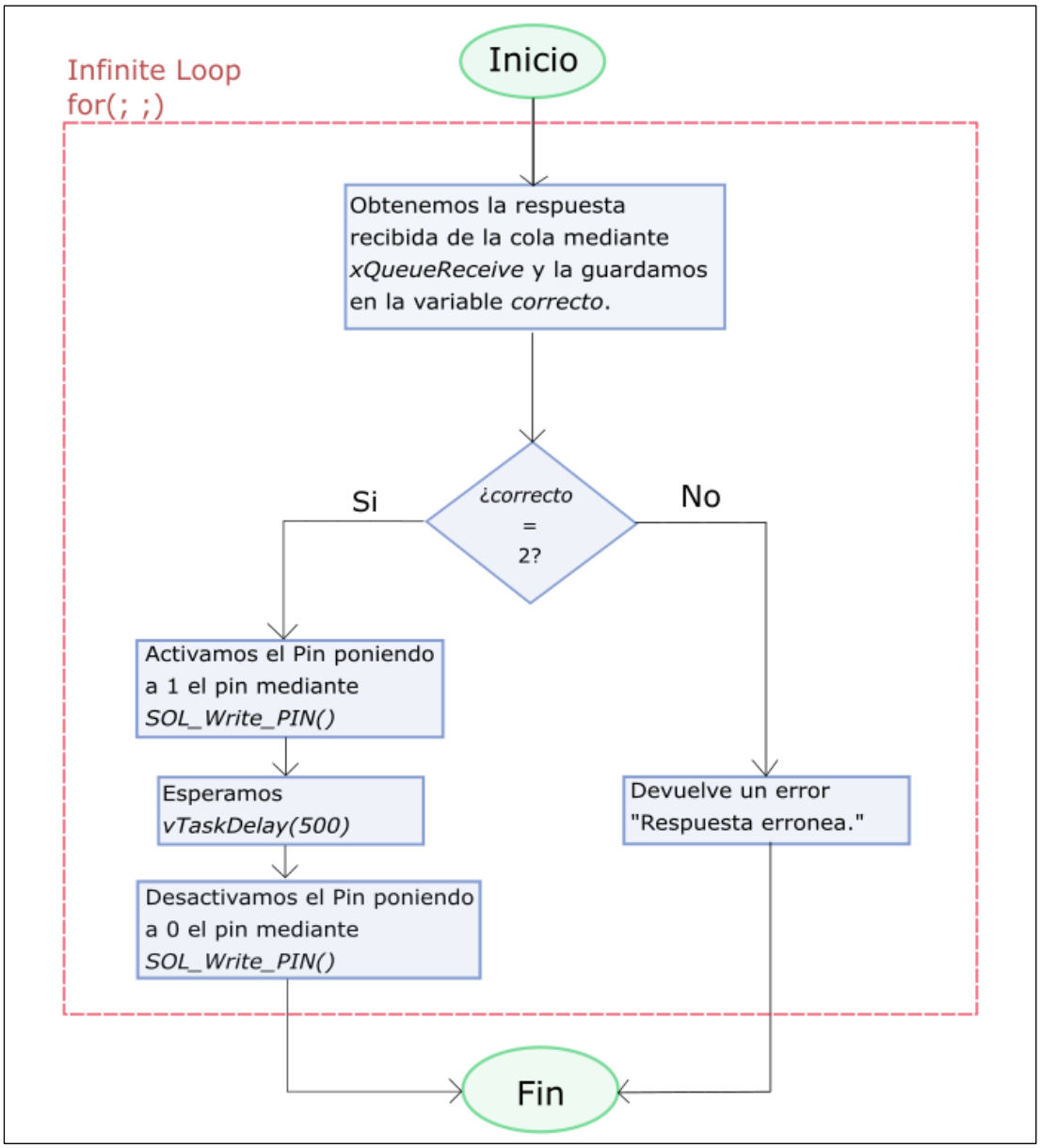

Figura 48. Diagrama tarea *vActuacionSOL()*.

A continuación, se valida el funcionamiento de nuestro sistema con la ayuda del software Docklight y se exponen los resultados de manera que quede demostrada la viabilidad de del mismo.

9 PRUEBAS DE VALIDACIÓN

Para validar el correcto funcionamiento de nuestro sistema y su implementación en FreeRTOS vamos a realizar una prueba desde el entorno de desarrollo IAR Embedded Workbech, monitorizando la comunicación entre el microcontrolador y el módulo de NB-IoT con el software Docklight.

El sistema operativo va a implementar una rutina entre las tareas *vSolicitudApertura* y *vActuacion SOL*, además de las propias de FreeTOS(Schedule, Debug…) y la comunicación entre ellas mediante la cola *colaActuacion*.

Vamos a ver, por un lado, la comunicación entre el módulo NB-IoT y el servidor, lo que correspondería a la tarea de solicitud. Y por otro, lo que está pasando paralelamente en el micro viendo sus mensajes de depuración.

```
27/01/2022 12:23:38.486 [COM6] - AT<CR><LF>

27/01/2022 12:23:38.502 [COM7] - AT<CR>
<CR><LF>
OK<CR><LF>

27/01/2022 12:23:39.590 [COM6] - AT+IPR=9600<CR><LF>

27/01/2022 12:23:39.607 [COM7] - AT+IPR=9600<CR>

27/01/2022 12:23:39.718 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:23:40.709 [COM6] - ATE0<CR><LF>

27/01/2022 12:23:40.710 [COM7] - ATE0<CR>
<CR><LF>
OK<CR><LF>

27/01/2022 12:23:41.813 [COM6] - AT+CPSMS=0<CR><LF>

27/01/2022 12:23:41.840 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:23:42.933 [COM6] - AT+CEDUMP=1<CR><LF>

27/01/2022 12:23:42.949 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:23:44.037 [COM6] - AT+CMNB=2<CR><LF>

27/01/2022 12:23:44.053 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:23:45.157 [COM6] - AT+CBANDCFG="NB-IOT",3,8,20<CR><LF>

27/01/2022 12:23:45.189 [COM7] - <CR><LF>
OK<CR><LF>
<CR><LF>
DST: 0<CR><LF>
<CR><LF>
*PSUTTZ: 22/01/20,13:12:35","+04",0<CR><LF>
```

Uso de comando At para confirmar la comunicación.

Establece el baud rate a 9600 baudios.

Desactiva el modo eco.

Deshabilita el uso del modo de ahorro de energía.

Establece que el módulo se reseteará si se bloquea.

Elige que tecnología usar, en este caso NB-IoT.

Configura la banda de NB-IoT.

```
27/01/2022 12:23:48.293 [COM6] - AT+CCID<CR><LF>

27/01/2022 12:23:48.319 [COM7] - <CR><LF>
89882390000057258850<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:23:49.301 [COM6] - AT+CGATT?<CR><LF>

27/01/2022 12:23:49.317 [COM7] - <CR><LF>
+CGATT: 1<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:00.308 [COM6] - AT+CGREG?<CR><LF>

27/01/2022 12:24:00.333 [COM7] - <CR><LF>
+CGREG: 0,5<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:01.332 [COM6] - AT+CGREG=1<CR><LF>

27/01/2022 12:24:01.348 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:24:02.436 [COM6] - AT+CNACT?<CR><LF>

27/01/2022 12:24:02.470 [COM7] - <CR><LF>
+CNACT: 0,0,"0.0.0.0"<CR><LF>
+CNACT: 1,0,"0.0.0.0"<CR><LF>
+CNACT: 2,0,"0.0.0.0"<CR><LF>
+CNACT: 3,0,"0.0.0.0"<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:03.556 [COM6] - AT+CNACT=0,1<CR><LF>

27/01/2022 12:24:03.587 [COM7] - <CR><LF>
OK<CR><LF>
<CR><LF>
+APP PDP: 0,ACTIVE<CR><LF>

27/01/2022 12:24:07.667 [COM6] - AT+CGREG?<CR><LF>

27/01/2022 12:24:07.693 [COM7] - <CR><LF>
+CGREG: 1,5<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:11.683 [COM6] - AT+CGATT?<CR><LF>

27/01/2022 12:24:11.710 [COM7] - <CR><LF>
+CGATT: 1<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:18.691 [COM6] - AT+CNACT?<CR><LF>

27/01/2022 12:24:18.731 [COM7] - <CR><LF>
+CNACT: 0,1,"100.93.0.41"<CR><LF>
+CNACT: 1,0,"0.0.0.0"<CR><LF>
+CNACT: 2,0,"0.0.0.0"<CR><LF>
+CNACT: 3,0,"0.0.0.0"<CR><LF>
<CR><LF>
OK<CR><LF>
```

Muestra el ICCID.

Pregunta por el estado de conexión GPRS. En este caso responde 1, conectado.

Pregunta por el estado de registro de la red.

Habilita el código de resultado registro no solicitado del registro de red.

Pregunta por la activación de la res de aplicaciones. Ninguna activa.

Activa una red de aplicaciones.

Pregunta por el estado del registro de red. Recive un registro de red roaming disponible.

Pregunta por la conexión al servicio GPRS y recibe confirmación de que esta conectado.

Pregunta por la activación de la res de aplicaciones. Recive una activa y su dirección IP.

```
27/01/2022 12:24:19.811 [COM6] - AT+CASTATE?<CR><LF>

27/01/2022 12:24:19.827 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:24:20.819 [COM6] - AT+CAOPEN=0,0,"UDP","86.109.110.101",7013<CR><LF>

27/01/2022 12:24:20.889 [COM7] - <CR><LF>
+CAOPEN: 0,0<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:26.883 [COM6] - AT+CASTATE?<CR><LF>

27/01/2022 12:24:26.904 [COM7] - <CR><LF>
+CASTATE: 0,1<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:27.890 [COM6] - AT+CASEND=0,1,9000<CR><LF>

27/01/2022 12:24:27.922 [COM7] - <CR><LF>
>
27/01/2022 12:24:28.914 [COM6] - 1<CR><LF>

27/01/2022 12:24:28.936 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:24:29.522 [COM7] - <CR><LF>
+CADATAIND: 0<CR><LF>
<CR><LF>
+CADATAIND: 0<CR><LF>

27/01/2022 12:24:30.914 [COM6] - AT+CAACK=0<CR><LF>

27/01/2022 12:24:30.950 [COM7] - <CR><LF>
+CAACK: 1,0<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:31.938 [COM6] - AT+CARECV=0,1<CR><LF>

27/01/2022 12:24:31.968 [COM7] - <CR><LF>
+CARECV: 1,2<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:32.962 [COM6] - AT+CASTATE?<CR><LF>

27/01/2022 12:24:32.982 [COM7] - <CR><LF>
+CASTATE: 0,1<CR><LF>
<CR><LF>
OK<CR><LF>

27/01/2022 12:24:33.970 [COM6] - AT+CACLOSE=0<CR><LF>

27/01/2022 12:24:34.002 [COM7] - <CR><LF>
OK<CR><LF>

27/01/2022 12:24:34.994 [COM6] - AT+CASTATE?<CR><LF>

27/01/2022 12:24:35.010 [COM7] - <CR><LF>
OK<CR><LF>
```

Consultar estado de conexión TCP/UDP.

Abre una conexión del tipo UDP.

Pregunta por el estado de conexión UDP y verifica que esta conectada a un servidor remoto.

Manda un dato de tamaño 1byte por la conexión establecida UDP (0) con un input time de 9000 milisegundos.

Consulta que la información enviada sea correcta.

Recibe datos por la conexión UDP (0) establecida. Establece el tamaño del dato que espera recibir a 1 byte.

Consultar estado de conexión TCP/UDP.

Cierra la conexión UDP.

Consultar estado de conexión TCP/UDP.

```
27/01/2022 12:24:37.026 [COM6] - AT+CPSMS=1,,,"01011111","00000001"<CR><LF>

27/01/2022 12:24:37.074 [COM7] - <CR><LF>
OK<CR><LF>
+APP PDP: 0,DEACTIVE<CR><LF>
<CR><LF>
+CPSMSTATUS: "ENTER PSM"<CR><LF>
```

Establece el modo de ahorro de energía.

Figura 49. Comunicación Docklight con el módulo NB-IoT.

Los comandos usados se detallan en el *Anexo*.

```
Mon, Jan 01, 200 00:03:25.984 [INFO]: ================================ <LF>
Mon, Jan 01, 200 00:03:25.984 [INFO]: INICIO PUERTO DEPURACION <LF>
Mon, Jan 01, 200 00:03:25.984 [INFO]: ================================ <LF>
Mon, Jan 01, 200 00:03:25.027 [INFO]: RTC: RTC iniciado correctamente <LF>
Mon, Jan 01, 200 00:03:25.035 [INFO]: IDWG: IWDG iniciado correctamente con valor de timeout de
28000 ms <LF>
Mon, Jan 01, 200 00:03:25.046 [INFO]: IDWG: Habilada depuracion, congelando el IWDG cuando este
en debug <LF>
Mon, Jan 01, 200 00:03:25.062 [DEBUG]: RTC: Alarma A del RTC configurada - 0:0:50 <LF>
Mon, Jan 01, 200 00:03:25.062 [DEBUG]: RTC: La proxima alarma A sera a las 0:4:15 <LF>
Mon, Jan 01, 200 00:03:25.062 [DEBUG]: RTC: Alarma B del RTC configurada - 0:0:30 <LF>
Mon, Jan 01, 200 00:03:25.062 [DEBUG]: RTC: La proxima alarma B sera a las 0:3:55 <LF>
Mon
20/01/2022 12:23:21.034 [RX] -, Jan 01, 200 00:03:25.101 [DEBUG]: vInicializacion: Heap
disponible: 26248 <LF>
Mon, Jan 01, 200 00:03:25.109 [INFO]: IWDG refrescado<LF>

20/01/2022 12:23:41.017 [RX] - Mon, Jan 01, 200 00:03:47.566 [INFO]: IWDG refrescado<LF>

20/01/2022 12:23:47.628 [RX] - Mon, Jan 01, 200 00:03:55.000 [DEBUG]: RTC: Alarma B del RTC
configurada - 0:0:30 <LF>

20/01/2022 12:23:48.289 [RX] - Mon, Jan 01, 200 00:03:55.742 [DEBUG]: SIM7080 Init: Modulo
inicializado correctamente<LF>

20/01/2022 12:23:49.303 [RX] - Mon, Jan 01, 200 00:03:56.882 [DEBUG]: SIM7080 SIM ID: SIM ID
encontrado correctamente. SIM ID = 89882390000057258850<LF>

20/01/2022 12:23:50.304 [RX] - Mon, Jan 01, 200 00:03:58.007 [DEBUG]: SIM7080 Check Connection:
Modulo conectado a la red<LF>

20/01/2022 12:24:01.038 [RX] - Mon, Jan 01, 200 00:04:10.078 [INFO]: IWDG refrescado<LF>

20/01/2022 12:24:05.414 [RX] - Mon, Jan 01, 200 00:04:15.000 [DEBUG]: RTC: Alarma A del RTC
configurada - 0:0:50 <LF>

20/01/2022 12:24:08.674 [RX] - Mon, Jan 01, 200 00:04:18.667 [INFO]: SIM7080 Init Network:
Iniciada conexion roaming<LF>

20/01/2022 12:24:14.301 [RX] - Mon, Jan 01, 200 00:04:25.000 [DEBUG]: RTC: Alarma B del RTC
configurada - 0:0:30 <LF>

20/01/2022 12:24:15.689 [RX] - Mon, Jan 01, 200 00:04:26.562 [SUCCESS]: vInicializacion:
Conectado a la red<LF>

20/01/2022 12:24:21.064 [RX] - Mon, Jan 01, 200 00:04:32.609 [INFO]: IWDG refrescado<LF>

20/01/2022 12:24:27.906 [RX] - Mon, Jan 01, 200 00:04:40.308 [DEBUG]: SIM7080 Open TCP/UDP
connection: Apertura del socket correcta<LF>

20/01/2022 12:24:32.940 [RX] - Mon, Jan 01, 200 00:04:40.281 [INFO]: Respuesta servidor
correcta<LF>
Mon, Jan 01, 200 00:04:40.281 [INFO]: Envio por cola correcto<LF>
Mon, Jan 01, 200 00:04:46.000 [SUCCESS]: Respuesta servidor correcta<LF>
Mon, Jan 01, 200 00:04:46.007 [ERROR]: Envio por cola correcto<LF>
Mon, Jan 01, 200 00:04:46.015 [SUCCESS]: Pin activado con exito<LF>

20/01/2022 12:24:36.015 [RX] - Mon, Jan 01, 200 00:04:49.429 [DEBUG]: SIM7080 Close Connection:
Socket cerrado correctamente<LF>
```

```
20/01/2022 12:24:37.057 [RX] - Mon, Jan 01, 200 00:04:50.601 [DEBUG]: SIM7080 Close Connection:
Socket fue cerrado previamente<LF>

20/01/2022 12:24:37.983 [RX] - Mon, Jan 01, 200 00:04:51.644 [SUCCESS]: Pin desactivado con
exito<LF>
Mon, Jan 01, 200 00:04:51.726 [INFO]: SIM7080 Set Power Mode: Establecido modo consumo PSM<LF>
```

Figura 50. Comunicación Docklight con el microcontrolador.

La latencia del módulo NB-IoT es de 5,006 segundos, desde que se envía el comando hasta que recibe la respuesta del servidor. Pero se puede optimizar hasta llegar a 2,065 segundos si omitimos el comando 'AT+CAACK' que consulta si la información enviada es correcta.

Por otro lado, la del sistema es de un segundo aproximadamente (0,988s), desde que el servidor envía la orden hasta que se activa el pin.

En este capítulo podemos comprobar que lo desarrollado en este proyecto ha cubierto los requisitos para el funcionamiento esperado.

En el siguiente, y utlimo capitulo, se redactan las coclusiones obtenidas a lo largo de este proyecto y posibles líneas futuras de mejora del mismo.

10 CONCLUSIONES Y LÍNEAS FUTURAS

La posibilidad de desarrollar este proyecto, a nivel personal y académico, me ha permitido conocer el mundo de los sistemas inalámbricos y las redes de comunicación, así como profundizar en el Internet de las Cosas y descubrir todas las aplicaciones y ámbitos en los que existe, y que sin duda iran a más por su crecimiento imparable. Por otro lado, me ha permitido adquirir conocimientos sobre la programación en C que desconocía y sobre todo en FreeRTOS, el cual he considerado muy interesante y valido para mi futuro laboral.

A nivel conceptual, con el desarrollo de este proyecto se ha comprobado que el desarrollo del 'Internet de las cosas' se ha convertido en la mejor alternativa a los sistemas cableados permitiendo implementar infinitas soluciones mejorando considerablemente el coste y la eficiencia energética de estas. Estas características añadidas a su monitorización remota hacen que cada vez más dispositivos de nuestro entorno estén conectador a internet y entre ellos mismos permitiéndonos crear aplicaciones que faciliten nuestra vida cada vez más. Estas aplicaciones abarcan casi cualquier campo de la vida cotidiana que podamos imaginar.

Una vez conocido este 'Internet de las cosas', se ha realizado un estudio del estado actual de los dispositivos de precintado remoto justificando así este Trabajo de Fin de Grado. Nos hemos enfocado en las cerraduras inteligentes o de precintado remoto, en como funcionan, que tipos hay y en cómo nos facilitan el día a día.

Tras la justificación del proyecto hemos realizado un segundo estudio del estado de las redes inalámbricas, en concreto de las LPWAN, como son LTE-M, LoRaWAN, NB-IoT y Sigfox. Vamos a compararlas analizando sus características, pero, sobre todo, vamos a enforcarnos en el modo que tiene de recibir la información de un servidor o de otros dispositivos.

Posteriormente, se han detallado los módulos que se han usado para llevar a cabo la implementación del proyecto. Como son el microcontrolador STM32L152RE destacado por su ultra bajo consumo, CPU de 32 bits, alimentación de 3.3V y 5V, 11 perifericos de comunicación y una memoria RAM de hasta 80 Kbytes. Su función es la de comunicarse con el módulo de NB-IoT y soportar el programa de FreeRTOS desarrollado.

El segundo sería el actuador solenoide DSML-0224-12, de 12V de alimentación que se encarga de demostrar la viabilidad del proyecto. En tercer lugar, el módulo de NB-IoT, caracterizado por su comunicación bidireccional con el servidor seleccionado y la capacidad de extensión con interfaces como UART, GPIO e I2C. Es controlado mediante comandos AT, y es el encargado de la comunicación entre el usuario y la nube, y su principal función es enviar y recibir del servidor los datos.

Por último, su utiliza un módulo TTL, FT4232H-56Q Mini Module, para interceptar los mensajes que transmite/recibe módulo NB-IoT, así como los del microcontrolador a modo de depuración, y mostrarlos en la pantalla usando el programa Docklight.

Para la implementación se han realizado un *setup* con los dispositivos mencionados anteriormente añadiendo una batería de 12V para poder alimentar el actuador.

Posteriormente, se ha desarrollado un driver para controlar el actuador en el cual se implementan las funcionalidades que permiten, configurar el pin GPIO al que corresponderá el mismo, leer del pin, escribir en el pin y actuar según nos indique la información que recibamos de la UART.

Tras esto, se elabora el programa de FreeRTOS que se debe ejecutar en el microcontrolador, que cuenta con las tareas destinadas a la inicialización del módulo, a la solicitud (de la atuacion) y a la propia actuación del solenoide. Todas ellas cuentan con las funciones desarrolladas tanto en el driver de control del actuador previamente citado, y del driver de NB-IoT cedido por el GIE.

Por último, se ha validado dicho programa y se han expuesto los resultados de manera que queda demostrada la viabilidad de del mismo.

La realización de este proyecto deja la puerta abierta a un sinfín de propuestas y mejoras, por la simple razón de que la tecnología basada en IoT se encuentra en un continuo proceso de mejora y desarrollo.

Principalmente, la idea de este proyecto era la creación de un sistema de precintado remoto que pudiera ser integrado en una aplicación IoT. La línea que habría que seguir para continuar desarrollando y mejorándolo sería la creación de una App, móvil, por ejemplo, para poder controlar nosotros mismos cuando hacer uso o no del sistema. Conectariamos en este caso la App al servidor para gestionar la información y esta controlaría la actuación mediante, por ejemplo, bluetooth.

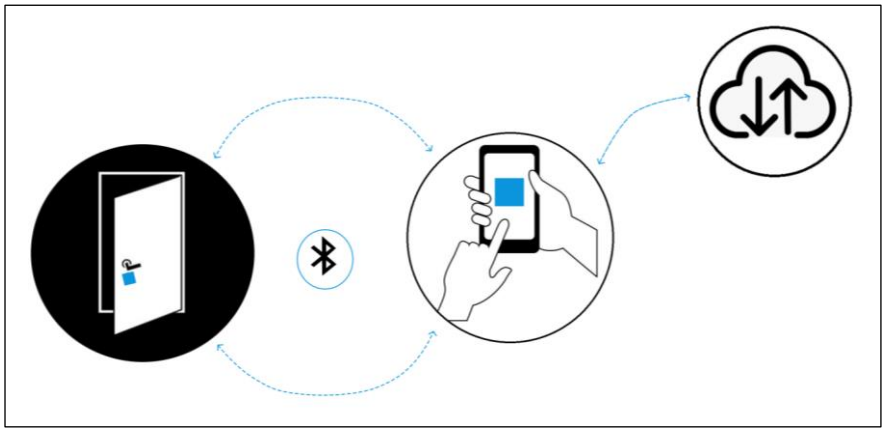

Figura 51. Esquema mejora con App.

Los ámbitos de aplicación de este son cualquiera que requiera abrir puertas, cerraduras o ventanas. Como pueden ser abrir nuestra puerta de casa, cajas fuertes, la taquilla del gimnasio, el cofre de la moto, el garaje….

Una mejora o variante que se podría desarrollar sería no solo el control del actuador para abrir o cerrar el dispositivo, sino también añadir al sistema un sensor para que nos permitiera saber el estado de este. Con esta función podríamos usarlo para la detección de apertura de ventanas y puertas para prevenir intrusos.

Otra sería la gestión de la app. Podría permitir elegir como se abre la puerta, si por proximidad del dispositivo móvil a la misma, o tocando la pantalla. Que personas pueden abrir, si el propietario de la cerradura o autorizar otro para que hagan uso de ella, etc.

Una de las mejoras más importantes seria la mejora del actuador o cerradura. No va a ser el mismo si se va a abrir una puerta de casa, un garaje comunitario u oficina, una habitación de hotel o una caja fuerte. Según la función que se quiera cubrir el dispositivo tendrá que ser de menor o mayor tamaño y disponer de mayor o menor seguridad.

En resumen, la aplicación de este proyecto esta horientada este ámbito pero como hemos dicho, debido a el rápido crecimiento y continuo desarrollo de la tecnología y en concreto, del IoT, puede ir mejorándose e implementándose en cualquier otro.

ANEXO

¿Qué es un actuador solenoide y para qué sive?

Un *solenoide* es cualquier dispositivo físico capaz de crear un campo magnético sumamente uniforme e intenso en su interior, y muy débil en el exterior.

Un ejemplo puede ser el de una bobina de hilo conductor aislado y enrrollado helicoidalmente. En este caso el campo magnético sería unifrome en su interior y fuera sería nulo.

Por ende, un *actuador solenoide* o *actuador electromagnético* es un dispositivo constituido por una bobina junto con un perno central (ferromagnético), los cuales son independientes entre sí. Este actuador es capaz de generar movimiento lineal directamente, sin necesidad de mecanismos o engranajes. Esto es gracias a que, cuando se le da energía a la bobina, y una corriente eléctrica fluye a través de ella, esta genera un campo magnético cuyos polos están determinados por la dirección del flujo de dicha corriente. Como el perno es de un material ferrmagnético, este campo lo atraerá hacia el centro del bobinado provocando su desplazamiento. [24]

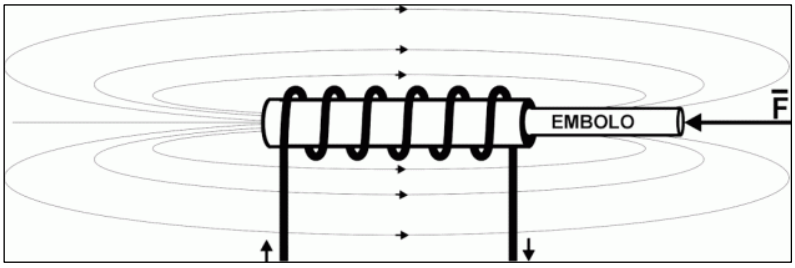

Figura 52. Esquema funcional actuador solenoide. [23]

Normalmente dispone también de un resorte entre el cuerpo de la bobina y el perno central, para que, al cesar la corriente, este vuelva a su posición original.

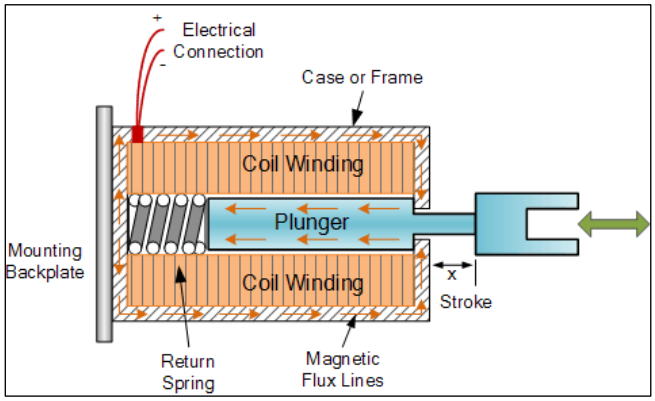

Figura 53. Esquema general actuador solenoide. [23]

La tensión de alimentación de estos actuadores depende de su tamaño, aunque los más populares son los de 5V, 12V y 24V. Por otro lado, su consumo varía entre 100mA para los más pequeños, hasta 1-2 A para los más grandes.

Estos solenoides tienen la ventaja de realizar movimientos rápidos y ser muy sencillos de manejar y mantener. La fuerza que pueden ejercer va desde gramos-fuerza hasta kilogramos-fuerza. Esta depende de la cantidad de espiras que tenga el embobinado, a más espiras, mayor será la fuerza inducida.

Sin embargo, la fuerza que ejerce un solenoide no es lineal, es decir, ejerce menos fuerza cuando el perno está extendido, que cuando está cerca de su posición final. Otra desventaja sería que no se dispone de

ningún tipo de control sobre la velocidad o la posición del perno, simplemente podemos tirar o empujar una carga. [24]

Existen las siguientes configuraciones de solenoides:

- De tiro o disparo: Son los más comunes. El embolo se desplaza mientras se energice la bobina, y cuando esto no ocurra vuelve a su posición inicial.
- De cierre: En estos, los émbolos contienen un imán permanente. Cuando se energiza el embobinado el émbolo se mueve y cuando llega hasta el final de su carrera el imán lo mantiene ahí, incluso cuando deja de pasar corriente a través del embobinado. Para regresar el émbolo a la posición original, hay que energizar nuevamente el embobinado inversamente, esto es, invertir la polaridad de las conexiones.
- Rotatorios: Aquí el émbolo gira un ángulo fijo (comúnmente entre los 25° y 90°) en vez de moverse linealmente.

Los parámetros más importantes de un actuador solenoide son:

- Carrera del émbolo: Es la distancia que recorre el perno (o émbolo) desde su posición inicial hasta la final.
- Fuerza: Fuerza con la que actúa el émbolo al ser energizado.
- Ciclo de trabajo (Duty Cycle): Es el tiempo que el solenoide puede estar energizado a su voltaje nominal sin sobrecalentarse. Por eso hay un tiempo que puede estar energizado y hay un tiempo en el que el solenoide está apagado para disipar parte del calor generado antes de volver a energizarse otra vez

El ciclo de trabajo se puede calcular con la siguiente expresión:

$$Duty\ Cycle = \frac{T_{ON}}{T_{ON} + T_{OFF}} \cdot 100\%$$

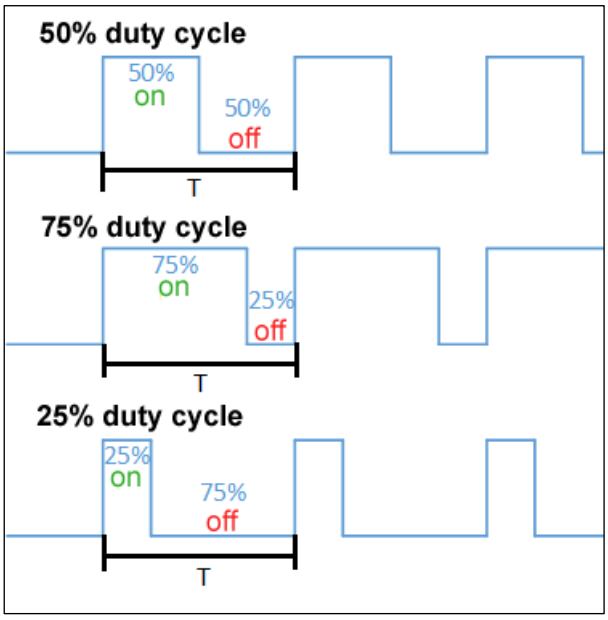

Figura 54. Ejemplos de Duty Cycle. [25]

Código de las funciones

En esta sección mostramos el código correspondiente tanto del driver del solenoide como del código para el uso del NB-IoT.

GIE_SOLENOIDE_DRIVER.c:

```
#include "GIE_SOLENOIDE_DRIVER.h"
#include "GIE_UART_DRIVER.h"
#include "GIE_GPIO_DRIVER.h"

/*Private variables*/
uint8_t *valor; //para recibir del pin.
char recibido[30]="";

/* Variable para codigos de depuración*/
uint8_t status_SOL;
/**/



/**
 * @brief      Inicializa la depuración, seleccionando la UART y la
configuración de esta.
 *             Solo se ha usado para probar el funcionamiento.
 *
 * @param[in]  UART_PORT       Puerto de la UART
 *
 * @return      @ref SOL_INIT_CODES
 *
 */
/*INICIALIZACIÓN DE LA UART*/
uint8_t SOL_Init(uint8_t UART_PORT){

 /* DeclaraciOn estructura configuraciOn UART*/
 UART_Struct_Config configUART;

 /* Configuración de la uart */
 configUART.BaudRate = 115200;
 configUART.WordLength = UART_WORDLENGTH_8B;
 configUART.StopBits = UART_STOPBITS_1;
 configUART.Parity = UART_PARITY_NONE;

 /* Inicialización de la uart */
 status_SOL = UART_Init(UART_PORT, configUART);
 if (status_SOL != HAL_OK)
  {
    return SOL_INIT_UART_ERROR;
  }

 return SOL_INIT_OK;
}



/**
 * @brief      Inicializa configuración del GPIO
 *
 * @return     SOL_INIT_CODES
 */
/*INICIALIZACIÓN Y CONFIGURACIÓN DEL GPIO*/
uint8_t SOL_Confg_Init(){
```

```
/*Declaración estructura PIN*/
GPIO_PIN configPIN;

/*Configuración del PIN del GPIO*/
configPIN.number = 9;
configPIN.port = 'C';

/*Declaración estructura configuración GPIO*/
GPIO_Struct_Config configGPIO;

/*Configuración del GPIO*/
configGPIO.mode = GPIO_MODE_OUTPUT_PP;
configGPIO.pull = GPIO_PULLDOWN;

status_SOL = GPIO_Init(configPIN, configGPIO);
if(status_SOL == GPIO_OK){
    return SOL_INIT_OK;
}
return SOL_INIT_CONFIG_ERROR;
}



/**
 * @brief      Escritura en PIN
 *
 * @param[in]  configPIN       Estructura GPIO_PIN
 * @param[in]  val             Valor para escribir en el PIN
 *                             (0 desactivado o 1 activado)
 *
 * @return      @ref SOL_READ_UART_CODES y SOL_WRITE_CODES
 *
 */
uint8_t SOL_Write_PIN(GPIO_PIN configPIN, uint8_t val){
status_SOL = GPIO_Write(configPIN, val);
if(status_SOL!=GPIO_OK){
    return SOL_WRITE_ERROR;
}
return SOL_WRITE_OK;
}



/**
 * @brief      Leer del PIN
 *
 * @param[in]  configPIN        Estructura GPIO_PIN
 *
 * @return      @ref SOL_READ_CODES
 *
 */
uint8_t SOL_Read_PIN(GPIO_PIN configPIN){
  status_SOL = GPIO_Read(configPIN,valor);
  if(status_SOL!=GPIO_OK){
    return SOL_READ_ERROR;
  }
return SOL_READ_OK;
}



/**
```

```
 * @brief      Actuación del pin en función de lo recibido por UART
 *
 * @param[in]  UART_PORT        Puerto de la UART
 * @param[in]  configPIN        Estructura GPIO_PIN
 * @param[in]  esperado         Lo que esperamos recibir por la UART
 *
 * @return      @ref SOL_READ_CODES
 *
 */
uint8_t SOL_Actuacion(uint8_t UART_PORT, GPIO_PIN configPIN, char esperado){
  status_SOL = UART_ReadValue(UART_PORT, recibido, 100, 1000);
  HAL_Delay(500);

  int comp;
  comp==strncmp(recibido,esperado,100);
  if(comp==0){
    status_SOL = SOL_Write(configPIN,1);
    if(status_SOL==SOL_WRITE_OK){
      return SOL_ACT_OK;
      /*Descomentar esta parte si queremos desactivarlo*/
      /*HAL_Delay(500);
      status_SOL = SOL_Write(configPIN,0);
      if(status_SOL==SOL_WRITE_OK){
        return SOL_ACT_OK;
      }*/
    }
  return SOL_WRITE_ERROR;
  }
return SOL_ACT_ERROR;

}
```

NBIOT.c:

```
  /* ========================================================================== */
  /* INICIO NB-IoT - Introducir en la parte donde inicia perifericos */
  /* ========================================================================== */

  /*Configuramos el GPIO PIN*/
  GPIO_PIN PWRKEY_Pin1;
  PWRKEY_Pin1.number = 5;
  PWRKEY_Pin1.port   = 'B';

  /* Variable de reintentos de inicializacion */
  uint8_t SIM7080_retries = 0;

  /* Cadena de caracteres para respuesta NB-IoT */
  char respuesta[100];

  /* Vaariables necesarias para el driver del nbiot */
  char puerto[] = "7013";
  char ip[] = "86.109.110.101";
  char substring[10];

  while (status != SIM7080_INIT_OK){
    status = SIM7080_Init(UART_1, 9600, PWRKEY_Pin1, COMISSIONING_OFF, 0);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }

  while(status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED && SIM7080_retries <=
12){ //12 reintentos = 2 minutos
    SIM7080_retries++;
    status = SIM7080_CheckNetworkConnection();
    SIM7080_Debug(status,&NIVEL, CHAR);
```

```
    MandaColaDep(NIVEL, CHAR);
    vTaskDelay(10000);
  }
  SIM7080_retries = 0;

  /* Comprueba si hay conexion forzando operador Vodafone */
  if(status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED){
    status = SIM7080_Comissioning(OPERADOR);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
    vTaskDelay(3000/portTICK_PERIOD_MS);

    /* Si no ha cogido conexion forzando operador, establece busqueda automatica */
    if(status != SIM7080_COMISSIONING_OK && status !=
SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED){
      status = SIM7080_Comissioning(COMISSIONING_AUTO);
      SIM7080_Debug(status,&NIVEL, CHAR);
      MandaColaDep(NIVEL, CHAR);
      vTaskDelay(3000/portTICK_PERIOD_MS);

      /* Comprueba conexión */
      while(SIM7080_retries <= 30 && status !=
SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED){
        status = SIM7080_CheckNetworkConnection();
        SIM7080_Debug(status,&NIVEL, CHAR);
        MandaColaDep(NIVEL, CHAR);
        SIM7080_retries++;
        vTaskDelay(10000);
        if(SIM7080_retries >= 30)
        {
          MandaColaDep(VERBOSE_FATAL, "No se pudo conectar el modulo SIM7080 \n");
        }
      }
    }
  }

  /* Inicia red */
  SIM7080_retries = 0;
  while(status != SIM7080_INIT_NETWORK_CONNECTED_ROAMMING && status !=
SIM7080_INIT_NETWORK_CONNECTED_HOME_NETWORK ){
    status = SIM7080_InitNetwork();
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
    SIM7080_retries++;
    if (SIM7080_retries >= 9){
      SIM7080_Reboot();
      _SIM7080_Delay(2000);
      SIM7080_WakeUp();
    }
    vTaskDelay(3000);
  }

  /* Comprueba si esta conectado a red */
  SIM7080_retries = 0;
  status = SIM7080_CHECK_NETWORK_CONNECTION_NOT_CONNECTED;
  while (status != SIM7080_CHECK_NETWORK_CONNECTION_CONNECTED && SIM7080_retries <= 20
)
  {
    /* Damos 1 min de margen para que se conecte */
    if (SIM7080_retries >= 20){
      MandaColaDep(VERBOSE_ERROR, "vInicializacion: No se pudo conectar a la red.\n");
    }

    /* Comprobamos conexion cada 3 segundos */
    status = SIM7080_CheckNetworkConnection();
    SIM7080_retries++;
    vTaskDelay(3000);
  }
```

```
  MandaColaDep(VERBOSE_SUCCESS, "vInicializacion: Conectado a la red\n");

  /* Inicia conexion TCP */
  SIM7080_retries = 0;
  while ((status != SIM7080_OPEN_CONNECTION_OK) & SIM7080_retries <= 3){
    /* Damos 1 reintento mas para que se conecte al topic */
    if (SIM7080_retries >= 3)
    {
      MandaColaDep(VERBOSE_ERROR, "No se ha podido iniciar la transmision.\n");
    }
    /* Reintentamos cada 3 segundos */
    vTaskDelay(3000);
    SIM7080_retries++;
    status = SIM7080_InitConnection(SIM7080_UDP, puerto, ip, 0);
    CHAR[0]='\0';
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }
  /* ================================================================== */
  /* FIN INICIO NB-IoT */
  /* ================================================================== */


  /* ================================================================== */
  /* INICIO ENVIO NB-IoT . Introducir en la tarea donde se inicia el envio */
  /* ================================================================== */
  /* Envia trama en socket 0 */
  SIM7080_TransmitDatagram(0, "1");

  /* Recibe respuesta del servidor en socket 0 */
  _SIM7080_SendATCommand("AT+CARECV=0,1", "OK", respuesta, 100, 1000);

  /* Comprobacion de la substring, comprobando si el mensaje recibido es 2.*/
  _SIM7080_GetSubstring(respuesta, "+CARECV: 1,", "\r\n", substring);
  if (substring[0] == '2'){
    MandaColaDep(VERBOSE_SUCCESS, "Respuesta servidor correcta.\n");
  }else{
    MandaColaDep(VERBOSE_ERROR, "Respuesta servidor erronea.\n");
  }
  /* ================================================================== */
  /* FIN ENVIO NB-IoT */
  /* ================================================================== */


  /* ================================================================== */
  /* INICIO FINALIZACION NB-IoT . Introducir en la tarea despues del envio*/
  /* ================================================================== */
  /* Cierra conexion con el socket*/
  while (status != SIM7080_CLOSE_NETWORK_OK; ){
    status=SIM7080_CloseConnection(0);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }
  /* Manda al modulo a bajo consumo*/
 while (status != SIM7080_CLOSE_NETWORK_OK; ){
    status=SIM7080_SetPowerMode(SIM7080_PSM_MODE);
    SIM7080_Debug(status,&NIVEL, CHAR);
    MandaColaDep(NIVEL, CHAR);
  }
  /* ================================================================== */
  /* FIN FINALIZACIÓN NB-IoT */
  /* ================================================================== */
```

Comandos AT usados

[26]

AT: Solo devuleve un “OK” y no hace nada más. Se usa para chequear la comunicación.

AT+IPR: Establecer el local baud rate.

AT+IPR Establecer el Baud Rate local	
Comando de prueba ***AT+IPR=?***	**Respuesta:** **+IPR:** (lista de <**rate**>s detectables automáticamente admitidas), (lista de <**rate**>s solo fijas admitidas) **OK**
Comando de lectura ***AT+IPR?***	**Respuesta:** **+IPR: <rate>** **OK**
Comando de escritura ***AT+IC=<rate>***	**Respuesta:** Esta configuración de parámetros determina la velocidad de datos del TA en la interfaz serial. La tasa de Comando entra en vigor después de la emisión de cualquier código de resultado asociado con la línea de Comando actual. **OK**

Tabla 4. Comandos AT usados para establecer el Baud Rate del módulo NBIoT.

Valores definidos	
<rate>	Baud Rate por segundo 0 300 600 1200 ….

Tabla 5. Valores definidos del comando AT+IPR.

AT+CREBOOT: Resetea el módulo.

AT+CREBOOT Obtiene red APN en NB-IOT	
Comando de prueba ***AT+CREBOOT=?***	**Respuesta:** **OK**
Comando de ejecución ***AT+CREBOOT***	**Respuesta:** Ccid data **OK**

Tabla 6. Comandos AT usado para resetear el módulo NBIoT.

AT+CEDUMP: Establecer si el módulo se reinicia cuando se bloquea.

AT+CEDUMP Establecer si el módulo se reinicia cuando se bloquea	
Comando de lectura ***AT+CEDUMP=?***	**Respuesta:** **+CEDUMP: <mode>** **OK**
Comando de escritura ***AT+CEDUMP=<mode>***	**Respuesta:** **OK**

Tabla 7. Comandos AT usado para reiniciar el módulo si se bloquea.

Valores definidos	
<mode>	Modo de volcado 0 El módulo se restablecerá cuando el módulo se bloquee (predeterminado) 1 El módulo entrará en modo de descarga cuando el módulo se bloquee

Tabla 8. Valores definidos del comando AT+CEDUMP.

AT+CCID: Muestra el ICCID.

AT+CCID Obtiene red APN en NB-IOT	
Comando de prueba ***AT+CCID=?***	**Respuesta:** **OK**
Comando de ejecución ***AT+CCID***	**Respuesta:** Ccid data **OK**

Tabla 9. Comandos AT usado para muestrear el ICCID del módulo NBIoT.

AT+CMNB: Selección preferida entre CAT-M y NB-IoT.

AT+CMNB Selección preferida entre CAT-M y NB-IoT	
Comando de prueba ***AT+CMNB=?***	**Respuesta:** **+CMNB:** (lista de **<mode>** compatibles) **OK**
Comando de lectura ***AT+CMNB?***	**Respuesta:** **+CMNB: <mode>** **OK**
Comando de escritura **AT+CMNB=<mode>**	**Respuesta:** **OK**

Tabla 10. Comandos AT usado para seleccionar CAT-M o NB-IoT.

Valores definidos	
<mode>	1 CAT-M 2 NB-IoT 3 CAT-M and NB-IoT

Tabla 11. Valores definidos del comando AT+CMN

ATE: Comando que establece el modo Echo.

AT+IPR Establecer el Baud Rate local	
Comando de ejecución ***ATE<value>***	**Respuesta:** Esta configuración determina si el TA repite o no los caracteres recibidos del TE durante el estado de Comando. **OK**

Tabla 12. Comandos AT usados para establecer el modo Echo del módulo NBIoT.

Valores definidos	
<value>	0 Modo Echo OFF 1 Modo Echo On

Tabla 13. Valores definidos del comando AT+ATE.

AT+CACLOSE: Cierra una conexión TCP/UDP.

AT+CAACK Consulta para enviar información de datos	
Comando de prueba ***AT+CACLOSE=?***	**Respuesta:** **+CAACK:** (rango de **<cid>** admitidos) **OK**
Comando de escritura ***AT+CACLOSE=<cid>***	**Respuesta:** **OK**
Código de resultado no solicitado	Si <**autoClose_s**>se establece a 1, se informará cuando se desconecte la conexión remota. **+CACERRAR: <cid>**

Tabla 14. Comandos AT usado para cerrar una conexión TCP/UDP del módulo NBIoT.

<cid>: Identificador TCP/UDP.

AT+CARECV: Usado para recibir datos a través de la conexión que establecemos.

AT+CARECV Recibir datos a través de una conexión establecida	
Comando de prueba ***AT+CARECV=?***	**Respuesta:** **+CARECV:** (rango de **<cid>** admitidos), (**rango** de **<readlen>** compatibles) **OK**
Comando de escritura ***AT+CARECV=<cid>,<readlen>***	**Respuesta:** **+CARECV: [<remote IP>,<remote port>,]<recvlen>,**……// datos de salida **OK** (Nota: <IP remota> y <puerto remoto> se mostrarán si AT+CASRIP=1)

Tabla 15. Comandos AT usados para la recepción a través del módulo NBIoT.

Valores definidos	
<cid>	Identificador TCP/UDP.
<readlen>	Número solicitado de bytes de datos para ser leídos.
<recvlen>	Bytes de datos que se han recibido realmente.
<remote IP>	IP remota.
<remote port>	Puerto remoto.

Tabla 16. Valores definidos del comando AT+CARECV.

AT+CASEND: Usado para enviar datos a través de la conexión que establecemos.

AT+CASEND Enviar datos a través de una conexión establecida	
Comando de prueba ***AT+CASEND=?***	**Respuesta:** **+CASEND:** (rango de <cid> admitidos),(rango de <datalen> admitidos),(rango de <inputtime> soportados) **OK**
Comando de escritura ***AT+CASEND=<cid>***	**Respuesta:** **+CASEND: <leftsize>** **OK**
Comando de escritura ***AT+CASEND=<cid>,<datalen>[,<inputtime>]***	**Respuesta:** **+CASEND: <cid>,<datalen>**….// datos de entrada **OK**

Tabla 17. Comandos AT usados para el envío de datos a través del módulo NBIoT.

Valores definidos	
<cid>	Identificador TCP/UDP.
<leftsize>	Tamaño libre de consultas para el búfer de envío.
<datalen>	Número solicitado de bytes de datos a transmitir.
<inputtime>	Milisegundos, debe ingresar datos durante este período o no podrá ingresar datos cuando se agote el tiempo.

Tabla 18. Valores definidos del comando AT+CASEND.

AT+COPS: Selección del operador.

AT+COPS Selección del operador	
Comando de prueba ***AT+COPS=?***	**Respuesta:** TA devuelve una lista de cuatrillizos, cada uno de los cuales representa a un operador presente en la red. Cualquiera de los formatos puede no estar disponible y debería ser un campo vacío. La lista de operadores estará en orden: red de origen, redes referenciadas en SIM y otras redes. **+COPS**: (lista de **<stat>** admitidos, <**oper**> alfanumérico largo, <**oper**> alfanumérico corto, <**oper**> numérico, <**netact**>)s[,,(lista de <**mode**> admitidos),(lista de <**format**> admitidos] **OK**
Comando de lectura ***AT+COPS?***	**Respuesta:** TA devuelve el modo actual y el operador actualmente seleccionado. Si no se selecciona ningún operador, se omiten **<formato>** y **<oper>.** **+COPS: <modo>[,<formato>,<oper>,<netact>]** **OK**
Comando de escritura ***AT+COPS=<mode>,[<format>[,<oper>]]***	**Respuesta:** TA fuerza un intento de seleccionar y registrar el operador de red GSM. Si el operador seleccionado no está disponible, no se seleccionará ningún otro operador (excepto <**mode**>=4). El formato de nombre del operador seleccionado se aplicará a los comandos de lectura adicionales (**AT+COPS?**). **OK**

Tabla 19. Comandos AT usados para seleccionar el operador del módulo NBIoT.

Valores definidos	
<stat>	0 Desconocido 1 Operador disponible 2 Operador actual 3 Operador prohibido
<oper>	Referirse [27.007] al operador en formato según <**format**> .
<mode>	0 Modo automático; El campo <**oper**> se ignora 1 Manual (el campo <**oper**> debe estar presente y <AcT> opcionalmente) 2 Cancelación manual de la red

	3 Establezca solo <**format**> (¿para comando de lectura +COPS?) - no se muestra en la respuesta del comando de lectura 4 Manual/automático (el campo <**oper**> debe estar presente); si falla la selección manual, se ingresa al modo automático (<modo>=0)
<format>	0 Alfanumérico de formato largo <**oper**> 1 formato corto alfanumérico <**oper**> 2 Numérico <**oper**>; Número de identificación del área de ubicación GSM
<netact>	0 Tecnología de acceso GSM especificada por el usuario 1 GSM compacto 3 GSM EGPRS 7 Tecnología de acceso LTE M1 A GB especificada por el usuario 9 Tecnología de acceso LTE NB S1 especificada por el usuario

Tabla 20. Valores definidos del comando AT+COPS.

AT+CGNAPN: Obtiene red APN en NB-IOT.

AT+CGNAPN Obtiene red APN en NB-IOT	
Comando de prueba ***AT+CGNAPN=?***	**Respuesta:** **+CGNAPN:** (lista de **<valid>** admitidos), **<length>** **OK**
Comando de ejecución ***AT+CGNAPN***	**Respuesta:** **+CGNAPN**: <**valid**>,<**Network_APN**> **OK**

Tabla 21. Comandos AT usados para obtener red APN con el módulo NBIoT.

Valores definidos	
<valid>	0 La red no envió el parámetro APN al UE. En el caso, <**Network_APN**> es NULL. 1 La red envió el parámetro APN al UE.
<length>	Max longitud de <**network_APN**>.
<Network_APN>	Tipo de cadena. La red envía el parámetro APN al UE cuando el UE registra la red CAT-M o NB-IOT con éxito. En GSM, <**Network_APN**> siempre es NULL.

Tabla 22. Valores definidos del comando AT+CGNAPN

AT+CGREG: Estado de registro de red.

AT+CGREG Estado de registro de red	
Comando de prueba ***AT+CGREG=?***	**Respuesta:** **+CGREG**: (lista de **<n>** admitidos) **OK**
Comando de lectura ***AT+CGREG?***	**Respuesta:** **+CGREG**: **<n>,<stat>[,<lac>,<ci>,<netact>,<rac>[,[<Active-Time>],[<Periodic-RAU>],[<GPRS-READY -temporizador>]]]** **OK**
Comando de escritura ***AT+CGREG[=<n>]***	**Respuesta:** **OK**

Tabla 23. Comandos AT usado para establecer el estado de registro de red del módulo NBIoT.

Valores definidos	
<n>	0 Deshabilitar registro de red código de resultado no solicitado 1 Habilitar registro de red código de resultado no solicitado **+CGREG: <stat>** 2 Habilitar registro de red e información de ubicación código de resultado no solicitado **+CGREG: <stat>[,<lac>,<ci>,<netact>,<rac**>] 4 Habilitar visualización de tiempo GPRS y RAU periódico
<stat>	0 No registrado, MT no está buscando actualmente un operador para registrarse. El servicio GPRS está deshabilitado, el UE puede conectarse para GPRS si lo solicita el usuario. 1 Red doméstica registrada. 2 No registrado, pero MT está tratando de adjuntar o buscando un operador para registrarse. El servicio GPRS está habilitado, pero actualmente no hay disponible una PLMN permitida. El UE iniciará una conexión GPRS tan pronto como esté disponible una PLMN permitida. 3 Registro denegado, el servicio GPRS está deshabilitado, el UE no puede conectarse para GPRS si el usuario lo solicita. 4 Desconocido 5 Registrado, itinerancia 6 DSAT_REG_REGISTERED_MAX /* ¡Solo para uso interno! */
<lac>	Tipo de cadena (la cadena debe incluirse entre comillas); código de área de ubicación de dos bytes en formato hexadecimal (por ejemplo, "00C3" equivale a 195 en decimal)

<ci>	Tipo de cadena (la cadena debe incluirse entre comillas); ID de celda de dos bytes en formato hexadecimal
<netact>	0 Tecnología de acceso GSM especificada por el usuario 1 GSM compacto 3 GSM EGPRS 7 Tecnología de acceso LTE M1 A GB especificada por el usuario 9 Tecnología de acceso LTE NB S1 especificada por el usuario
<rac>	Tipo de cadena; código de área de enrutamiento de un byte en formato hexadecimal
<Active-Time>	tipo de cadena; un byte en un formato de 8 bits. Valor de tiempo activo solicitado (T3324) que se asignará al UE. El valor de tiempo activo solicitado se codifica como un byte (octeto 3) del elemento de información del temporizador GPRS 2 codificado como formato de bits (por ejemplo, "00100100" equivale a 4 minutos).
<Pariodic-RAU>	tipo de cadena; un byte en un formato de 8 bits. Valor TAU periódico extendido solicitado (T3412) para ser asignado al UE en E-UTRAN. El valor TAU periódico ampliado solicitado se codifica como un byte (octeto 3) del elemento de información del temporizador GPRS 3 codificado como formato de bits (por ejemplo, "01000111" es igual a 70 horas).
<GPRS-READY-timer>	tipo de cadena; un byte en un formato de 8 bits. Se solicitó el valor del temporizador GPRS READY (T3314) para ser asignado al UE en GERAN/UTRAN. El valor del temporizador GPRS READY solicitado se codifica como un byte (octeto 2) del elemento de información del temporizador GPRS codificado como formato de bit (por ejemplo, "01000011" equivale a 3 decihoras o 18 minutos).

Tabla 24. Valores definidos del comando AT+CGREG.

AT+CPSMS: Configuración del modo de ahorro de energía.

AT+CPSMS Configuración del modo de ahorro de energía	
Comando de prueba ***AT+CPSMS=?***	**Respuesta:** **+CPSMS:** (lista de **<mode>** compatibles), (lista de **<Requested_Periodic-RAU>** compatibles),(lista de **<Requested_GPRS-READY-timer >** compatibles),(lista de **<Requested_Periodic-TAU>** compatibles) ,(lista de **<Requested_Active-Time>** admitidos) **OK**
Comando de lectura ***AT+CPSMS?***	**Respuesta:** **+CPSMS: <mode>,[<Requested_Periodic-RAU>],[<Requested_GPRS-READY-timer>],[<Requested_Periodic-TAU>],[<Requested_Active-Time>]** **OK**
Comando de escritura **AT+CPSMS=[<mode>[,<Requested_Periodic-RAU>[,<Requested_GPRS-READY-timer>[,<Requested_Periodic-TAU>[,<Requested_Active-Time>]]]]]**	**Respuesta:** **OK**

Tabla 25. Comandos AT usado para la configuración del modo de ahorro de energía del módulo NBIoT.

Valores definidos	
<mode>	0 Deshabilitar el uso de PSM 1 Habilitar el uso de PSM
<Requested_Periodic-RAU>	No soportado
<Requested_GPRS-READY-timer>	No soportado
<Requested_Periodic-TAU>	Tipo de cadena; un byte en un formato de 8 bits. Valor TAU periódico extendido solicitado (T3412) para ser asignado al UE en E-UTRAN. El valor TAU periódico ampliado solicitado se codifica como un byte (octeto 3) del elemento de información del temporizador GPRS 3 codificado como formato de bits (por ejemplo, "01000111" es igual a 70 horas). Para la codificación y el rango de valores, consulte GPRS Timer3 IE en 3GPP TS 24.008 [8] Tabla 10.5.163a/3GPP TS 24.008. Ver también 3GPP TS 23.682 [149] y 3GPP TS 23.401 [82]. El valor predeterminado, si está disponible, es específico del fabricante.

<Requested_Active-Time>	Tipo de cadena; un byte en un formato de 8 bits. Valor de tiempo activo solicitado (T3324) que se asignará al UE. El valor de tiempo activo solicitado se codifica como un byte (octeto 3) del elemento de información del temporizador GPRS 2 codificado como formato de bits (por ejemplo, "00100100" equivale a 4 minutos). Para la codificación y el rango de valores, consulte GPRS Timer 2 IE en 3GPP TS 24.008 [8] Tabla 10.5.163/3GPP TS 24.008. Ver también 3GPP TS 23.682 [149], 3GPP TS 23.060 [47] y 3GPP TS 23.401 [82]. El valor predeterminado, si está disponible, es específico del fabricante.

Tabla 26. Valores definidos del comando AT+CCPSMS.

AT+CAACK: Consulta para enviar información de datos.

AT+CAACK Consulta para enviar información de datos	
Comando de prueba ***AT+CAACK=?***	**Respuesta:** **+CAACK:** (rango de **<cid>** admitidos) **OK**
Comando de escritura ***AT+CAACK=<cid>***	**Respuesta:** **+CAACK: <totalsize>,<unacksize>** **OK**

Tabla 27. Comandos AT usado para consultar el envio de info de datos del módulo NBIoT.

Valores definidos	
<cid>	Identificador TCP/UDP
<totalsize>	Tamaño total de los datos enviados
<unacksize>	El tamaño de los datos no confirmados

Tabla 28. Valores definidos del comando AT+CAACK.

AT+CBANDCFG: Configurar la banda CAT-M o NB-IOT.

AT+CBANDCFG	**Configurar la banda CAT-M o NB-IOT**
Comando de prueba ***AT+CBANDCFG=?***	**Respuesta:** **+CBANDCFG**: (lista de <**mode**> compatibles), (lista de <**band**> compatibles) **OK**
Comando de lectura ***AT+CBANDCFG?***	**Respuesta:** **+CBANDCFG: "CAT-M",<band>[,<band>…]<CR><LF>+CBANDCFG: "NB-IOT",<band>[,<band>…]** **OK**
Comando de escritura **AT+CBANDCFG=<mode>,<band>[,<band>…]**	**Respuesta:** **OK**

Tabla 29. Comandos AT usado para configurar la banda del módulo NBIoT.

Valores definidos	
<mode>	Tipo de cadena; modo de sistema de red. "CAT-M" → LTE Cat. M1 (eMTC) "NB-IOT" → Internet de las cosas de banda estrecha
<band>	Tipo entero; El valor de <**band**> debe estar en la lista de bandas de obtención de **AT+CBANDCFG=?**

Tabla 30. Valores definidos del comando AT+CBANDCFG.

REFERENCIAS

[1] LA VANGUARDIA, "Internet de las cosas: cuando todo está conectado.", [En línea]. Disponible: https://www.lavanguardia.com/vida/junior-report/20190301/46752655177/internet-cosas-dispositivos-conectados-iot.html.

[2] ADSLZone, "Todo lo que debes saber sobre las cerraduras inteligentes", [En línea]. Disponible: https://www.adslzone.net/reportajes/domotica/guia-compra-cerraduras-inteligentes/.

[3] "Wikipedia", [En línea]. Disponible en: https://es.wikipedia.org/wiki/Red_inal%C3%A1mbrica.

[4] Rob Faludi., «¿Cómo se comunican los dispositivos IoT? », 26 marzo 2021. [En línea]. Disponible: https://es.digi.com/blog/post/how-do-iot-devices-communicate.

[5] Universidad Abierta de Cataluña., «¿Qué es NB-IoT? », 22 noviembre 2018. [En línea]. Disponible: https://blogs.uoc.edu/informatica/que-es-nb-iot/.

[6] Fernando C, "LPWAN: qué son y para qué se utilizan", M2M - Logitek. 21 julio 2020. [En línea]. Disponible: https://www.m2mlogitek.com/lpwan-que-son-y-para-que-se-utilizan/.

[7] Semak., "LoRaWAN - ¿Qué es?". [En línea]. Disponible: https://www.semak.com.ar/tienda/lorawan-que-es.

[8] Programador clic, "LoRaWAN". [En línea]. Disponible: https://programmerclick.com/article/1617299759/.

[9] J. E. Crespo, «Aprendiendo Arduino,» "Demo MKRFOX1200", 7 marzo 2018. [En línea]. Disponible: https://www.aprendiendoarduino.com/tag/mensajes-sigfox/.

[10] Mouser Electronics España, "DSML-0224-12", [En línea]. Disponible: https://www.mouser.es/datasheet/2/632/delta_electronics_06152020_DSML-0224-1860122.pdf.

[11] armMBED, "NUCLEO-L152RE", [En línea]. Disponible: https://os.mbed.com/platforms/ST-Nucleo-L152RE/.

[12] STMicroelectronics, " STM32 Nucleo-64 boards (MB1136)" UM1724 User manual, 2020.

[13] "FT4232 Mini Module Datasheet," [En línea]. Disponible: https://www.farnell.com/datasheets/1915281.pdf.

[14] «AXSF10 ANT21868 Datasheet,» [En línea]. Disponible: https://www.alldatasheet.com/datasheet-pdf/pdf/763729/ONSEMI/AX-SF10-ANT21- 868.html.

[15] Melopero ®, " NB-IoT / Cat-M (eMTC) / GNSS HAT para Raspberry Pi, Aplicable a nivel mundial.", [En línea]. Disponible: https://www.melopero.com/es/tienda/raspberry-pi/hats/nb-iot-cat-memtc-gnss-hat-para-raspberry-pi-globally-applicable/.

[16] SIMCom. "SIM7070_SIM7080_SIM7090 Series_ThreadX DAM_User Guide_V1.00", [En línea].

[17] Kernighan, B. y Ritchie, D., 2011. *The C programming language*. Englewood Cliffs, N.J.: Prentice-Hall.

[18] Guillermo. H. G, «Página Web Personal,» "RTOS". [En línea]. Disponible: http://www.guillehg.com/index.php?option=com_content&view=category&id=39&Itemid=689.

[19] Amazon Web Services, «FreeRTOS,». [En línea]. Disponible: https://aws.amazon.com/es/freertos/.

[20] FreeRTOS™, "Kernel,". [En línea]. Disponible: https://www.freertos.org/RTOS-task-states.html.

[21] DSI, Departamento de Sistemas e Informática. [En línea]. Disponible: "https://www.dsi.fceia.unr.edu.ar/images/Sistemas_Embebidos/Presentacion_FreeRTOS.pdf

[22] FreeRTOS™, «The FreeRTOS Reference Manual,». [En línea]. Disponible: https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf.

[23] 330ohms, "Solenoies-Actuadores mecánicos", [En línea]. Disponible: https://blog.330ohms.com/2016/03/14/solenoides-actuadores-mecanicos/.

[24] Murky Robot, "Solenoides: Actuador Electromagnético", [En línea]. Disponible: https://www.murkyrobot.com/guias/actuadores/solenoides.

[25] Wikimedia Commons. "File: Duty Cycle Examples.". [En línea]. Disponible: https://commons.wikimedia.org/wiki/File:Duty_Cycle_Examples.png.

[26] «SIM7070_SIM7080_SIM7090 Series_AT Command Manual,» [En línea]. Disponible: https://www.waveshare.com/w/upload/0/02/SIM7070_SIM7080_SIM7090_Series_AT_Command_Manual_V1.03.pdf.

[27] David Sánchez Rosado, NB-IoT. Tecnologías celulares *narrow-band.* Análisis práctico de las soluciones de Telefónica y Vodafone. *Trabajo fin de máster, Universidad Complutense de Madrid*, 2019.

[28] Futurizable. «Tres letras que están cambiando el mundo,», 4 noviembre 2016. [En línea]. Disponible: https://futurizable.com/iot/.