Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Desarrollo de un sistema domótico basado en OpenHAB empleando dispositivos Arduino

Autor: José Enrique Romero Olías
Tutor: Jesús Iván Maza Alcañiz

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Desarrollo de un sistema domótico basado en OpenHAB empleando dispositivos Arduino

Autor:
José Enrique Romero Olías

Tutor:
Jesús Iván Maza Alcañiz
Profesor titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Desarrollo de un sistema domótico basado en OpenHAB empleando dispositivos Arduino

Autor: José Enrique Romero Olías

Tutor: Jesús Iván Maza Alcañiz

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia y amigos

A mi abuela

Agradecimientos

El final de este trayecto está llegando a su fin y me gustaría agradecer a mi familia el apoyo que me ha dado, no sólo en estos años de Universidad, sino durante toda mi vida. Sin ellos no sería la persona que soy ahora.

Papá, gracias por tu constancia y cariño; gracias por todas esas noches en velas que pasaste conmigo de chico hasta que aprendiera el temario; gracias por las lecciones de lectura en las escaleras de Marchena. Papá, sin ti nunca habría llegado hasta aquí. Estaré eternamente agredido por el esfuerzo que hiciste y sigues haciendo para que cada día sea mejor. Te quiero mucho, papá.

Mamá, no encuentro palabra que pueda usar para agradecerte como debería todo lo que haces por mí. Seguiré intentando diariamente demostrarte lo mucho que te quiero.

Myriam y Celia, agradeceros el soportarme diariamente. En especial a ti, Celia, por ser un ejemplo de constancia y esfuerzo. En este caso es el hermano mayor el que tiene que aprender, y mucho, de la más chica de la familia. Resulta que a veces sí es cierto que lo mejor viene en pequeñas dosis.

A los que son mis amigos desde no recuerdo ya cuándo, gracias. Álvaro, Rafa, Juan, Loren y Pablo, gracias por todos estos años; por compartir mis mismas peculiaridades. No importa el tiempo que pase sin vernos, cuando nos juntamos siempre volvemos a ser esos chavales de secundaria. En particular a ti, Álvaro, por estar siempre ahí (aunque siendo vecinos tampoco es que tengas demasiadas escapatorias).

Alfonso, Dani, Juanma, Valle, María, Laura y María, mis compañeros durante estos años de Universidad. Gracias por hacer que me supere diariamente y por aportarme tantos buenos recuerdos durante estos años.

A todas las personas que han formado parte de mi vida y que, en mayor o menor medida, han contribuido a hacer de mí la persona que soy hoy en día.

Por último, agradecer a mi profesor Iván Maza su ayuda y consejos para acometer este último proyecto de mi etapa universitaria.

José Enrique Romero Olías

Sevilla, 2019

Resumen

La domótica es el conjunto de tecnologías aplicadas al control y la automatización de ciertas acciones de los objetos de la vivienda, dando lugar a la llamada casa inteligente. La interacción entre el usuario y el sistema domótico proporciona seguridad, confort y gestión eficiente de la energía.

Hoy en día, y cada vez más, existen numerosos dispositivos de internet de las cosas (IoT), sensores y actuadores, así como soluciones de automatización de la vivienda. El problema habitual es triple. En primer lugar, las diferentes soluciones de automatización son incapaces de interaccionar con otras soluciones. En segundo lugar, sólo permiten la integración de aquellos dispositivos distribuidos por el mismo fabricante. Finalmente, tenemos el costo elevado de dichos dispositivos.

El objetivo final de este proyecto es realizar una implementación de un sistema domótico, usando dispositivos de diferentes fabricantes y bajo costo. Integrando estos dispositivos en un único sistema de automatización administrado y configurado por el usuario final.

Abstract

Domotic is the conjunct of technologies applied to the control and automation of certain actions of the objects in the house, which results in the so call intelligent home. The interaction between the user and the domotic system provides security, comfort and efficient management of the energy.

Nowadays, and every time more, there are numerous Internet of Things devices (IoT), sensors and actuators, as well as solutions for the automation of the house. The common problem is a triple one. First of all, the different automation solutions are not capable of interacting among them. In second place, they only allow the integration of those devices distributed by the same producer. Finally, there is the high cost of such devices.

The ultimate goal of this project is to carry out an implementation of a domotic system making use of low cost devices from different suppliers and integrating them in a unique automation system which can be configured and administrated by the final user.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

E/S	Entrada / Salida
DC	Corriente Continua
AC	Corriente Alterna
V	Voltios
mA	Miliamperios
Kohm	Kilo Ohmio
MHz	Megahercios
KHz	Kilohercios
KB	Kilobytes
g	Gramos
mm	Milímetros
D0	Pin digital cero
A0	Pin analógico cero
TX	Transmisión
RX	Recepción
<	Menor
>	Mayor

1 INTRODUCCIÓN

There´s nothing I believe in more strongly than getting young people interested in science and engineering, for a better tomorrow, for all humankind.

- Bill Nye -

En este capítulo se va a exponer el panorama actual de la domótica o automatización de la vivienda, y el por qué resulta de interés la realización de un sistema domótico unificador de soluciones y tecnologías. Se describe también la organización del documento y se realiza una pequeña introducción de las partes principales de este proyecto

1.1 Motivación

Debido al gran avance en la tecnología de la automatización de la vivienda que ha tenido lugar en la última década, son numerosas las compañías que intentan colocarse a la cabeza de esta tecnología, fabricando sus propios dispositivos y software para la integración de estos, sin que haya aún una marca dominadora clara.

La gran popularidad actual de los dispositivos IoT, unido al continuo interés del control de nuestro entorno, ha provocado que cada vez más sean las personas que optan por esta tecnología para facilitar su día a día. Esta situación ha traído consigo el interés de las empresas tecnológicas, las cuales fabrican productos propios con un precio excesivamente elevado en comparación al costo de producción. Este sobrecoste se debe, en gran parte, al deseo de las personas de manipular lo menos posible la configuración del dispositivo.

1.2 Objetivos

Este proyecto tiene como objetivo principal realizar el diseño e implementación de un sistema domótico de bajo coste, dónde la administración de los datos involucrados no recaiga en empresas externas, sino que sea el usuario final el que administre dichos datos.

Con este proyecto se pretende poder unificar lo mejor de las diferentes soluciones de automatización existentes en el mercado con las necesidades individuales que cada usuario pueda tener, sirviendo este proyecto como una base común.

En todo momento, el sistema domótico gira en torno a las necesidades individuales del usuario final, dejando en su mano la administración de los sensores, dispositivos, … así como de la tecnología deseada para la comunicación de los mismos y el control de los datos aportados por los sensores.

1.3 Organización del documento

El presente documento se ha estructurado en cinco grandes bloques para facilitar la comprensión del proyecto.

En primer lugar, se realiza una explicación teórica del eje central del sistema de automatización: OpenHAB. En este primer bloque, se explicará en detalle los principales conceptos usados en el software, y que son necesarios para la interpretación del documento.

El segundo bloque consistirá en una exposición de las características del hardware utilizado en el sistema domótico. El hardware estará dividido a su vez en tres secciones: Arduino, NodeMCU y Raspberry Pi.

Con estos dos primeros bloques ya están expuestas las diferentes piezas que componen el sistema domótico a desarrollar, con lo que pasamos a un tercer bloque en el que se detalla cómo se ha decidido unir dichas piezas. Este tercer bloque estará dividido en dos secciones: Comunicación entre la placa nodeMCU y Arduino, y la comunicación entre Raspberry Pi (openHAB) y nodeMCU.

Una vez detalladas las piezas y la comunicación entre ellas, los dos bloques restantes estarán destinados al desarrollo práctico del sistema y a las conclusiones y líneas de futuro.

2 ENTORNO OPENHAB

It's not that we use technology, we live technology.

- Godfrey Reggio-

En este capítulo, se expondrán los conceptos en los que se fundamenta openHAB, así como una descripción del mismo y los protocolos en los que se basa.

2.1 Introducción

openHAB es una plataforma de automatización doméstica, de código abierto, que integra diferentes sistemas de automatización del hogar, dispositivos y tecnologías en una única solución, y funciona como centro del sistema de automatización (openHAB. Open Home Automation Bus. Documentation).

openHAB está desarrollado en Java, y se basa principalmente en Eclipse SmartHome. Utiliza Apache Karaf junto con Eclipse Equinox para crear un entorno en tiempo de ejecución OSGi y usa Jetty como servidor HTTP. openHAB es un software altamente modular, que puede extenderse a través de complementos o extensiones según las necesidades del usuario (Figura 2.1)

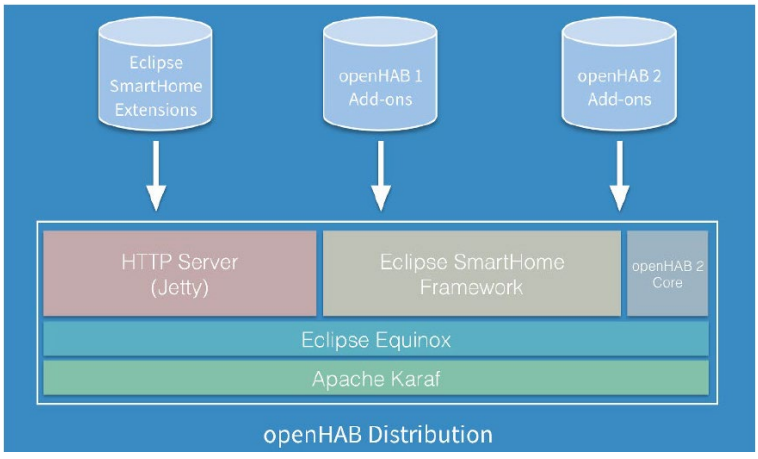


Figura 2.1 - Distribución openHAB

En este proyecto, openHAB será el elemento principal de la arquitectura, recogiendo y mostrando, mediante una interfaz personalizada, los datos obtenidos por los diferentes sensores e interaccionando con los actuadores.

Se ha elegido este software como solución para la integración de dispositivos por ser uno de los más completos en su sector gracias a la extensa comunidad que lo soporta. Actualmente, openHAB integra más de 2000 tecnologías y sistemas diferentes, miles de dispositivos, permite usar comandos de voz, mostrar notificaciones y puede ser instalado sobre diferentes S.O tales como Linux, macOS, Windows y en dispositivos variados, como Raspberry Pi, Docker, … (openHAB. open Home Automation Bus)

2.2 Conceptos

openHAB emplea una terminología que es necesario entender para la correcta administración del software y la cual se explica a continuación:

THING: Una *thing* es cualquier entidad que puede ser añadida al sistema. Las *things,* por lo tanto, pueden ser sensores, dispositivos electrónicos, ... aunque no necesariamente deben ser algo físico, también podría ser, por ejemplo, un servicio web.

CHANNEL: Las *things* exponen sus capacidades a través de los *channels*, en adelante "canales". Se entiende por capacidad, cualquier propiedad configurable. Como ejemplo de esto puede ponerse una bombilla que sea capaz de cambiar de color: tendrá un canal de apagado/encendido y otro canal de color.

BINDING: Los *binding,* o "asociación dedicada", pueden verse como adaptadores software, haciendo que las things estén disponibles en el sistema de automatización. Los *bindings* forman parte de lo que se conoce como *add-ons*, en adelante extensiones, y proveen una forma de linkar las things con los *items*,

ITEM: Los *items*, en adelante ítems, representan las capacidades que pueden ser usadas en el ámbito software de openHAB. Los ítems tienen estados y reciben comandos para cambiar de un estado a otro.

LINK: Un link es un enlace entre un canal y un ítem.

SITEMAP: Se trata de un menú visual de los ítems y sus valores.

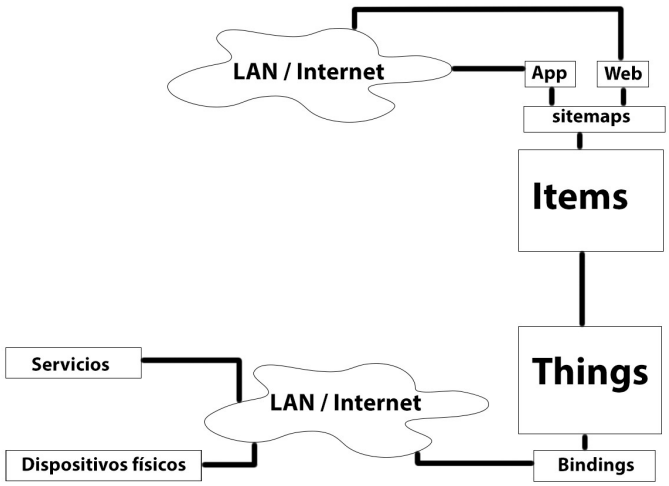


Figura 2.2 - Conceptos openHAB

2.2.1 Things

Como se ha comentado, las *things* son cualquier entidad que pueda ser añadida al sistema. Pueden tener propiedades de configuración opcionales u obligatorias, como puede ser una dirección IP, un token, …

Cada *thing* tiene asociado un estado que ayudaría a identificar cualquier tipo de problema con el dispositivo o servicio. Los posibles estados de las *things* son:

Tabla 2.1 - Propiedades de los things

Estado	Descripción
UNINITIALIZED	Estado inicial de una *thing* cuando es añadida a openHAB o se inicia el framework. Este estado es asignado también a una *thing* si el proceso de INITIALIZING falla o si el binding no está disponible. Cuando una *thing* tiene este estado, los comandos que son enviados por los *bindings* no son procesados.
INITIALIZING	Este estado es asignado mientras el *binding* inicializa la *thing*, con lo que el tiempo que éste estado es asignado depende del *binding*. Durante este estado los comandos no son procesados.
UNKNOWN	El *handler* o manejador está totalmente iniciado, pero aún no es capaz de representar si la *thing* está ONLINE u OFFLINE. En este estado, la *thing* ya funcionaría correctamente, el *framework* ya podría mandar comandos y el procesado de comandos no depende del *handler*.
ONLINE	El dispositivo o servicio representado por una *thing* está encendido y puede procesar los comandos.
OFFLINE	La *thing* está en modo apagado o desconectado y no procesa los comandos, aunque el framework puede enviar comandos. De esta forma, puede enviarse el comando para que vuelva al estado ONLINE y la *thing* se encienda.
REMOVING	El dispositivo o servicio representado por una *thing* está siendo retirado, pero el *binding* aún no ha confirmado que haya sido retirado. Este estado se da cuando un *binding* necesita comunicarse con el dispositivo/servicio para desparejarse. La *thing* probablemente no funcionará durante este estado y los comandos no serán procesados.
REMOVED	El dispositivo o servicio ha sido eliminado del sistema externo a openHAB después de que se haya pasado por el estado REMOVING por el *framework*.

Los estados UNINTIALIZED, INITIALIZING y REMOVING son establecidos por el *framework,* mientras que los estados UNKONWN, ONLINE y OFFLINE son asignados por el *binding*.

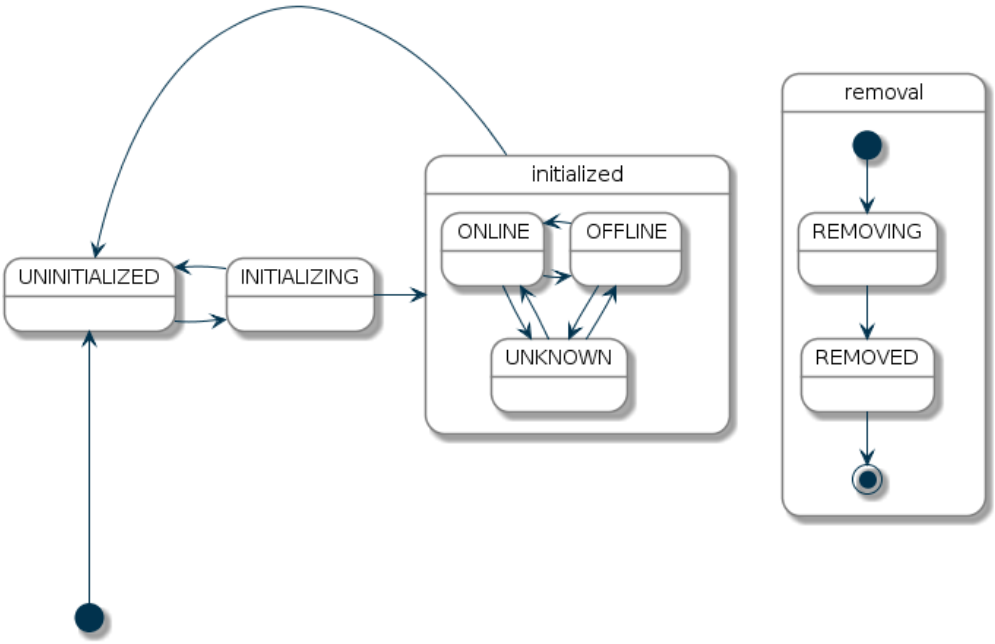


Figura 2.3 - Diagrama de estados de things

2.2.2 Ítems

La separación entre las *things* y los ítems es debido a la separación que hace Eclipse SmartHome entre el mundo físico (*things*) y la aplicación (construida en torno al concepto de ítems).

Actualmente openHAB soporta los ítems que aparecen en Tabla 2.2

Tabla 2.2 - Tipos de ítems

Ítem	Descripción	Comandos soportados
Color	Información del color RGB	On/Off, Increase/Decrease, Percent, HSB
Contact	Informa sobre el estado de una *thing* que puede estar abierto o cerrado (Ventana, puerta, …)	Open/Closed
DateTime	Informa sobre la hora y fecha	-
Dimmer	Almacena el porcentaje de un potenciómetro	On/Off, Increase/Decrease, Percent
Group	Sirve para almacenar varios ítems	-
Image	Almacena una imagen (en formato binario)	-
Location	Guarda las coordenadas GPS	Point
Number	Acopia valores en formato numérico.	Decimal
Number:<dimensión>	Especifica una unidad de medida para el valor numérico	Quantity
Player	Permite controlar reproductores	Play/Pause, Next/Previous, Rewind, Fast, Forward
Rollershutter	Usado para gestionar una persiana a motor	Up/Down, Stop/Move, Percent.
String	Almacena texto	String
Switch	Conmutador que informa del estado del ítem	On/Off

2.2.3 Sitemap

Los *sitemaps* preparan los ítems para que puedan ser visualizados en la interfaz de usuario. Durante la codificación de un *sitemap* podemos hacer uso de los elementos recogidos en Tabla 2.3

Tabla 2.3 - Elementos de un sitemap

Elemento	Descripción
Chart	Gráfico de datos aportados por una base de datos
Colorpicker	Permite elegir al usuario un color proporcionado por una paleta de colores
Default	Representa el ítem según su tipo
Frame	Elemento flotante en el que se pueden colocar otros elementos
Group	Agrupa los elementos de un grupo en un bloque fijo
Image	Muestra una imagen mediante su URL
Mapview	Muestra la localización de un *ítem* en un OpenStreetMap (OSM)
Selection	Muestra una barra con la representación de valores para asignar a un *ítem*
Setpoint	Muestra botones de incremento y decremento junto con el valor actual
Slider	Representa un valor en una barra deslizante
Switch	Muestra un interruptor con las posiciones ON y OFF
Text	Representa un texto
Video	Reproduce un vídeo en directo mediante una URL
Webview	Muestra el contenido de una página web.

3 DESCRIPCIÓN DEL HARDWARE EMPLEADO

One machine can do the work of tifty ordinary men. No machine can do the work of one extraordinary man.

- Elbert Hubbard -

En este capítulo se detallarán las especificaciones de interés de los dispositivos hardware empleados para el desarrollo del sistema domótico.

3.1 Arduino

Arduino es una plataforma electrónica de código abierto basada en hardware y software fácil de usar, que nació en el Instituto de Diseño de Interacción Ivrea como una herramienta fácil de usar para la creación rápida de prototipos y que, con el tiempo, fabricó placas de desarrollo para aplicaciones IoT, impresión 3D, …

Las placas de Arduino son las que se han decido emplear para el sistema domótico por su software fácil de usar, interfaz limpia y por el gran abanico de sensores y actuadores de bajo coste existentes en el mercado.

3.1.1 Arduino UNO

En este proyecto se ha utilizado la placa de desarrollo Arduino UNO R3 en su versión DIP, la cual está basada en el microcontrolador ATmega328P y que, al ser versión DIP, permite desmontar el microcontrolador de la placa.

3.1.1.1 Especificaciones técnicas

Los datos técnicos provistos por el fabricante aparecen en la Tabla 3.1:

Tabla 3.1 - Especificaciones técnicas Arduino UNO

Microcontrolador	ATmega328P
Tensión de funcionamiento	5V
Tensión de alimentación recomendada	7-12V
Tensión de alimentación límite	6-20V
Pines Digitales de E/S	14 (de los cuales 6 permiten salida PWM)
Pines Digitales PWM	6
Pines de entrada analógicos	6

Corriente DC por pin E/S	20mA
Corriente DC por pin a 3.3V	50mA
Memoria Flash	32KB (ATmega328P) de los cuales 0.5KB son usados por el bootloader
SRAM	2KB (ATmega328P)
EEPROM	1KB(ATmega328P)
Velocidad de reloj	16MHZ
LED en la placa	13
Largo	68.6 mm
Ancho	53.4mm
Peso	25g

3.1.1.2 Pinout microcontrolador ATmega328P

El microcontrolador ATmega328 del Arduino UNO puede estar montado en la placa de dos formas: soldado a la placa (SMD) y desmontable (DIP). La única diferencia significativa es la posibilidad de ser desmontado de la placa en el caso de la versión DIP.

El *pinout* del ATmega328, en ambas versiones, se ilustra con la Figura 3.1y la Figura 3.2:

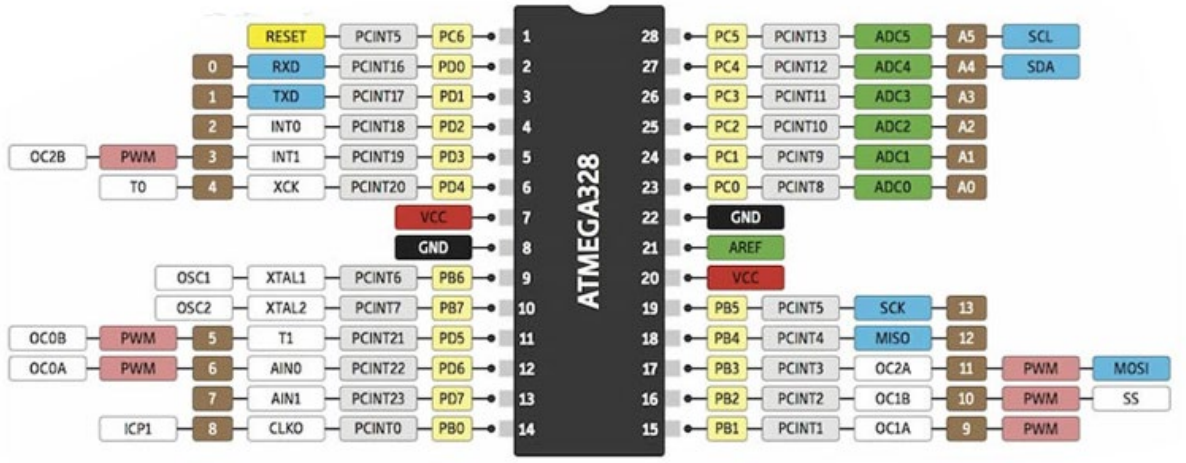


Figura 3.1 - Pinout ATmega328P DIP

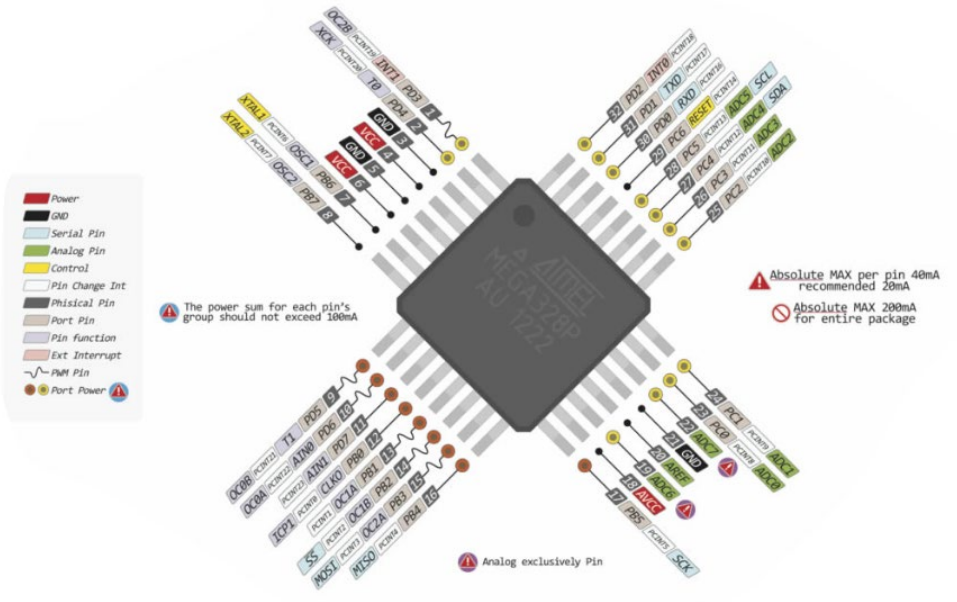
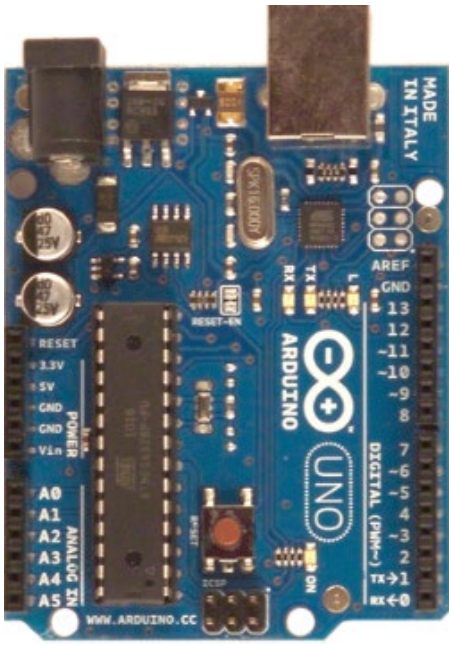


Figura 3.2 - Pinout ATmega328P SMD

3.1.1.3 Descripción pines Arduino UNO

En la subsección anterior se han expuesto los pines del microcontrolador usado en Arduino UNO, cuyo diseño se refleja en la Figura 3.3. Los pines digitales de Arduino a los que vamos a poder tener acceso pueden funcionar tanto de entrada (INPUT) como de salida (OUTPUT), configurándolo apropiadamente por software; mientras que los pines analógicos no hace falta configurarlos, pueden funcionar en cualquier momento como entrada o salida.

El chip ATmega328P incluye una resistencia *pull-up* de 20-50 kOhm en los pines de E/S, la cual puede configurarse para que esté activa o no mediante software.



Los pines más relevantes de la aplaca Arduino UNO para este proyecto son:

Pin 0-13: Entradas / Salidas digitales

Pin A0-A5: Entradas / Salida analógicas. Pueden ser configurados como pines digitales.

Pin 5V: Salida que proporciona 5V

Reset: Entrada que, a nivel bajo, provoca un reinicio de la placa.

Figura 3.3 - Placa Arduino UNO DIP

Con el fin de diferenciar aún más los pines digitales de los analógicos, aquellos pines que sean digitales serán referenciados con una "D" delante del número de pin (p.ej: Pin D0 = Pin 0).

Algunos pines digitales y analógicos tienen otras funciones adicionales. A continuación, Tabla 3.2, se detallan los pines digitales y analógicos de la placa Arduino UNO y sus funciones adicionales:

Tabla 3.2 – Asociación pines Microcontrolador / Arduino UNO y funciones adicionales

Pin Arduino	Pin Microcontrolador	Función adicional
D0	PD0	Recepción para comunicación UART
D1	PD1	Transmisión para comunicación UART
D2	PD2	Entrada de interrupción
D3	PD3	Entrada de interrupción Modulación por ancho de pulso PWM
D4	PD4	-
D5	PD5	Modulación por ancho de pulso PWM
D6	PD6	Modulación por ancho de pulso PWM
D7	PD7	-
D8	PB0	-
D9	PB1	Modulación por ancho de pulso PWM
D10	PB2	SS para comunicación SPI Modulación por ancho de pulso PWM
D11	PB3	MOSI para comunicación SPI Modulación por ancho de pulso PWM
D12	PB4	MISO para comunicación SPI
D13	PB5	SCK para comunicación SPI

		Tiene un led asociado en la placa al lado del pin
A0	PC0	-
A1	PC1	-
A2	PC2	-
A3	PC3	-
A4	PC4	SDA para comunicación I2C
A5	PC5	SCL para comunicación I2C

3.1.2 Arduino MEGA

Adicionalmente a la placa Arduino UNO, en este proyecto se ha utilizado la placa de desarrollo Arduino MEGA 2560, que emplea el microcontrolador ATmega2560.

Esta placa es apropiada para los casos que requieran más conexiones y/o requieran más pines con funciones especiales ya que, como se detallará en esta sección, la placa MEGA supera a la UNO en cuanto a pines digitales, analógicos, memoria ROM, …

3.1.2.1 Especificaciones técnicas

A continuación, Tabla 3.3, se exponen los datos técnicos facilitados por el fabricante:

Tabla 3.3 - Especificaciones técnicas Arduino MEGA

Microcontrolador	ATmega2560
Tensión de funcionamiento	5V
Tensión de alimentación recomendada	7-12V
Tensión de alimentación límite	6-20V
Pines Digitales de E/S	54
Pines Digitales PWM	15
Pines de entrada analógicos	16
Corriente DC por pin E/S	20mA
Corriente DC por pin a 3.3V	50mA
Memoria Flash	256KB (de los cuales 8KB son usados por el *bootloader*)
SRAM	8KB
EEPROM	4KB
Velocidad de reloj	16MHz
LED en la placa	13
Largo	101.52mm
Ancho	53.3mm
Peso	37g

3.1.2.2 Pinout microcontrolador ATmega2560

A diferencia del Arduino UNO, la placa MEGA sólo ofrece una versión, que se corresponde con el microcontrolador soldado a la placa. Como se ha mencionado, el microcontrolador que integra es el ATmega2560, cuyo *pinout* se proporciona en la Figura 3.4:

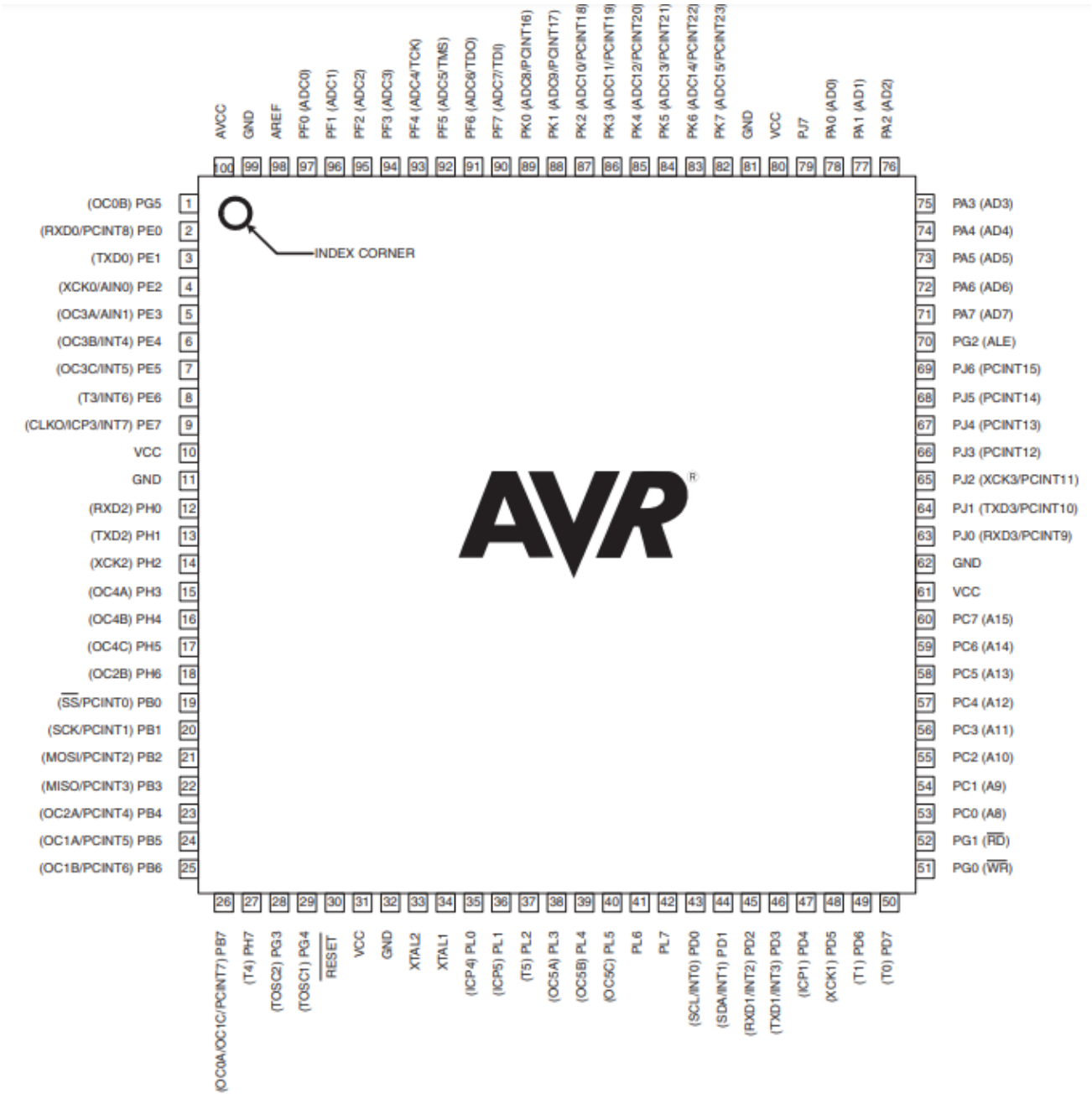


Figura 3.4 - Pinout ATmega2560

3.1.2.3 Descripción pines Arduino MEGA

Tal y como ocurre con el chip ATmega328P, el microcontrolador ATmega2560 incluye resistencias *pull-up* de 20-50 ohm en los pines de E/S que podremos activar o desactivar vía software. La funcionalidad de los pines tales como 5V, Reset, … son idénticas a las de Arduino UNO, por lo que no se explicarán de nuevo en este apartado. En la placa Arduino MEGA varía el número de pines digitales, siendo éstos desde el pin 0 hasta el pin 53 (para diferenciar que son pines digitales se usará una "D" delante del número del pin. p.ej: Pin D0=pin 0), y de pines Analógicos: A0-A15. Al igual que ocurría en el caso de la placa UNO, los pines analógicos pueden ser configurados como digitales o analógicos.

Figura 3.5 - Presentación placa Arduino MEGA 2560

La placa Arduino MEGA ofrece más pines con funciones adicionales. A continuación, Tabla 3.4, se detalla la correspondencia de los pines accesibles de la placa de Arduino con el microcontrolador y las funciones de éstos:

Tabla 3.4 - Asociación pines Microcontrolador / Arduino MEGA y funciones adicionales

Pin Arduino	Pin Microcontrolador	Función adicional
D0	PE0	Serial Port 0 RX
D1	PE1	Serial Port 0 TX
D2	PE4	Modulación por ancho de pulso PWM Interrupción externa 4
D3	PE5	Modulación por ancho de pulso PWM Interrupción externa 5
D4	PG5	Modulación por ancho de pulso PWM
D5	PE3	Modulación por ancho de pulso PWM
D6	PH3	Modulación por ancho de pulso PWM
D7	PH4	Modulación por ancho de pulso PWM
D8	PH5	Modulación por ancho de pulso PWM
D9	PH6	Modulación por ancho de pulso PWM
D10	PB4	Modulación por ancho de pulso PWM
D11	PB5	Modulación por ancho de pulso PWM
D12	PB6	Modulación por ancho de pulso PWM
D13	PB7	Modulación por ancho de pulso PWM Tiene asociado led en la placa
D14	PJ1	Serial Port 3 TX
D15	PJ0	Serial Port 3 RX
D16	PH1	Serial Port 2 TX
D17	PH0	Serial Port 2 RX
D18	PD3	Serial Port 1 TX Interrupción externa 3
D19	PD2	Serial Port 1 RX Interrupción externa 2

D20	PD1	SDA en comunicación I2C Interrupción externa 1
D21	PD0	SCL en comunicación I2C Interrupción externa 0
D22	PA0	-
D23	PA1	-
D24	PA2	-
D25	PA3	-
D26	PA4	-
D27	PA5	-
D28	PA6	-
D29	PA7	-
D30	PC7	-
D31	PC6	-
D32	PC5	-
D33	PC4	-
D34	PC3	-
D35	PC2	-
D36	PC1	-
D37	PC0	-
D38	PD7	-
D39	PG2	-
D40	PG1	-
D41	PG0	-
D42	PL7	-
D43	PL6	-
D44	PL5	Modulación por ancho de pulso PWM
D45	PL4	Modulación por ancho de pulso PWM
D46	PL3	Modulación por ancho de pulso PWM
D47	PL2	-
D48	PL1	-
D49	PL0	-
D50	PB3	MISO en comunicación SPI
D51	PB2	MOSI en comunicación SPI
D52	PB1	SCK en comunicación SPI
D53	PB0	SS en comunicación SPI
A0	PF0	-
A1	PF1	-
A2	PF2	-
A3	PF3	-
A4	PF4	-
A5	PF5	-
A6	PF6	-
A7	PF7	-
A8	PK0	-
A9	PK1	-
A10	PK2	-
A11	PK3	-
A12	PK4	-
A13	PK5	-
A14	PK6	-
A15	PK7	-

3.1.3 Sensores empleados

3.1.3.1 Módulo relé

En este proyecto se utilizan módulos relés con el fin de poder gestionar la alimentación de dispositivos que operan con tensiones de hasta 250V AC.

Un módulo relé, cuyo diagrama de conexión aparece en la Figura 3.6, dispones de tres pines de conexión hacia la placa de Arduino. Estos tres pines corresponden a la tensión de entrada del actuador (VCC), tierra (GND) y pin de entrada de datos o *signal* (DATA).

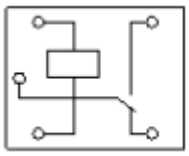


Figura 3.6 - Diagrama conexión relé

3.1.3.2 Sensor de temperatura y humedad DHT22

El sensor DHT22, también llamado AM2302, es un sensor ya calibrado para aportar una medida de temperatura y humedad lo más precisa posible. La tensión de alimentación debe encontrarse entre un mínimo de 3.3V y un máximo de 6V, siendo 5V lo recomendado.

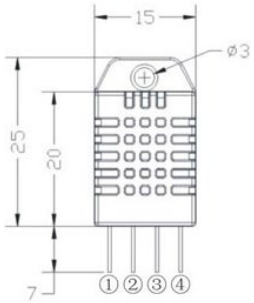


Figura 3.7 - Sensor DHT22

Este sensor dispone de 4 pines para su funcionamiento que, según la Figura 3.7, son:

1. Tensión de entrada: VDD (entre 3.3V y 6V)
2. Pin de datos: DATA
3. Tierra: GND
4. Tierra: GND

El conexionado del sensor DHT22 con la placa de desarrollo se muestra en la Figura 3.8:

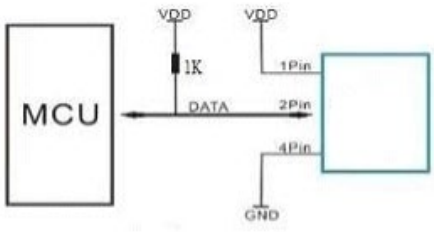


Figura 3.8 - Conexiones DHT22

Para la transmisión de los datos, el sensor DHT22 utiliza un único pin, el pin DATA. A través de esta conexión, el sensor envía 16 bits para transmitir la humedad, 16 bits de datos de temperatura y 8 bits de *check-sum.*

Los 16 bits de humedad se convierten a un número digital de tres dígitos, para posteriormente ser dividido entre diez y así obtener un decimal. Este dato será representado como un porcentaje.

Para la transmisión de temperatura se sigue el mismo procedimiento de la humedad relativa: Se transmiten 16 bits que serán convertidos en un decimal de tres dígitos del cual, al dividir entre 10, obtenemos el dato con un decimal de precisión. De los 16 bits transmitidos, el bit más significativo es utilizado para temperaturas inferiores a cero grados Celsius, poniéndose este bit a 1 si se cumple dicha condición.

Cabe destacar que los datos de temperatura pueden obtenerse tanto en grados Celsius como en grados Fahrenheit, dependiendo del comando utilizado para solicitar el dato.

Debido a que la línea de datos se encuentra a colector abierto, en nivel de reposo, la tensión en la línea será igual a la tensión de entrada, por lo que tanto el sensor como el equipo o placa de desarrollo únicamente podrán forzar nivel bajo. Una transmisión de datos desde el sensor hasta el equipo que solicita los datos sigue la siguiente secuencia (Figura 3.9):

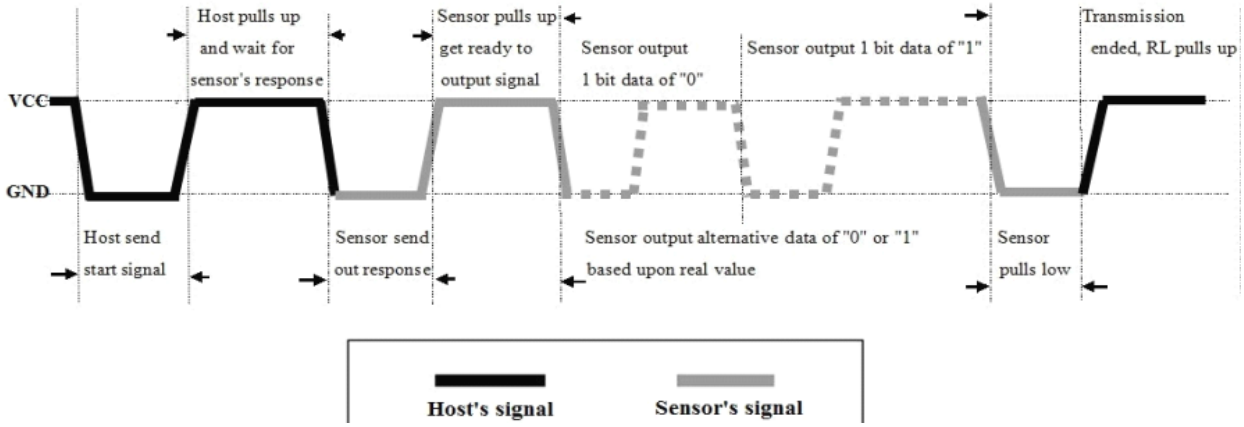


Figura 3.9 - Transmisión datos DHT22

Como se observa en la Figura 3.9, la transmisión se da siempre y cuando el equipo solicite datos del sensor. Para ello, pone la línea DATA a nivel bajo durante un tiempo mínimo de 1ms. Posteriormente, la línea debe ponerse a nivel alto durante un período de entre 20 y 40 microsegundos, que son seguidos de dos intervalos de 80 microsegundos. El sensor fuerza durante los primeros 80 microsegundos nivel bajo en la línea y durante los otros 80 microsegundos nivel alto para indicar al equipo que va a transmitir los datos almacenados en el sensor.

Para el envío de los datos del sensor, la transmisión de cada bit comienza con un nivel bajo durante 50 microsegundos, tras los cuales la línea pasa a nivel alto. La forma de diferenciar entre bits ceros y bits unos, es por el tiempo a nivel alto que dura la transmisión, siendo 0 si la línea a nivel alto está durante 26-28 microsegundos, y un 1 si está a nivel alto durante 70 microsegundos. El tiempo a nivel bajo entre bits es de 50 microsegundos.

Tras transmitir los 40 bits de datos del sensor, el fin de transmisión se indica cuando la línea se queda estable a nivel alto.

Cabe mencionar, que el modelo del sensor utilizado en este proyecto no aporta los datos de temperatura y humedad de manera continuada, sino que, expresamente, se le pide un dato a entregar. Además, el sensor usado puede estimar una medida de la sensación térmica basándose en los datos de temperatura y humedad que recoge.

3.1.3.3 Sensor de monóxido de carbono MQ-7

El sensor de monóxido de carbono MQ-7, Figura 3.10, se usa, tanto a nivel doméstico como industrial, con el fin de detectar fugas de gas.

MQ-7 utiliza el gas SnO2 para detectar la presencia de monóxido de carbono en el aire. El gas SnO2 tiene una conductividad más reducida en ausencia del monóxido de carbono, y aumenta la conductividad a medida que aumenta la concentración de CO.

El sensor posee un calentador interno que le sirve para medir el nivel de monóxido de carbono.

Las lecturas de monóxido se realizan a temperatura baja cuando el calentador es alimentado con 1.5V. Durante este intervalo de tiempo, el sensor acumula partículas de CO y otras partículas parásitas (cualquier partícula

que no sea del gas CO), aumentando o disminuyendo la conductividad de dicho sensor en presencias altas y bajas de monóxido de carbono, respectivamente.

Durante el tiempo en el que la temperatura del sensor es baja, se absorben partículas de gases parásitos. Estas partículas se queman con temperaturas altas del calentador; esto es, cuando el calentador es alimentado con una fuente de 5V.

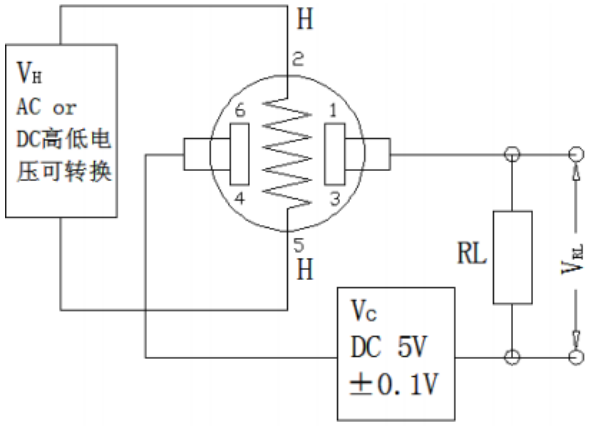


Figura 3.10 - Estructura sensor MQ-7

El sensor MQ-7 requiere de dos entradas de tensión, una entrada para alimentar el calentador (VH) y otra entrada para alimentar el circuito (VC).

Para este sensor, la tensión del calentador debe estar a 1.5V, con 0.1V de margen, para el intervalo de detección de CO. A 5V, con 0.1V de margen, para el estado de no detección.

El sensor tiene conectado una resistencia de carga RL con el fin de cerrar el circuito y realizar un divisor de tensión para poder leer los datos desde un microcontrolador.

3.1.3.4 Sensor de apertura MC38

El sensor MC38 está compuesto por un interruptor y un imán, y es usado para detectar la apertura de elementos tales como puertas, ventanas, …

El interruptor tiene conectado dos cables, uno para alimentación y otro para transmitir el estado, y se coloca en la parte fija de la puerta o ventana. El imán no tiene ningún cable conectado e irá en la parte móvil de la ventana o puerta.

En el caso de que no haya ningún imán próximo al interruptor, éste permanecerá abierto. En caso contrario, el interruptor se cerrará.

Según las especificaciones del fabricante, el interruptor se cierra con la existencia de un imán en una distancia inferior a 15mm.

3.1.3.5 Sensor de corriente ACS712

El sensor de corriente ACS712, Figura 3.11, puede ser adquirido en tres versiones, según se necesite trabajar con corrientes de 5, 20 o 30 Amperios.

El sensor ACS712 utiliza un sensor de efecto Hall que detecta el campo magnético producido por la corriente que circula por el cable que se está midiendo, para aportar un voltaje proporcional. Es un sensor invasivo, lo que significa que debe estar físicamente en serie en el circuito.

Dependiendo del sensor usado, la sensibilidad varía. Para este proyecto se ha utilizado el sensor que soporta 20 Amperios, cuya sensibilidad, dada por el fabricante, es de 100mV/A.

Para una corriente de 0 Amperios el sensor ACS712 genera un valor en la salida de 2.5 Voltios, incrementándose de forma proporcional a la sensibilidad. La ecuación de la salida del sensor es por la tanto una recta cuya ecuación es: V=sensibilidad·corriente + 2.5

Figura 3.11 - Sensor ACS712

3.2 nodeMCU

NodeMCU, Figura 3.12, es una plataforma abierta de IoT que incluye tanto firmware, funcionando sobre el Sistema en Chip (en adelante *SoC*), ESP8266, desarrollado por Espressif System, como hardware, basado en el módulo ESP-12.

El *SoC* ESP8266 ofrece una solución de red Wifi completa y autónoma, pudiendo dotar de acceso inalámbrico a cualquier diseño basado en microcontroladores mediante cualquiera de las interfaces que proporciona (SPI, I2C, UART)

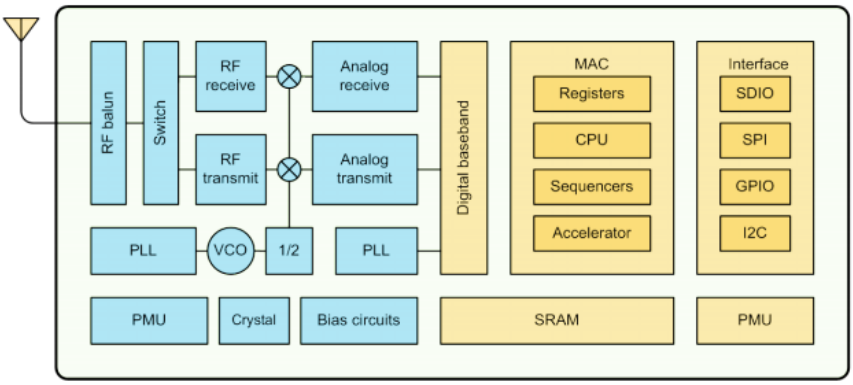


Figura 3.12 - Diagrama de bloques del chip ESP8266

El *SoC* ESP8266 integra el microcontrolador Tensilica L106 de 32 bits, el cual opera con una señal de reloj de 80MHz, pudiendo llegar hasta los 160MHz. Las principales características del ESP8266 son:

- MCU de 32 bits de bajo consumo (Tensilica L106)
- Módulo Wifi a 2.3 GHz
- SRAM de 36 kB aproximadamente
- SPI Flash externa de hasta 16Mbyte
- Entrada analógica de 10 bits
- 17 pines de entrada y salida GPIO

El módulo ESP12-E incorpora la memoria flash externa mencionada anteriormente y, además, permite el acceso a los pines del *SoC* y al microcontrolador.

En el nivel superior, nos encontramos ya con el kit o placa de desarrollo nodeMCU, que nos permite acceder a los pines y programar el microcontrolador. Las características a destacar de la placa de desarrollo son:

- Tensión operacional a 3.3V
- Conectividad micro USB
- Protocolo 802.11 b/g/n
- Antena construida sobre el módulo ESP-12E
- Capacidad de usar PWM, I2C, SPI, UART, 1 pin analógico
- Compatibilidad con Arduino IDE
- Soporte de los lenguajes de programación Lua y Arduino C.

A continuación, Figura 3.13, se ilustra la accesibilidad de los pines de la placa de desarrollo y sus funciones:

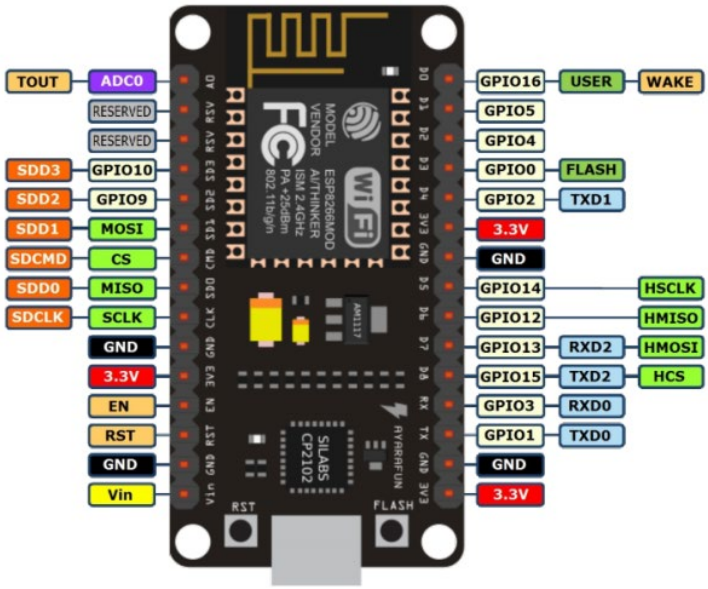


Figura 3.13 - Pinout nodeMCU

En total, contamos con:

- 13 pines digitales (D0-D12)
- 1 pin analógico (A0)
- 3 pines operacionales a 3.3V
- 1 pin operacional a 5V (Vin)
- 4 pines de tierra GND

La nomenclatura GPIOX se debe a que indica a qué pin del ESP8266 está conectado el pin de la placa de desarrollo. La razón de que los pines GPIO no sigan una numeración ordenada se debe precisamente a que algunos pines del ESP8266 se usan para conectarse a una memoria flash externa, tal y como se comentó anteriormente.

Los pines de alimentación sirven para alimentar tanto a sensores como a la propia placa. Si la placa es alimentada con 3.3V, el pin de 5V de salida quedaría inservible; sin embargo, si la placa se alimenta a través del puerto microUSB, se le estarían proporcionado 5V a la placa de desarrollo que, mediante un regulador de voltaje, aporta 3.3V a los respectivos pines y 5V al pin Vin.

3.3 Raspberry Pi 3

Raspberry Pi es un ordenador de placa única de bajo coste que, en su tercera versión y modelo B, está compuesto por un procesador Broadcom a 1.20GHz de velocidad de reloj y una RAM de 1GB.

El modelo Raspberry Pi 3 B integra conectividad Wifi sin necesidad de elementos externos y aporta los siguientes periféricos:

- 48 pines GPIO.
- 2 pines con soporte para I2C
- 2 pines UART
- 2 pines SD/SDIO
- 4 puertos USB 2.0

- 1 interfaz Ethernet
- 1 interfaz HDMI
- 1 interfaz micro USB
- 1 puerto DPI
- 1 interfaz NAND
- 1 interfaz de cámara CSI de 4 carriles
- 1 interfaz de cámara CSI de 2 carriles
- 1 interfaz de cámara DSI de 4 carriles
- 1 interfaz de cámara DSI de 4 carriles

Los modelos Raspberry Pi no incluyen memoria, teniendo que aportarse ésta de forma externa mediante una tarjeta microSD.

4 PROTOCOLOS DE COMUNICACIÓN

Every once in a while, a new technology, an old problem, and a big idea turn into an innovation.

- Dean Kamen -

En el presente capítulo se habla en detalle de los dos principales protocolos que intervienen en este proyecto: MQTT e I2C.

4.1 I2C

El protocolo *Inter-Integrated Circuit Communicaction* (I2C) es un protocolo serie síncrono, el cual sólo requiere de dos conexiones en cada dispositivo para la comunicación *half-duplex*. Una conexión para la transmisión serie de datos (SDA) y otra conexión para la señal de reloj (SCL).

4.1.1 Introducción

En el protocolo I2C cada dispositivo que forma parte de la comunicación está identificado por una dirección única de 7 bits (A6-A0) o 10 bits (A9-A0). Durante una transmisión de datos, cualquier dispositivo puede actuar como transmisor, enviando datos por el bus SDA, o como receptor, recibiendo datos por el bus SDA.

Adicionalmente, los dispositivos también se diferencian en maestros y esclavos. El maestro es el dispositivo que genera la señal de reloj y es el que inicia la transferencia de datos. El esclavo es el dispositivo que actúa según lo ordenado por el maestro, enviando o recibiendo datos.

Una de las grandes ventajas de este protocolo es la posibilidad de tener varios esclavos, usando únicamente las dos conexiones mencionadas anteriormente (SDA y SCL); esto es posible gracias a que cada dispositivo está identificado mediante una dirección única.

4.1.2 Bus SDA y SCL

Ambas líneas o buses son bidireccionales y a colector abierto, estando conectadas a resistencias *pull-up,* como se muestra en la Figura 4.1, que hacen que en el estado de reposo ambas se encuentren a nivel alto; por esto, los dispositivos sólo podrán forzar nivel bajo en las líneas.

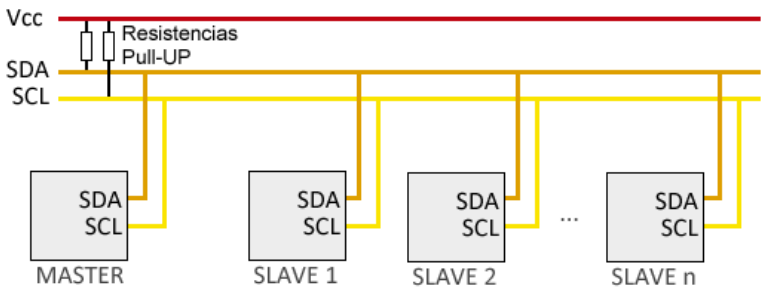


Figura 4.1 - Conexionado Maestro/Esclavo I2C

El protocolo I2C permite varias configuraciones, pudiendo los datos ser transmitidos a velocidades de hasta:

- 100 kb/s en la configuración estándar.
- 400 kbit/s en *Fast-mode*.
- 1 Mbit/s en *Fast-mode Plus*.

Otra de las ventajas de este protocolo es que no establece un voltaje para el nivel alto y el nivel bajo, sino que se usa como referencia el valor de la tensión de las líneas SDA y SCL, estipulándose el valor alto a partir del 70% de dicho nivel y el nivel bajo si es menor del 30%.

4.1.3 Transmisión de datos

Toda transmisión de datos, la cual es originada y terminada por el maestro, comienzan cuando se da la condición de START, y terminan con la condición de STOP (Figura 4.2).

La condición de START se produce con un cambio de la línea SDA de nivel alto a nivel bajo, estando la línea SCL a nivel alto.

La condición de STOP se establece cuando la línea SDA cambia de nivel bajo a nivel alto, estando SCL a nivel alto.

En una transmisión, desde que se produce la condición de START, y hasta que no se dé la condición de STOP, se considerará que el bus está ocupado. Entre una condición de START y una de STOP pueden darse varias condiciones de START con el fin de enviar más bits en una misma transmisión.

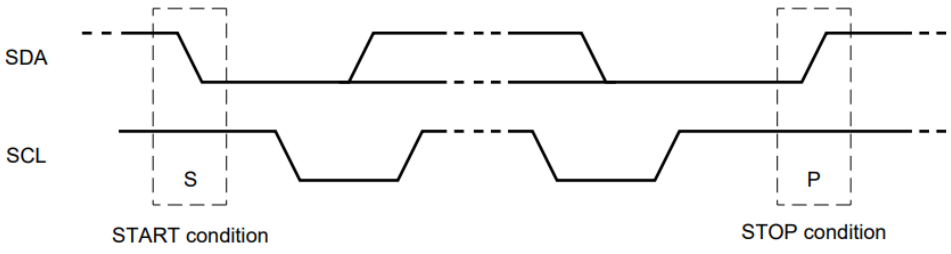


Figura 4.2 - Condiciones de START y STOP

Otro aspecto por destacar es que durante la transmisión de los datos a través de la línea SDA, la transmisión debe ser estable durante el intervalo a nivel alto del ciclo de reloj, por lo que el cambio de nivel alto a bajo y viceversa de la línea SDA se realiza cuando la señal de reloj está a nivel bajo (Figura 4.3).

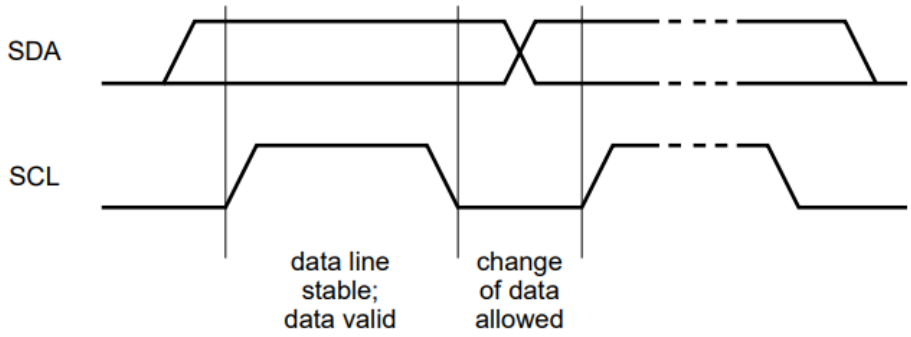


Figura 4.3 - Transmisión bit

4.1.3.1 Dirección de esclavo y bit de escritura lectura

Después de la condición de START, el dispositivo que actúa como maestro escribe la dirección del esclavo con el que va a interactuar. Esta dirección puede estar formada por 7 bits o por 10 bits (en este proyecto no se realizarán conexiones con direcciones de 10 bits, por lo que este caso queda fuera de estudio en el presente documento), siendo transmitido primero el bit más significativo.

Los siete bits de dirección son seguidos por un octavo bit, bit de dirección de dato R/W. Si el bit R/W se encuentra a nivel alto, se interpreta que el maestro va a enviar los siguientes 8 bits al esclavo especificado por la dirección previa; la operación es de escritura. Si el bit R/W está a nivel bajo, el maestro está solicitando datos al esclavo indicado; la operación es de lectura.

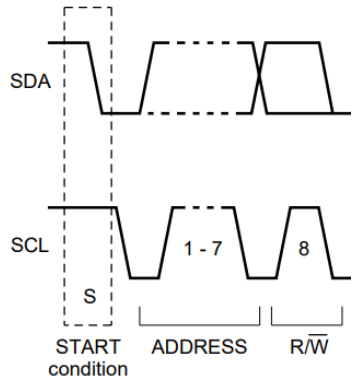


Figura 4.4 - Dirección de esclavo y bit R/W

El protocolo I2C establece una serie de direcciones reservadas, por lo que no podrán ser asignadas a ningún dispositivo. Estas direcciones son:

Tabla 4.1 - Direcciones reservadas en I2C

Dirección de esclavo	Bit R/W	Descripción
0000 000	0	Dirección general
0000 000	1	Byte START
0000 001	X	Dirección CBUS
0000 010	X	Reservado para diferentes formatos de bus
0000 011	X	Reservado
0000 1XX	X	Master en modo HS
1111 1XX	1	ID de dispositivo
1111 0XX	X	Direccionamiento esclavo de 10 bits

Siendo 1=Nivel alto, 0=Nivel bajo, X= Nivel alto o nivel bajo.

4.1.3.2 Formato de Byte

El protocolo I2C establece que todo Byte de datos enviado a través de la línea SDA debe tener una longitud fija de 8 caracteres seguido por un bit de ACK. Si se desea enviar más datos durante la transmisión, debe de producirse una condición de START posterior al bit de ACK.

Los datos son transferidos con los bits más significativos (MSB) al comienzo. Si un esclavo no puede continuar recibiendo o transmitiendo otro byte completo de datos porque deba atender otra acción (como la atención de una interrupción), puede forzar la señal de reloj SCL a nivel bajo, forzando al otro extremo a esperar. La transferencia continúa cuando el esclavo esté listo para enviar/recibir otro byte de datos y libere la línea SCL.

El dispositivo que escribe el byte de datos viene dado por el bit R/W explicado anteriormente. Si la operación es de escritura, el byte de datos es escrito por el maestro, en caso el caso contrario, la operación es de lectura, por lo que es el esclavo el que envía los datos al maestro; el esclavo escribe el byte de datos.

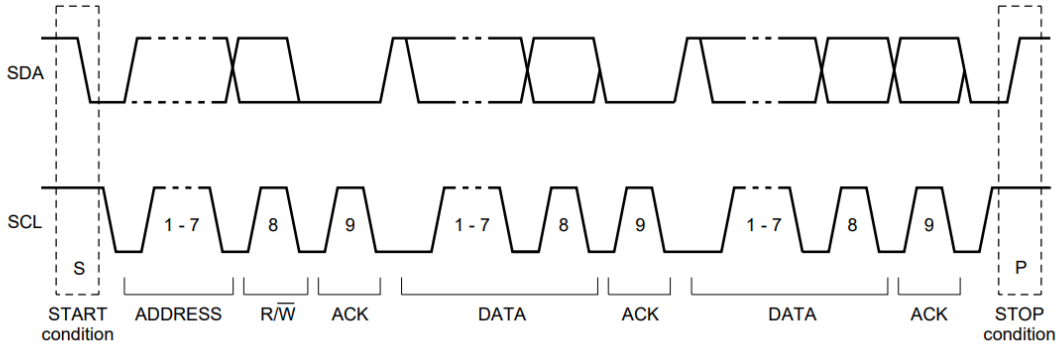


Figura 4.5 - Formato de trama

4.2 MQTT

MQTT (*Message Queue Telemetry Transport*) fue creado en 1999 por dos ingenieros de IBM y Arcom (ahora Eurotech), las cuales, tenían la necesidad de crear un nuevo protocolo con el fin de conectar sensores de oleoductos con satélites a través de redes pocos fiables. Este nuevo protocolo debía tener un consumo muy bajo y, además, requerir poco ancho de banda.

En 2011, IBM y Eurotch donaron MQTT al protocolo de Eclipse “Paho”, y en 2014 se convierte en un estándar abierto de OASIS, consorcio sin ánimo de lucro que impulsa el desarrollo y adaptación de estándares abiertos para la sociedad de la información global. Finalmente, en 2016 el estándar de OASIS es aprobado por la ISO (ISO/IEC 20922).

4.2.1 Introducción

MQTT es un protocolo de transporte de mensajería máquina a máquina (M2M). Este protocolo está basado en el modelo cliente-servidor publicador/suscriptor y que, por las características mencionadas anteriormente, hace que se un protocolo de especial interés en el contexto IoT.

Este protocolo se ejecuta sobre la capa TCP / IP del modelo OSI, o sobre otros protocolos de red que proporcionen conexiones bidireccionales ordenadas y sin pérdidas. Algunas de las funcionalidades más destacadas del protocolo son:

- Uso del patrón publicador/suscriptor que puede proporcionar mensajes de uno a muchos.
- Transporte de mensajes con independencia de la carga útil de los mismos.
- Uso de paquetes para publicar o suscribirse a mensajes de aplicación.
- Mínima sobrecarga del tráfico de red.
- Tres calidades de servicio para la entrega de mensajes: *At most once*, *at least once*, *exactly once*.
- Mecanismo de notificación cuando se produce una desconexión atípica

4.2.2 Terminología

Para entender el funcionamiento del protocolo MQTT es necesario comprender ciertos términos a los cuales se hacen referencia durante toda la explicación del protocolo.

Tabla 4.2 - Terminología MQTT

Concepto	Descripción
Conexión	El constructor proporcionado por el protocolo de transporte que está siendo usado por MQTT conecta el cliente con el servidor, y proporciona los medios para enviar flujos de bytes bidireccionales, ordenados, y sin pérdidas.
Mensaje de aplicación	Datos transportados por el protocolo MQTT a través de la red procedentes de una aplicación. Este tipo de mensaje contiene: Carga útil, calidad de servicio, propiedades, *topic*.
Topic	Cadena de caracteres que identifican una serie de flujos de bytes.
Cliente	Un cliente MQTT realiza las siguientes acciones: • Abre conexión contra el servidor. • Publica mensajes de aplicación para informar a otros clientes. • Se suscribe para solicitar los mensajes de aplicación en los que esté interesado. • Anula suscripción para dejar de recibir mensajes de aplicación. • Cierra la conexión con el servidor
Servidor	Programa o dispositivo que actúa de intermediario entre los clientes que publican mensajes de aplicación y los clientes que se han suscrito a dichos flujos de mensajes. Un servidor: Acepta conexiones de clientes. Acepta mensajes de aplicación publicados por clientes. Procesa suscripciones y anulaciones solicitadas por los clientes. Reenvía mensajes de aplicación que coincidan con el flujo de mensajes suscritos por los clientes. Cierra la conexión desde el cliente.
Suscripción	Una suscripción conlleva un filtrado de *Topic* y una *QoS* máxima. Una suscripción está asociada con una única sesión. Una sesión puede contener más de una suscripción. cada suscripción dentro de una sesión tiene un filtro de *Topic* diferente.
Sesión	Interacción entre un cliente y un servidor.
Filtro de *Topic*	Expresión contenida en una suscripción para indicar el interés del cliente en las publicaciones de uno o más *Topic*.
Paquete de control MQTT	Paquete de información enviado a través de la red. MQTT define quince tipos diferentes de paquetes de control.
Will Message	Mensaje de aplicación que es publicado por el servidor después de que la conexión de red se haya cerrado de forma anómala.

4.2.3 Paquete de control MQTT

En esta subsección se expondrán los principales mensajes de una conexión MQTT.

4.2.3.1 Formato

El protocolo MQTT operando mediante el intercambio en un sentido de una serie de paquetes de control. Este apartado describe el formato de estos paquetes.

Un paquete de control MQTT se compone de hasta tres partes, siempre en el orden siguiente:

Tabla 4.3 - Formato paquete MQTT

Cabecera fija: Presente en todos los paquetes de control
Cabecera variable: Presente en algunos paquetes de control
Carga útil o *payload*: Presente en algunos paquetes de control

4.2.3.1.1 Cabecera fija

Todo paquete de control MQTT contiene una cabecera fija con el siguiente formato:

Tabla 4.4 - Cabecera Fija MQTT

Bit / Byte	7	6	5	4	3	2	1	0
1	Tipo de paquete de control MQTT				*Flags* específicos para cada tipo de paquete de control MQTT			
2	Longitud restante							

Tipo de paquete de control

Se corresponden con los bits del 7 al 4 del primer byte del paquete. Estos 4 bits corresponden a un valor sin signo y cuyos valores y significados son los siguientes:

Tabla 4.5 - Tipos de paquetes de control MQTT

Nombre	Valor	Dirección de envío	Descripción
Reservado	0	-	Reservado
CONNECT	1	Cliente → Servidor	Petición de conexión con el servidor por parte del cliente
CONNACK	2	Servidor → Cliente	Confirmación por parte del servidor para la conexión
PUBLISH	3	Cliente → Servidor Servidor → Cliente	Mensaje de publicación
PUBACK	4	Cliente → Servidor Servidor → Cliente	Confirmación para la publicación (QoS 1)
PUBREC	5	Cliente → Servidor Servidor → Cliente	Publicación recibida (Paso 1 QoS 2)
PUBREL	6	Cliente → Servidor Servidor → Cliente	Confirmación publicación (Paso 2 QoS 2)
PUBCOMP	7	Cliente → Servidor	Publicación completa (Paso 3 QoS 3)
SUBSCRIBE	8	Servidor → Cliente	Petición de suscripción
SUBACK	9	Servidor → Cliente	Confirmación suscripción
UNSUBSCRIBE	10	Cliente → Servidor	Petición de anulación de suscripción
UNSUBACK	11	Servidor → Cliente	Confirmación de anulación de suscripción
PINGREQ	12	Cliente → Servidor	Petición de PING
PINGRESP	13	Servidor → Cliente	Respuesta de PING
DISCONNECT	14	Cliente → Servidor Servidor → Cliente	Notificación de desconexión
AUTH	15	Cliente → Servidor Servidor → Cliente	Intercambio de autenticación

Flags específicos del tipo de paquete de control

Los últimos 4 bits del primer byte de la cabecera fija son usados para indicar parámetros correspondientes al tipo de mensaje. Actualmente sólo son configurables las *flags* de los mensajes PUBLISH; el resto de mensajes tienen *flags* reservadas (serán usadas en una futura versión del protocolo) y deben de tener unos valores específicos para cada tipo de mensaje.

Según el tipo de paquete, los bits correspondientes a las *flags* se corresponden o tienen que tomar los siguientes valores:

Tabla 4.6 - Flags paquetes de control MQTT

Paquete de control MQTT	Flags de la cabecera fija	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reservado	0	0	0	0
CONNACK	Reservado	0	0	0	0

PUBLISH	Usado en MQTT v5.0	Entrega duplicada de paquete PUBLISH	QoS		Retención mensaje PUBLISH
PUBACK	Reservado	0	0	0	0
PUBREC	Reservado	0	0	0	0
PUBREL	Reservado	0	0	1	0
PUBCOMP	Reservado	0	0	0	0
SUBSCRIBE	Reservado	0	0	1	0
SUBACK	Reservado	0	0	0	0
UNSUBSCRIBE	Reservado	0	0	1	0
UNSUBACK	Reservado	0	0	0	0
PINGREQ	Reservado	0	0	0	0
PINGRESP	Reservado	0	0	0	0
DISCONNECT	Reservado	0	0	0	0
AUTH	Reservado	0	0	0	0

Longitud restante

El campo de longitud restante es un entero variable de bytes, que representa el número de bytes que quedan dentro del paquete de control actual, incluyendo los datos de la cabecera variable y la carga útil. No incluye los bytes utilizados para codificar este mismo campo.

El tamaño del paquete es el número total de bytes en un paquete de control de MQTT; esto es, la longitud de la cabecera fija más el campo "longitud restante"

4.2.3.1.2 Cabecera variable

Situada entre la cabecera fija y la carga útil, el contenido de la cabecera variable depende del tipo de paquete, aunque suele ser habitual que posea el campo identificador de paquete.

Identificador de paquete.

El campo identificador de paquete es un entero de dos bytes, y es utilizado en los siguientes paquetes de control:

Tabla 4.7 - Identificador de paquete MQTT

Paquete de Control MQTT	Identificador de paquete
CONNECT	NO
CONNACK	NO
PUBLISH	SI (Siempre que el QoS sea superior a 0)
PUBACK	SI
PUBREC	SI
PUBREL	SI
PUBCOMP	SI
SUBSCRIBE	SI
SUBACK	SI
UNSUBSCRIBE	SI
UNSUBACK	SI
PINGREQ	NO
PINGRESP	NO
DISCONNECT	NO
AUTH	NO

Propiedades

Se trata del último campo de la cabecera variable. Este campo está compuesto por una "longitud de propiedades" y la correspondiente propiedad.

- **Longitud de propiedad**

Codificada como una variable entera de un byte, no incluye los bytes usados para codificar el propio campo, sólo incluye la longitud de la propiedad. Si no hubiera ninguna propiedad que indicar, esta longitud tendría un valor nulo.

- **Propiedad**

Una propiedad consta de un identificador que define su uso y tipo de datos, seguido de un valor.

El identificador se codifica como un entero variable de un byte. Un paquete de control que contiene un identificador que no es válido para su tipo de paquete, o que contiene un valor que no es del tipo de datos especificado, se considera un paquete con formato incorrecto

Tabla 4.8 - Valores campo propiedades paquete de control MQTT

Identificador		Nombre (uso)	Tipo	Paquete / Will Properties
Dec	Hex			
1	0x01	Indicador del formato de la carga útil	Byte	PUBLISH, Will Properties
2	0x02	Intervalo de expiración del mensaje	Entero de cuatro bytes	PUBLISH, Will Properties
3	0x03	Tipo de contenido	Cadena codificada con UTF-8	PUBLISH, Will Properties
8	0x08	Respuesta de topic	Cadena codificada con UTF-8	PUBLISH, Will Properties
9	0x09	Datos	Datos en formato binario	PUBLISH, Will Properties
11	0x0B	Identificador de suscripción	Entero variable en bytes	PUBLISH, SUBSCRIBE
17	0x11	Intervalo de expiración de sesión	Entero de 4 bytes	CONNECT, CONNACK, DISCONNECT
18	0x12	Identificador asignado al cliente	Cadena codificada con UTF-8	CONNACK
19	0x13	Keep Alive del servidor	Entero de dos bytes	CONNACK
21	0x15	Método de autenticación	Cadena	CONNECT, CONNACK, AUTH
22	0x16	Datos de autenticación	Binario	CONNECT, CONNACK, AUTH
23	0x17	Información de problema en petición	Byte	CONNECT
24	0x18	Intervalo Will Delay	Entero de cuatro bytes	Will Properties
25	0x19	Información de petición de respuesta	Byte	CONNECT
26	0x1A	Información de respuesta	Cadena codificada con UTF-8	CONNACK
28	0x1C	Referencia del servidor	Cadena codificada con UTF-8	CONNACK, DISCONNECT
31	0x1F	Cadena de información de razón	Cadena codificada con UTF-8	CONNACK, PUBACK, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, AUTH
33	0x21	Máximo de recepción	Entero de dos bytes	CONNECT, CONNACK
34	0x22	Máximo de Topic Alias	Entero de dos bytes	CONNECT, CONNACK
35	0x23	Topic Alias	Entero de dos bytes	PUBLISH
36	0x24	Máxima QoS	Byte	CONNACK
37	0x25	Retain available***	Byte	CONNACK
38	0x26	Propiedades del usuario	Pareja de cadenas	CONNECT, CONNACK, PUBLISH,

			codificadas en UTF-8	Will Properties, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, DISCONNECT, AUTH
39	0x27	Tamaño máximo del paquete	Entero de 4 bytes	CONNECT, CONNACK
40	0x28	Disponibilidad de suscripción Wildcard	Byte	CONNACK
41	0x29	Identificación de suscripción disponible	Byte	CONNACK
42	0x2A	Disponibilidad de suscripción compartida	Byte	CONNACK

4.2.3.1.3 Payload

El *payload* o la carga útil se encuentra al final del paquete de algunos mensajes MQTT. En el caso del mensaje PUBLISH, el *payload* corresponde al mensaje de aplicación.

Tabla 4.9 - Carga útil en paquetes de control MQTT

Paquete de control MQTT	Payload
CONNECT	Obligatorio
CONNACK	Ninguno
PUBLISH	Opcional
PUBACK	Ninguno
PUBREC	Ninguno
PUBREL	Ninguno
PUBCOMP	Ninguno
SUBSCRIBE	Obligatorio
SUBACK	Obligatorio
UNSUBSCRIBE	Obligatorio
UNSUBACK	Obligatorio
PINGREQ	Ninguno
PINGRESP	Ninguno
DISCONNECT	Ninguno
AUTH	Ninguno

4.2.3.1.4 Reason Code

Un *Reason Code* es un byte sin signo que indica el resultado de una operación. El valor habitual para indicar una operación satisfactoria es 0, aunque cualquier valor por debajo del 0x80 indica también que la operación se ha realizado correctamente, y un valor igual o superior a 0x80 indica operación fallida.

Los paquetes de control CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, DISCONNECT y AUTH tienen un único *Reason Code*, situado en la cabecera variable. Los paquetes SUBACK y UNSUBACK contienen una lista de uno o más *Reason Code* en el *Payload*.

4.2.3.2 Tipos de paquetes

4.2.3.2.1 CONNECT

Después de establecer conexión entre el cliente y el servidor, el primer paquete enviado con origen el cliente y destino el servidor es obligatoriamente un paquete CONNECT. Este mensaje sólo puede ser enviado una vez por el cliente.

El *payload* del paquete contiene uno o más campos codificados, los cuales identifican unívocamente al cliente, *topic*, *payload*, nombre de usuario y contraseña. Todos los campos anteriores, a excepción el identificador del cliente, son opcionales y la presencia de los mismos estará indicada por las respectivas banderas en una cabecera variable.

4.2.3.2.2 CONNACK

Este paquete es enviado por el servidor como respuesta a un paquete CONNECT.

El servidor envía un paquete CONNACK con un valor del campo *Reason Code* igual a 0x00 antes de enviar cualquier otro paquete distinto de un paquete AUTH. Únicamente se envía un CONNACK durante una conexión.

Si el cliente no recibe el paquete CONNACK proveniente del servidor después de un intervalo de tiempo, el cliente cierra la conexión. Este tiempo de espera depende de la aplicación que esté haciendo uso del protocolo MQTT.

4.2.3.2.3 PUBLISH

Paquete bidireccional entre cliente y servidor para transportar un mensaje de aplicación. En este paquete se indica el nombre del *topic*, así como el propio mensaje de la aplicación.

4.2.3.2.4 PUBACK

Paquete enviado como respuesta a un paquete PUBLISH. Este paquete sólo se envía si el paquete PUBLISH recibido con anterioridad tiene marcado un QoS igual a 1.

4.2.3.2.5 PUBREC

Paquete enviado como respuesta a un paquete PUBLISH con QoS igual a 2. Este paquete es el segundo paquete enviado del QoS de nivel 2.

4.2.3.2.6 PUBREL

Paquete de respuesta al paquete PUBREC. Es el tercer paquete de la QoS 2.

4.2.3.2.7 PUBCOMP

El paquete PUBCOMP es enviado como respuesta al paquete PUNREL. Este es el tercer y último paquete de QoS 2.

4.2.3.2.8 SUBSCRIBE

El paquete SUBSCRIBE es enviado del cliente al servidor con el fin de crear una o más suscripciones. Cada suscripción registra el interés del cliente por recibir las publicaciones en uno o más *topics*.

El servidor reenvía los mensajes de aplicación que fueron publicados con los *topics* a los que el cliente se haya suscrito mediante paquetes PUBLISH al cliente.

El paquete SUBSCRIBE también indica, para cada suscripción, el QoS máximo con el que el servidor puede enviar mensajes de aplicación al cliente.

4.2.3.2.9 SUBACK

Este paquete es enviado con origen el servidor, y como destino el cliente para informar que se ha recibido y se ha procesado el paquete SUBSCRIBE enviado por el cliente.

Un paquete SUBACK contiene una lista de Reason Codes que especifica el QoS máximo que puede ser garantizado, o el error que se ha encontrado en cada suscripción que fue solicitada por el cliente.

4.2.3.2.10 UNSUBSCRIBE

Paquete enviado por el cliente hacia el servidor para dejar de recibir mensajes del *topic* indicado en el paquete.

4.2.3.2.11 UNSUBACK

El paquete UNSUBACK es enviado por el servidor hacia el cliente para confirmar a este de que se ha recibido el mensaje UNSUBSCRIBE.

4.2.3.2.12 PINREQ

Paquete con origen en el cliente y destino hacia el servidor usado para diferentes propósitos:

- Indicar al servidor que el cliente sigue activo en el caso de que no se haya tenido que enviar ningún paquete de control MQTT del cliente hacia el servidor.

- Petición de respuesta al cliente para comprobar que el servidor sigue estando activo.
- Comprobación de que la conexión de red sigue estando activa

4.2.3.2.13 PINGRESP

Paquete enviado del servidor al cliente en respuesta a un paquete PINGREQ.

4.2.3.2.14 DISCONNECT

Este paquete es el último paquete de control MQTT enviado del cliente al servidor o viceversa.

En este paquete se indica la razón del cierre de conexión de red. Antes de cerrar la conexión de red el cliente o el servidor debe enviar un mensaje DISCONNECT. Si se cierra la conexión sin enviar previamente este paquete, se publica un mensaje WILL Message con el que el servidor cierra la conexión.

5 DESCRIPCIÓN DE LA SOLUCIÓN DESARROLLADA

Genius is one percent inspiration and ninety-nine percent perspiration

- Thomas Edison -

En este capítulo se detallará el planteamiento, los pasos seguidos y la configuración realizada para conseguir la automatización de la vivienda.

Conforme se fue planteando el sistema domótico que se expone en este capítulo, fueron surgiendo una serie de restricciones las cuales hicieron que el desarrollo del sistema tuviera que estructurarse en torno a ellas; además, el código de los dispositivos se ha elaborado de forma que pueda ser modificado fácilmente en un futuro para poder añadir más sensores, actuadores, ...

5.1 Comunicación entre dispositivos

El sistema de automatización propuesto en este proyecto, Figura 5.1, gira en torno a tres elementos hardware principales, los cuales fueron expuestos en el capítulo 3, y dos protocolos, explicados en el capítulo 4, que permitirán la comunicación entre ellos.

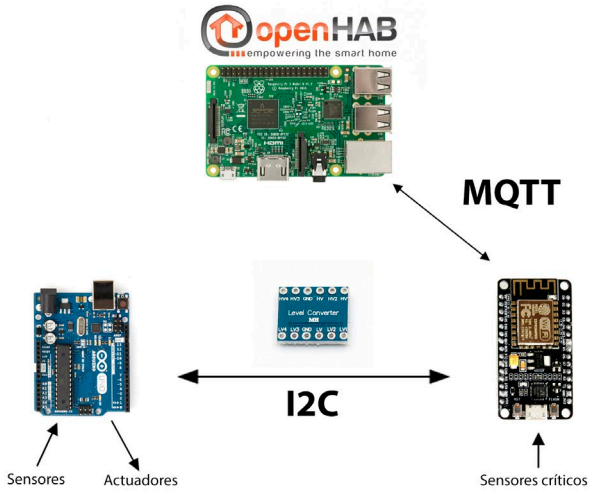


Figura 5.1 - Hardware principal y protocolos de comunicación

La comunicación entre dispositivos se ha diseñado de forma que sea lo más reutilizable posible; esto quiere decir que, para distintos grupos de parejas Arduino - nodeMCU, el código a modificar será mínimo. Por esta razón se han establecido las consideraciones expuestas en las siguientes subsecciones.

5.1.1 Comunicación MQTT

En la comunicación MQTT entre la placa nodeMCU y Raspberry Pi, se ha considerado que la forma más cómoda en cuanto a reutilización de código se refiere es estableciendo los *topics* de los mensajes de una forma genérica. De esta forma, los *topics* utilizados siguen la estructura: XNNN-P, dónde:

- X: Indica si el *topic* del mensaje involucra un pin analógico o digital. Puede tomar los valores "A" o "D".
- NNN: Tres dígitos, de los cuales los dos primeros hacen referencia al pin que está siendo utilizado. El tercer dígito toma un valor distinto de cero en el caso que el sensor conectado a dicho pin pueda aportar más de un dato. Este es el caso del sensor DHT22, que aporta datos de temperatura y humedad.
- P: Es el código que diferencia a un grupo Arduino-nodeMCU de otro, y hace que los *topics* sean diferentes entre un grupo y otro.

En una sesión MQTT, si la placa nodeMCU actúa como suscriptora de un *topic* determinado, ésta recibirá mensajes PUBLISH con los *topics* oportunos; sin embargo, si actúa como publicadora, enviará mensajes tipo PUBLISH.

Se ha estimado oportuno que los mensajes enviados por parte de la placa nodeMCU sean de forma periódica, siendo el período de envío de mensajes MQTT configurable para cada grupo Arduino-nodeMCU.

5.1.2 Comunicación I2C

La comunicación I2C, tal y como se expuso en el capítulo 4, sección primera, siempre es originada por el dispositivo que actúa como maestro. Mediante el bit R/W, el maestro indica si está enviando datos al esclavo o quiere recibir datos de este.

Para poder implementar el protocolo I2C en este proyecto, se ha hecho uso de la biblioteca "Wire.h", la cual tiene un par de restricciones a destacar:

1. Una transmisión I2C tiene como límite de envío 32 Bytes.
2. La placa nodeMCU no puede actuar como esclavo de la comunicación.

Con estas restricciones presentes, la placa nodeMCU tiene que ser la que actúe como maestro en la comunicación entre esta y la placa Arduino.

En cuanto a la primera de las restricciones, en el caso dónde la placa nodeMCU haya recibido un mensaje PUBLISH del protocolo MQTT, no existe ningún problema ya que el mensaje a enviar a Arduino siempre será menor de 32Bytes. El problema surge cuando la placa nodeMCU pida los datos de los sensores a la Arduino, pudiendo este tener que enviar más de 32Bytes.

La forma utilizada en este proyecto para solventar dicha limitación consta de dos premisas:

1. Los valores de los sensores almacenados por Arduino tendrán una longitud fija de cuatro dígitos.
2. Existirá una sincronización entre Arduino y nodeMCU, de forma que la placa nodeMCU sepa cuántos sensores están conectado a Arduino; esto es, nodeMCU sabe cuántos datos tiene que recibir de Arduino.

Con estas dos premisas, la placa nodeMCU puede saber cuántos Bytes ha recibido y cuántos quedan por recibir, de forma que establecerá tantas conexiones I2C con límite 32Bytes como sea necesario.

Motivado por la reutilización del código, y para no sobrecargar la memoria de programa de la placa nodeMCU, Arduino almacenará, tanto los datos de los sensores, como los *topics* oportunos, y enviará una cadena con todos estos datos a nodeMCU cada vez que el maestro solicite los datos. De esta forma, en la placa nodeMCU tendremos un código genérico, dónde únicamente se tendrá que separar el *topic* del valor, y enviar el mensaje MQTT con estos dos campos.

Cabe mencionar que el *topic* almacenado en estas placas y enviadas a nodeMCU únicamente hacen referencia a los datos de los pines usados. Esto es, según el formato expuesto en la subsección 5.1.1, Arduino almacenará como *topic* la cadena: "XNNN", con lo que nodeMCU tendrá que añadir la cadena "-P", siendo los valores X, NNN, y P, los mismos de la subsección 5.1.1

Tal y como se ha comentado, Arduino enviará una cadena con los *topics* y los valores de los sensores a nodeMCU cuando sea oportuno. Para facilitar la tarea de separar los campos *topic* y valor, se ha establecido que el campo *topic* comience con el carácter "+", a continuación del *topic*, y antes del valor del sensor, se incluye el carácter "-".

De esta forma, por cada valor de sensor que Arduino tenga que enviar a nodeMCU, enviará una cadena de la forma: "+XNNN-valor", siendo valor, una cadena de 4 caracteres. En una transmisión I2C, al tener un máximo de 32Bytes a enviar, se podrán transmitir, como máximo, 3 cadenas, que aportarán 3 datos de sensores y sus respectivos *topics*.

5.2 Preparación del entorno

En esta sección se expone la configuración sobre la que se basará el sistema domótico desarrollado en este proyecto.

5.2.1 Router Wifi

Uno de los aspectos claves del sistema domótico propuesto en este documento es la comunicación mediante el protocolo MQTT. Esta comunicación sólo es posible si los dispositivos están conectados a un router Wifi que les de acceso a la red asignándoles direcciones IP.

Un router Wifi asigna direcciones IP dinámicas a los dispositivos conectados a la red. Esta configuración supone un problema, ya que, si un dispositivo pierde momentáneamente la conectividad con el router, su dirección IP podría verse modificada.

Para tener una configuración estable, en este proyecto se ha decidido asignar direcciones IP estáticas desde la configuración del router. Además, se ha establecido que la primera dirección IP, 192.168.1.2, sea para la placa Raspberry Pi, y las direcciones consecutivas sean para las placas nodeMCU. En este proyecto las placas nodeMCU tienen las direcciones IP 192.168.1.3 y 192.168.1.4.

Un sistema domótico puede verse ampliado añadiendo todo tipo de dispositivos; por esta razón, en la configuración DHCP del router se ha reservado un rango de direcciones IP que no serán asignadas dinámicamente. De esta forma, estas direcciones IP podrán ser asignadas de forma estática cuando se considere oportuno.

La configuración LAN del router utilizado en este proyecto aparece en la Figura 5.2 siguiente:

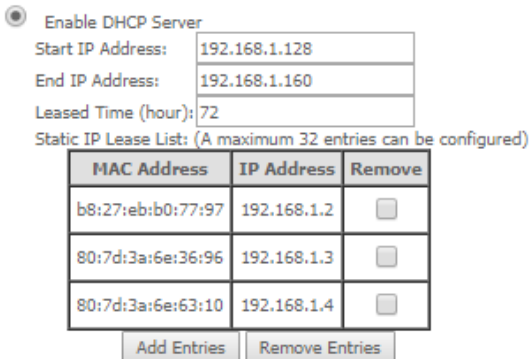


Figura 5.2 - Configuración DHCP

5.2.2 openHABian

openHAB está disponible para múltiples plataformas, sin embargo, no hace uso de muchos de los recursos incluidos por defecto en dichos sistemas operativos. Por esta razón, y debido a la gran popularidad de las plataformas tales como Raspberry Pi, existe una distribución específica para estas plataformas: openHABian

openHABian está basado en el sistema Raspbian Lite, con lo que la carga de procesos innecesarios en la placa se reduce enormemente.

Para poder hacer uso de este software, se deberá descargar la imagen desde el apartado de descargas de la web de openHAB, seleccionando como sistema Raspberry Pi y la versión que deseemos. Para este proyecto se ha usado la versión estable 2.4.0-1 de openHABian.

Una vez descargada, mediante una herramienta de grabación de imágenes ISO, como el programa Etcher, grabamos la imagen en una tarjeta microSD.

Para poder dotar a openHABian de conectividad Wifi, habrá que modificar el fichero `openhabian.conf`. En la línea `wifi_ssid` escribiremos el SSID de nuestra red Wifi, y en la línea `wifi_psk`, la contraseña de la red.

Configurado el acceso Wifi, y tras conectar la tarjeta a la placa Raspberry Pi, openHABian comienza la instalación de forma automática. Tras un intervalo de instalación de aproximadamente 30 minutos, el servicio de openHAB estará disponible con la dirección IP asignada por el router y en el puerto 8080.

Para poder acceder a la configuración de openHABian con el fin de poder administrar nuestro servicio, podremos hacerlo mediante SSH, dónde el nombre usuario y contraseña por defectos es: openhabian

5.2.2.1 Paquetes de administración de openHAB

openHAB ofrece al usuario un abanico de paquetes diferentes para su instalación. La elección de uno u otro depende del nivel de configuración y administración que el usuario quiera tener sobre openHAB.

Para configurar el paquete a instalar nos dirigimos al archivo `/etc/openhab2/services/addons.cfg`. Al editar este archivo, debemos descomentar la línea comenzada por la palabra "`package`" y asignarle el valor deseado. En este proyecto se ha trabajado con el paquete "*standard*". Las otras opciones posibles son: *minimal*, *simple*, *expert* y *demo*.

En este mismo fichero, se configurará los *bindings* que queramos instalar mediante la línea "`bindings`", dónde, al escribir el nombre del *binding* y guardar el fichero, openHABian descargará los *binding* para poder ser usados. En este trabajo se han utilizado los *binding mqtt1*.

El *binding* mqtt1 hace posible utilizar el protocolo MQTT en openHAB, y el *binding network* permite descubrir dispositivos conectados a la red mediante el uso de *ping's*

5.2.2.2 Servidor MQTT: Mosquitto

En este proyecto se utiliza el programa Mosquitto como servidor MQTT. Para instalar este servidor debemos ejecutar el comando: sudo apt-get install mosquitto

La configuración inicial dada es muy básica, pero podemos encontrar un ejemplo de configuración más

complejo en el directorio `/usr/share/doc/mosquitto/examples`.

En este directorio se encuentra el archivo mosquitto.conf.gz. Para descomprimirlo usamos el comando `sudo gunzip mosquitto.conf.gz`, y para copiarlo al directorio de la configuración de mosquitto, desde el mismo directorio (`/usr/share/doc/mosquitto/examples`) ejecutamos el comando `cp mosquitto.conf /etc/mosquitto/conf.d/`, nos dirigimos al directorio destino y lo editamos con `sudo nano mosquitto.conf`.

En este fichero, descomentamos las líneas del puerto de escucha (1883), y del protocolo mqtt.

Con esta configuración tendríamos un servidor MQTT llamado mosquitto, atendiendo conexiones por el puerto 1883.

5.2.2.3 Binding MQTT

En el apartado anterior se instaló el servidor de MQTT. Aunque el servidor esté instalado en openHABian, openHAB no está haciendo uso aún de este servicio.

Para que openHAB se conecte al servidor Mosquitto, debemos editar el fichero de configuración del *binding* MQTT. Este fichero se encuentra en el directorio `/etc/openhab2/services/mqtt.cfg`.

En este fichero podemos establecer los parámetros explicados en la sección 4.2 MQTT, tales como la calidad de servicio o QoS, usuario y contraseña de la sesión para la seguridad, …

Para establecer una sesión con el servidor MQTT tendremos que indicar la URL del servidor. Esta URL se indica mediante: NNN.url=tcp://DDD:PPP, dónde NNN, DDD y PPP son el nombre, dirección IP y puerto de escucha del servidor MQTT respectivamente.

Con la configuración del servidor MQTT realizada anteriormente, la URL del servidor tiene la forma: `mosquitto.url=tcp://192.168.1.2:1883`

5.2.3 Arduino IDE

5.2.3.1 nodeMCU

Las placas nodeMCU son compatibles con Arduino IDE; sin embargo, estas placas no vienen configuradas por defecto en el software.

El primer paso para poder utilizar las placas nodemCU es entrar en el administrador de tarjetas de Arduino IDE e incluir el paquete del chip ESP8266. Para ello, abrimos Arduino IDE y navegamos por los menús Archivo→Preferencias. En el campo "Gestor de URLs Adicionales de Tarjetas" escribimos la dirección: http://arduino.esp8266.com/stable/package_esp8266com_index.json. Tras pulsar sobre "OK", tendremos disponible el chip esp8266 en el administrados de placas.

Una vez realizado el paso anterior, podremos instalar la placa nodeMCU. Para esto, deberemos llegar al Gestor de tarjetas de Arduino mediante los menús Herramientas→Placa→Gestor de tarjetas. En esta sección deberemos escribir en el cuadro de búsqueda "ESP8266" con el fin de encontrar las librerías de las placas que integran este chip. En este proyecto se ha instalado la biblioteca proporcionada por "ESP8266 Community" (Figura 5.3).

Figura 5.3 - Gestor tarjetas Arduino IDE

Tras realizar estos pasos, ya podremos seleccionar la placa nodeMCU para subir nuestro código mediante los menús Herramientas→Placa→NodeMCU v1.0, pero no podremos hacer uso del protocolo MQTT.

5.2.3.2 Librería PubSubClient

Para poder conectarnos al servidor MQTT se ha instalado la librería PubSubClient, mediante: Programa→Incluir librería→Administrar Bibliotecas, y escribiendo en el cuadro de búsqueda "PubSubClient". En este proyecto se ha utilizado la librería creada por Nick O'Learly en su versión 2.7.0 (Figura 5.4).

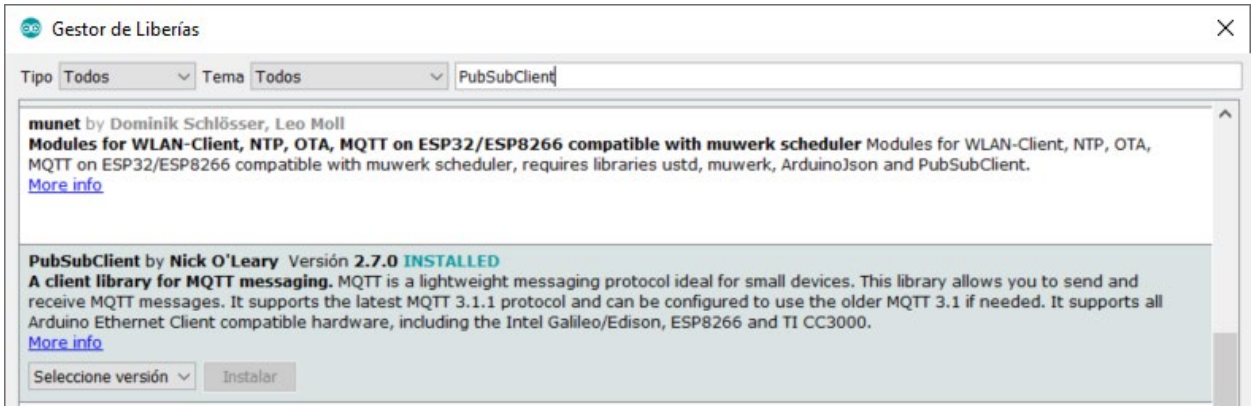


Figura 5.4 - Librería PubSubClient

5.2.3.3 Librería I2C

De la misma forma que se ha instalado la librería para hacer uso del protocolo MQTT, también es necesario hacer uso de una librería para poder establecer comunicación con el protocolo I2C entre la placa Arduino y nodeMCU. La librería usada para la comunicación I2C es "Wire.h" aunque, en este caso, la librería viene instalada en Arduino IDE por defecto.

5.2.3.4 Librería sensor temperatura - humedad

Para poder establecer comunicación con el sensor DHT22 y obtener los datos que aporta el sensor, en este proyecto se ha hecho uso de la librería desarrollada por Adafruit en su versión 1.3.4 (Figura 5.5)

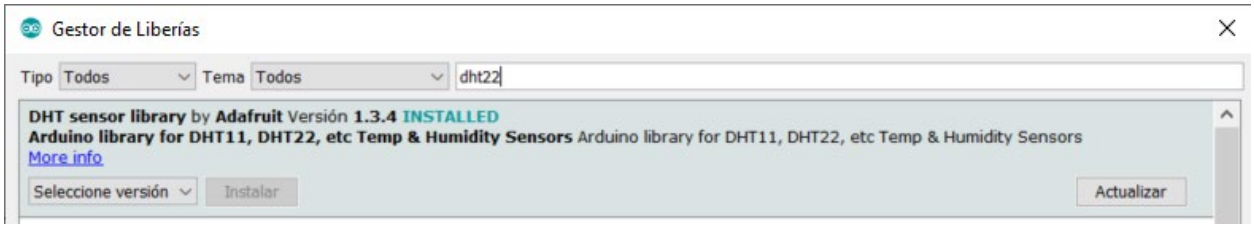


Figura 5.5 - Librería DHT sensor

5.3 Conexiones y funcionamiento grupos Arduino - nodeMCU

Anteriormente se ha comentado que este proyecto se estructura en grupos de placas Arduino - nodeMCU. En esta sección se explicará en detalle el código utilizado en cada placa y se ilustrarán las conexiones realizadas en las placas Arduino y nodeMCU para los dos grupos utilizados en el sistema domótico expuesto en este documento.

5.3.1 Conexiones

5.3.1.1 Grupo 1: Arduino MEGA - nodeMCU

Dada la multitud de pines disponibles en la placa MEGA de Arduino, esta placa será la que tenga mayor número de conexiones en este proyecto.

La placa Arduino MEGA usará los pines digitales número 20 (D20) y 21 (D21) para la comunicación I2C con la placa nodeMCU dónde la función de cada pin en una comunicación I2C viene detallada en el apartado 3.1.2.3

Los sensores DHT22 y MQ-7 son conectados a la placa MEGA mediante el pin digital número 9 (D9) y el pin analógico 0 (A0).

Adicionalmente, la placa Arduino MEGA tendrá conectado 7 leds que simularán 7 luces de la vivienda. Estas conexiones hacen uso de los pines digitales 2 (D2), 3 (D3), 4 (D4), 5 (D5), 6 (D6), 7 (D7) y 8 (D8)

En la placa nodeMCU, únicamente serán utilizados los pines oportunos para la comunicación I2C. Estos son, los pines D1 y D2.

La configuración realizada para este primer grupo queda ilustrada en la Figura 5.6.

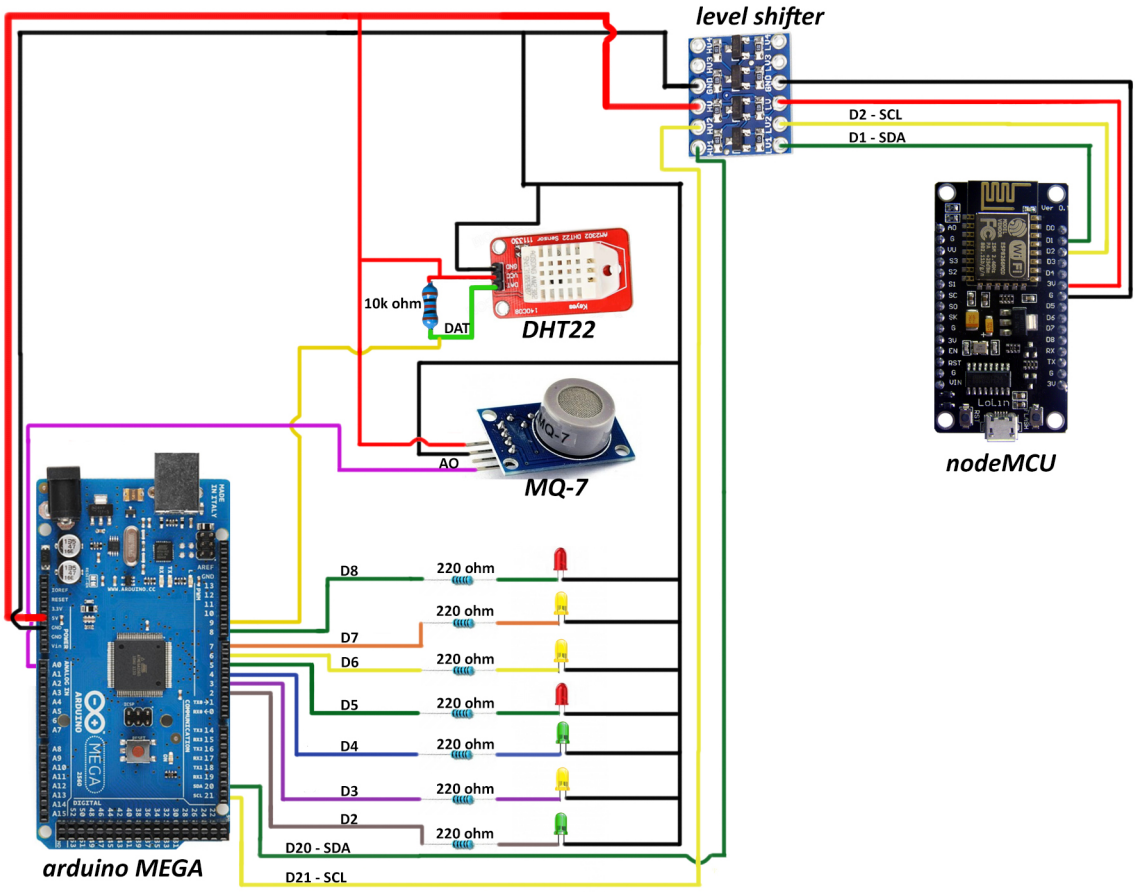


Figura 5.6 - Conexiones Grupo 1: Arduino MEGA - nodeMCU

5.3.1.2 Grupo 2: Arduino UNO - nodeMCU

En la configuración realizada para el segundo grupo, Figura 5.7, puede verse cómo se usan los pines de la placa Arduino UNO y nodeMCU.

De este grupo cabe destacar la diferencia de conexión del sensor MC38 que, debido a que es un sensor crítico, está conectado directamente a la placa nodeMCU. Debido a que el sensor trabaja con tensiones 5V, el conexionado con la placa nodeMCU se realiza mediante el conversor de nivel.

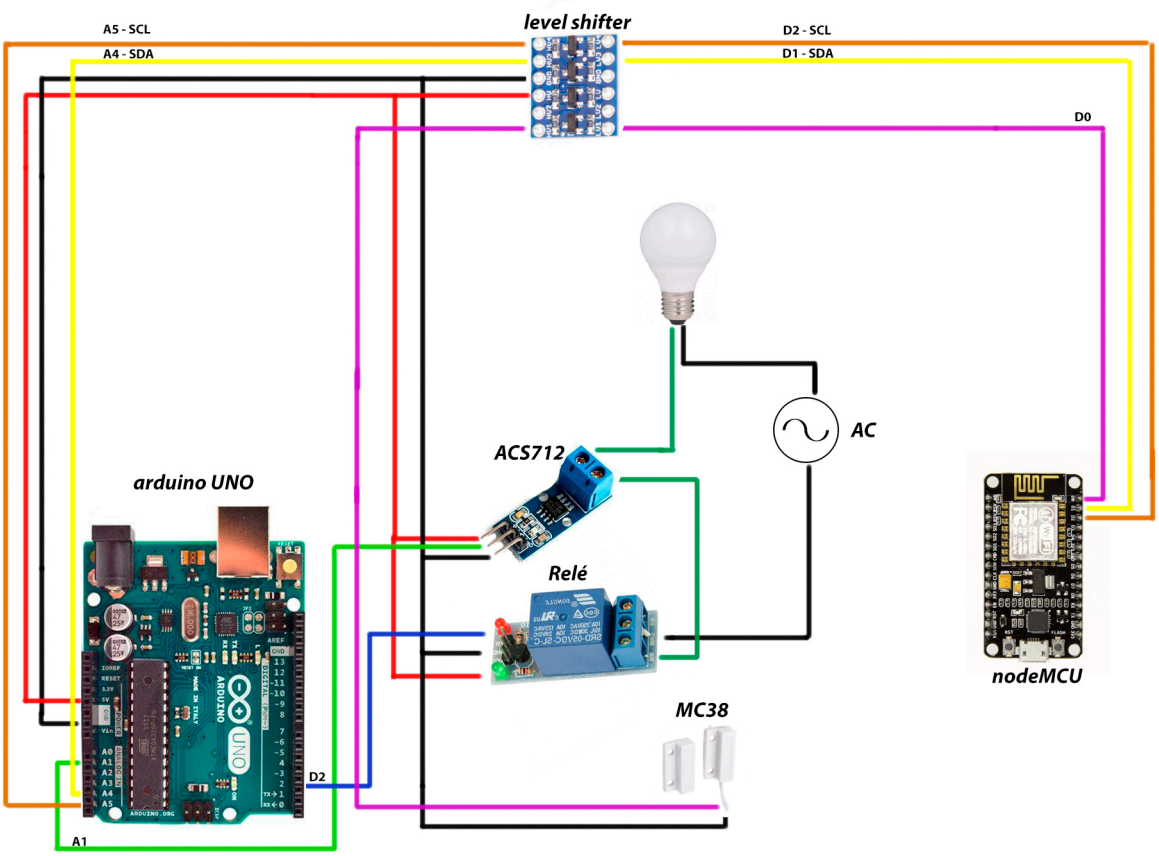


Figura 5.7 - Conexiones Grupo 2: Arduino UNO - nodeMCU

5.3.2 Funcionamiento

En esta subsección se expondrá el razonamiento seguido en la elaboración del código usado en las placas, así como las variables y funciones existentes.

Debido a la reutilización del código en los distintos grupos Arduino - nodeMCU, se explicará en detalle el funcionamiento del código utilizado como base en todas las placas, ya que la única diferencia entre el código de una placa y otra será únicamente el código para la obtención de los datos de los sensores, los *topics* utilizados y las direcciones IP en el caso de las placas nodeMCU. Para la explicación del código se utilizará como ejemplo el grupo1: Arduino MEGA - nodeMCU, cuyo código está recogido en el Anexo A.

5.3.2.1 nodeMCU

A modo de recordatorio, la placa nodeMCU será la encargada de establecer conexión con el servidor MQTT, enviar y recibir mensajes de los *topics* pertinentes para su grupo, solicitar los datos de los sensores de Arduino y, si se diera el caso, informar de los sensores críticos conectados directamente a la placa nodeMCU.

5.3.2.1.1 Librerías utilizadas

Toda placa nodeMCU utilizada en este proyecto debe incorporar tres librerías para poder realizar sus funciones básicas.

1. Wire.h: esta librería permitirá utilizar el protocolo I2C entre la placa Arduino y nodeMCU
2. ESP8266Wifi.h: librería propia del chip ESP8266 que permite la conexión Wifi
3. PubSubClient.h: se utilizará esta librería para implementar el protocolo MQTT

5.3.2.1.2 Variables globales

const char * ssid

Será utilizada por la librería ESP8266Wifi.h para la conexión Wifi. Su valor es el SSID de la conexión Wifi.

const char * password

Utilizada por la librería ESP8266Wifi.h, recoge la contraseña de la red Wifi.

const char* mqtt_server

Usada por una variable de la librería PubSubClient.h, esta variable se usará para almacenar la dirección IP del servidor MQTT.

WiFiClient clientWifi

Crea un cliente que puede conectarse a una dirección IP y puerto especificado por parámetros. Será usado para conectarse como cliente al servidor MQTT.

PubSubClient clientMQTT

Hará uso de la variable clientWifi para conectarse al servidor MQTT.

const int interval

Variable que guardará el valor, en milisegundos, del período de solicitud de datos a Arduino y envío al servidor MQTT.

unsigned long millisPrevio

Variable auxiliar para contar el tiempo especificado por la variable interval.

int valoresIN

Variable necesaria para el sincronismo con la placa Arduino. Recoge el número de datos de sensores que la placa nodeMCU tiene que recibir de Arduino.

5.3.2.1.3 Funciones

void setup()

El código de esta función se ejecutará una única vez y será el primero que se ejecute.

En esta función se establecen las comunicaciones I2C y MQTT.

void topicsSuscritos()

El propósito de esta función es suscribirse a los *topics* específicos de cada placa nodeMCU.

void setup_wifi()

Inicia la conexión Wifi con el router.

void reconnectMQTT()

Función usada para reconectar con el servidor MQTT.

void callback(char* topic, byte* payload, unsigned int length)

Esta función será llamada cuando se reciba un mensaje MQTT de tipo PUBLISH. Identifica el *topic* y el *payload*. Envía los datos a la placa Arduino.

void loop()

Función bucle infinito. Esta función se ejecutará de forma ininterrumpida hasta que se elimine la alimentación de la placa nodeMCU, provocando un reinicio de la misma.

Dentro de esta función se comprueba la conectividad con el router Wifi y el servidor MQTT, se solicitan datos a la placa Arduino de forma periódica y se envían al servidor MQTT.

5.3.2.1.4 Explicación del código

En los subapartados anteriores se han expuesto las librerías, variables globales y funciones en los que se basa el código de todas las placas nodeMCU utilizadas en este proyecto. En este apartado se explica en detalle el código de la placa nodeMCU perteneciente al grupo 1: Arduino MEGA - nodeMCU (Figura 5.8).

```
1  #include <Wire.h>
2  #include <ESP8266WiFi.h>
3  #include <PubSubClient.h>
4
5  /* Variables para la conexion Wifi */
6  const char* ssid = "JAZZTEL_5DB3";
7  const char* password = "96828h8C47868CaF882F";
8  const char* mqtt_server = "192.168.1.2";
9  WiFiClient clientWifi;
10 PubSubClient clientMQTT(clientWifi);
11
12 /* Variables para el intervalo de peticion de datos y envio a openHAB*/
13 const int interval = 7000;
14 unsigned long millisPrevio=0;
15 int valoresIN = 4; //Necesario indicar cuantos bytes tiene que recibir
```

Figura 5.8 - Librerías y variables globales nodeMCU grupo 1

En primer lugar, deben indicarse las librerías de las que se hará uso en el programa y definirse las variables globales que podrán ser usadas y serán comunes en todas las funciones.

Las librerías a importar son las ya mencionadas: Wire.h, ESP8266.h y PubSubClient.h.

Las variables globales ssid, password, mqtt_server clientWifi, clientMQTT y millisPrevio deben inicializarse de la misma forma en todas las placas nodeMCU.

La variable interval puede establecerse de forma diferente entre una placa nodeMCU y otra. El valor de la variable valoresIN dependerá de los datos que la placa Arduino aporte a nodeMCU.

En este proyecto, el *ssid* y contraseña de la red Wifi son los indicados en la Figura 5.8. El valor de mqtt_server se toma según lo configurado en el servidor DHCP del router Wifi; el cual, tiene especificado otorgar la dirección IP 192.168.1.2 a la placa Raspberry Pi.

Según se observa en la Figura 5.8 - Librerías y variables globales nodeMCU grupo 1, en las líneas 9 y 10 se crea un cliente MQTT y se establece el período para recibir los cuatro datos de Arduino en siete segundos.

```
18 void setup() {
19   Wire.begin(D1,D2); //sda, scl. Comunicacion i2c
20
21   setup_wifi();
22   clientMQTT.setServer(mqtt_server, 1883);
23   clientMQTT.setCallback(callback);
24   reconnectMQTT();
25 }
```

Figura 5.9 - Función setup placa nodeMCU grupo 1

En la función setup, ilustrada en la Figura 5.9, se establecen los parámetros para comunicación I2C con Arduino. El pin D1 se utilizará como línea SDA y el pin D2 como línea SCL del protocolo.

En la línea 22 y 23 se establecen los parámetros para la comunicación MQTT. En la línea 22 se especifican la dirección IP del servidor MQTT y el puerto de escucha, mientras que la línea 23 indica la función a ejecutar cuando se reciba un mensaje de tipo PUBLISH.

Por último, mediante las líneas 21 y 24 (Figura 5.9), la función setup_wifi (Figura 5.10) establece conexión Wifi y MQTT respectivamente.

```
92 void setup_wifi() {
93   // Nos conectamos a la red Wifi
94   WiFi.begin(ssid, password);
95   while (WiFi.status() != WL_CONNECTED) {
96     delay(500);
97   }
98 }
99
```

Figura 5.10 - Función setup_wifi nodeMCU grupo 1

Como ya se ha indicado, la función setup_wifi establece la conexión Wifi con el router, usándose las variables ssid y password para ello. La obtención de datos de los sensores no tiene sentido si no pueden ser enviados al servidor MQTT. Esto no es posible si no existe conectividad Wifi; por esta razón, la función setup_wifi integra un bucle infinito hasta que se consigue conectividad con el router.

```
100 void reconnectMQTT() {
101   // Bucle hasta que reconecta
102   while (!clientMQTT.connected()) {
103     if (clientMQTT.connect("placa1")) {
104       topicsSuscritos(); // Una vez conectado, volvemos a suscribirnos
105     }
106     else {
107       // Esperamos 5 segundos para volver a conectar
108       delay(5000);
109     }
110   }
111 }
112
```

Figura 5.11 - Función reconnectMQTT nodeMCU grupo 1

El propósito de la función reconnectMQTT (Figura 5.11) es conectarse como cliente al servidor MQTT. Al igual que ocurre con la función setup_wifi, se ha incluido como condición para salir de esta función, haber abierto sesión con el servidor MQTT. Una vez se haya podido conectar con el servidor MQTT, se llama a la función topicsSuscritos.

De esta función cabe destacar la línea 103. En esta línea se intenta conectar al servidor con un nombre de cliente concreto, en este caso, “placa1”. El nombre del cliente debe ser único, no puede haber dos clientes conectados con el mismo nombre, por lo que este es otro aspecto que varía de una placa nodeMCU a otra.

```
81
82 void topicsSuscritos(){//El -1 del final es el codigo de la placa. Cada una tendrá un codigo diferente.
83   clientMQTT.subscribe("D080-1"); //luz dormitorio principal
84   clientMQTT.subscribe("D070-1"); //luz dormitorio secundario
85   clientMQTT.subscribe("D060-1"); //luz salon
86   clientMQTT.subscribe("D050-1"); //luz cocina
87   clientMQTT.subscribe("D040-1"); //luz pasillo 1
88   clientMQTT.subscribe("D030-1"); //luz pasillo 2
89   clientMQTT.subscribe("D020-1"); //luz pasillo 3
90 }
91
```

Figura 5.12 - Función topicsSuscritos nodeMCU grupo 1

La función topicsSuscritos (Figura 5.12) tiene como finalidad suscribirse a los *topics* oportunos. La placa nodeMCU recibirá mensajes PUBLISH de dichos *topics* y deberá trasladar el mensaje a Arduino.

```
112
113 void callback(char* topic, byte* payload, unsigned int length) {
114   if (topic[0]=='D') {//actuamos sobre pines digitales: D/pinout-valor
115     String valor="";
116     valor +=topic;
117     valor+='-';
118     valor+=(char)payload[0];//Ej: D/1-1 (Poner en HIGH el pin digital numero 1
119     Wire.beginTransmission(2);  // inicia una transmicion al dispositivo #2
120     Wire.write(valor.c_str()); //Mandamos valor
121     Wire.endTransmission(); //Terminamos transmision
122   }
123   else {//pin analogico. No tenemos ninguna salida analogica
124   }
125 }
```

Figura 5.13 - Función callback nodeMCU grupo 1

Cuando la placa nodeMCU reciba mensajes PUBLISH del protocolo MQTT, la función que se ejecutará para procesar dichos mensajes, según lo configurado en la línea 23 de la función setup, es la función callback (Figura 5.13).

La biblioteca PubSubClient se encarga de diferenciar el *topic* y el *payload* del paquete MQTT, los cuales son pasados como parámetros a esta función para que podamos interpretarlos de la forma deseada.

En la función callback se pasa a la placa Arduino una cadena con el *topic* y el nuevo valor para que este lo interprete. La cadena pasada a Arduino tiene la forma: *topic*-valor.

Las líneas 119, 120 y 121 tienen como finalidad comenzar una transmisión con el esclavo con dirección pasada por parámetros, transmitir datos y terminar la transmisión respectivamente.

```
27 void loop() {
28   /* Comprobacion de conexion a la red */
29   if (WiFi.status() != WL_CONNECTED)
30     setup_wifi();
31
32   if (!clientMQTT.connected()) {
33     reconnectMQTT();
34   }
35   clientMQTT.loop();
36
```

Figura 5.14 - Función loop nodeMCU grupo 1. Fragmento 1

Por último, tenemos la función loop (Figura 5.14), la cual se ejecutará de forma continuada hasta que se retire la alimentación de la placa. Esta función es más extensa que las demás, por lo que se detallará por fragmentos.

En el primer fragmento del código, ilustrado por la Figura 5.14, se comprueba, en primer lugar, la conectividad Wifi con la línea 29. Si resulta que la placa se ha desconectado de la red, volveremos a conectarnos con la llamada a la función setup_wifi de la línea 30. A continuación, lo que se comprueba es la conexión con el servidor MQTT dónde, si se ha perdido la conexión, se intenta reconectar llamando a la función reconnectMQTT.

La última orden de este primer fragmento de código es la línea 35. Esta orden se corresponde con el cliente MQTT y es necesario ejecutarla periódicamente. Su función es informar al servidor MQTT que el cliente sigue estando conectado.

```
37  /*Peticion de datos si ha pasado el tiempo indicado */
38  if((unsigned long)(millis() - millisPrevio) >= interval){
39   millisPrevio=millis();
40
```

Figura 5.15 - Función loop nodeMCU grupo 1. Fragmento 2

Una vez realizada las comprobaciones de conectividad, se comprueba si se ha cumplido el tiempo de petición de datos a Arduino especificado por la variable interval (Figura 5.15). Esta comprobación se realiza con la función millis(), la cual devuelve una variable de tipo unsigned long, cuyo valor se corresponde con el número de mili segundos transcurridos desde que se comenzó a ejecutar el programa.

La forma de contar el período, con el valor devuelto por la función millis, es mediante la diferencia del instante actual y el último instante en que se solicitaron datos a Arduino. Si se cumple la condición del intervalo, se actualiza la variable millisPrevio, que almacena el instante en el cual se solicitaron datos a Arduino.

```
41  /*i2c maximo tx 32Bytes */
42  int cont_rx =0;  //Cuenta las entradas recibidas totales (cada entrada ocupa 10B)
43  int entradas_pide=0; //Numero de entradas (cada una ocupa 10Bytes) que se van a pedir en la transmision
44
```

Figura 5.16 - Función loop nodeMCU grupo 1. Fragmento 3

Las líneas 42 y 43 (Figura 5.16) tienen como propósito preparar la solicitud de datos a Arduino. Se recuerda que la librería utilizada para la comunicación I2C entre Arduino y nodeMCU, Wire.h, permite un máximo de 32 Bytes en una transmisión; por lo que, si los datos a transmitir exceden de dicha solicitud, habrá que establecer más de una solicitud. Se recuerda también que, por cada dato de un sensor, se transmite una cadena de 10 Bytes. Esta cadena está explicada en la subsección 5.1.2.

La forma de solventar la restricción anterior es saber cuántas entradas hay que pedir a Arduino, lo cual viene dado por la variable valoresIN, y cuántas entradas se han pedido hasta el momento. La variable cont_rx almacena el número de entradas pedidas totales, mientras que la variable entradas_pide guarda el número de entradas que se van a pedir en dicha transmisión. El valor de la variable entradas_pide podrá tener, por lo tanto, un valor entre cero y tres, ya que en una transmisión no se pueden solicitar más de 3 entradas (3 entradas = 30 Bytes).

```
45  while(cont_rx!=valoresIN){ //Comprobamos si ya hemos pedido todas las entradas
46     if((valoresIN-cont_rx)>=3){ //Si aun nos quedan por pedir mas de 30B
47       cont_rx += 3; //Sumamos los 30B que vamos a pedir al contador total de Bytes recibidos
48       entradas_pide = 3; //Indicamos que vamos a pedir 30Bytes
49     }
50     else{
51       entradas_pide = valoresIN-cont_rx; //Indicamos que vamos a pedir las entradas que faltan.
52       cont_rx += valoresIN-cont_rx; //Sumamos las entradas que faltan al contador de entradas recibidas
53     }
54    Wire.requestFrom(2, entradas_pide*10); //Pedimos al esclavo numero dos que nos mande los Bytes indicados
```

Figura 5.17 - Función loop nodeMCU grupo 1. Fragmento 4

El fragmento de código comprendido entre las líneas 45 y 53 corresponden a averiguar el valor de entradas_pide para pedir datos a Arduino (Figura 5.17).

Se estarán demandando datos a Arduino hasta que se hayan pedido todas las entradas. Esta condición se

corresponde con la línea 45.

Una vez obtenido el número de datos de sensores, la línea 54 se encarga de solicitar el número de Bytes correspondientes. Se recuerda que cada dato que aporta Arduino se corresponde con una cadena de 10 Bytes.

Una vez solicitado el número de Bytes correspondientes, se procede a recibirlos. Los datos se reciben Byte a Byte, y son concatenados a la cadena de nombre datos (Figura 5.18)

```
55      String datos = "";
56      while(Wire.available()){ //Recibimos los bytes
57         datos += (char)Wire.read(); //Leemos bytes. La cadena recibida sera una concatenacion
58                                     //de la cadena: +XXXX-YYYY, donde XXXX es el topic e YYYY es el valor del sensor
59      }
```

Figura 5.18 - Función loop nodeMCU grupo 1. Fragmento 5

La Figura 5.19 ilustra el último fragmento de código de la función loop. La finalidad de este código es obtener todos los topics y valores correspondientes de los datos recibidos de Arduino. Por cada topic y valor recibido se envía un mensaje PUBLISH mediante la línea 76.

```
60      int separadorCadena = datos.indexOf('+'); //
61
62      while(separadorCadena>=0){ //Mientras que nos falte por leer mas signos "+"
63       int separadorTopic = datos.indexOf('-', separadorCadena); //Buscamos el siguiente signo "-"
64       String topic = datos.substring(separadorCadena+1, separadorTopic); //Primero es inclusive, segundo no.
65                                                                          //Nos saltamos el primer elemento (D o A) hasta el signo "-" - 1.
66       topic+="-1"; //Añadimos al topic el codigo de la placa que envia los datos.
67       separadorCadena = datos.indexOf('+', separadorTopic); //Buscamos el siguiente signo "+" para tomarlo como indice referencia
68       String valor = "";
69       if(separadorCadena>=0){ //Si queda alguna cadena mas por leer...
70         valor = datos.substring(separadorTopic+1, separadorCadena); //Leemos a partir del segundo elemento (nos saltamos el primero (D o A)
71                                                                     //hasta el separador de cadena -1
72       }
73       else{ //Si no quedan mas cadenas por leer...
74         valor = datos.substring(separadorTopic+1); //Leemos hasta el final.
75       }
76       clientMQTT.publish(topic.c_str(), valor.c_str()); //Publicamos directamente.
77      }
78     }
79  }
80 }
```

Figura 5.19 - Función loop nodeMCU grupo 1. Fragmento 6

5.3.2.2 Arduino

5.3.2.2.1 Librerías

Las librerías usadas en la placa Arduino del grupo 1 son:

1. Wire.h: esta librería permitirá utilizar el protocolo I2C entre la placa Arduino y nodeMCU. Es necesario que toda placa Arduino importe esta librería
2. DHT.h: librería para poder recopilar datos del sensor DHT22. Esta librería se incorpora porque este sensor está conectado a la placa.

5.3.2.2.2 Variables globales

const int interval

Variable que guardará el valor, en milisegundos, del período de recopilación de datos de los sensores.

unsigned long millisPrevio

Variable auxiliar para contar el tiempo especificado por la variable interval.

int cont_tx

El objetivo de esta variable es almacenar el número de entradas enviadas a nodeMCU en una transmisión I2C.

boolean tablaD[][2]

Tabla multidimensional de dos columnas. El propósito de esta variable es almacenar los datos correspondientes a los pines digitales. La primera columna indica si el pin está a nivel alto o bajo, mientras que la segunda indica si el pin debe configurarse a como entrada o como salida.

int asociacionD[]

Esta variable asocia los datos de cada fila de la variable tablaD con un pin de la placa Arduino.

String valoresIN_D[]

Tabla de cadena de caracteres. Estas cadenas se enviarán a la placa nodeMCU cuando solicite los datos de los sensores. La estructura de estas cadenas es: *topic*-valor.

int asociacionA[]

Tabla que recoge los pines analógicos que se van a usar en la placa Arduino.

String valoresIN_A[]

Tabla de cadenas *topic*-valor de los sensores analógicos.

DHT dht

Esta variable es la única que no es común a todas las placas Arduino. Se usará para obtener los valores del sensor DHT22.

5.3.2.2.3 Funciones

void setup()

El código de esta función se ejecutará una única vez, y será el primero que se ejecute.

En esta función se establecen las configuraciones del protocolo I2C y la configuración de los pines de la placa. Adicionalmente, en el caso de la placa Arduino MEGA, se inicia el sensor DHT22.

void configPinout()

Esta función hace uso de las variables tablaD y asociacionD para configurar los pines digitales de Arduino.

void cambiarValor(size_t howmany)

Función a ejecutar cuando la placa nodeMCU envíe datos al esclavo.

void solicitanDatos()

El código de esta función se ejecutará cuando la placa nodeMCU solicite datos del esclavo.

void loop()

Función bucle infinito. Esta función se ejecutará de forma ininterrumpida hasta que se elimine la alimentación de la placa Arduino, provocando un reinicio de la misma.

Dentro de esta función se obtendrán los datos de los sensores de forma periódica.

5.3.2.2.4 Explicación código

En este subapartado se entrará en detalle en el código de las placas Arduino de este proyecto, particularizando para el código de la placa Arduino MEGA correspondiente al grupo 1.

En este código inicial (Figura 5.20), tal y como se hizo con la placa nodeMCU, se importan las librerías que se usarán en el programa. Además, se definen e inician las variables globales.

De las variables globales explicadas en el subapartado 5.3.2.2.2, se destaca la variable tablaD y asociaciónD. En la Figura 5.20 puede verse la definición de estas dos variables. Mientras que la variable asociacionD recoge el pin a usar, la variableD recoge, en filas de parejas, las propiedades de cada entrada de la tabla asociacionD.

La entrada cero de la variable asociacionD tiene las propiedades de la fila 0 de la tablaD. Esto es, el pin 2 (asociacionD[0]) deberá iniciarse a nivel bajo (tablaD[0][0]) y será un pin de salida (tablaD[0][1]), mientras que el pin 9 (asociacionD[7]), deberá iniciarse a nivel bajo (tablaD[7][0]) y será un pin de entrada (tablaD[7][1]). Al ser un pin de entrada, el campo tablaD[7][0] no se verá reflejado.

Se ha decidido separar la asociación de los pines y las propiedades para reducir el tamaño en memoria, ya que una variable int ocupa 2 Bytes, mientras que una variable boolean ocupa sólo 1 Byte.

```
#include <Wire.h>
#include "DHT.h"

const int interval = 5000;
unsigned long millisPrevio=0;

int cont_tx=0;

boolean tablaD[][2]= //Tabla pines digitales. primero si HIGH=true, segundo si output=true
{
  {false, true}, //Luz pasillo 3 = D2.
  {false, true}, //Luz pasillo 2 = D3.
  {false, true}, //Luz pasillo 1 = D4.
  {false, true}, //Luz cocina = D5.
  {false, true}, //Luz salon = D6.
  {false, true}, //Luz dormitorio secundario = D7.
  {false, true}, //Luz dormitorio principal = D8.
  {false, false} //Sensor de temperatura DHT22 (3 datos)= D9
};

int asociacionD[]={2, 3, 4, 5, 6, 7, 8, 9};
```

Figura 5.20 - Librerías y variables globales Arduino MEGA. Fragmento 1

Como se comentó en el subapartado Variables globales, la variable valoresIN_D (Figura 5.21) recoge las cadenas a pasar a nodeMCU correspondiente a los pines digitales. A estas cadenas, que siguen el formato *topic*-valor, se les asigna un valor inicial de forma que, si existiera algún fallo con un sensor, se vería reflejado al tener un valor extremo.

```

String valoresIN_D[] =
{
  "D091-9999", //Temperatura. Pin D9, dato 1
  "D092-9999", //Humedad. Pin D9, dato 2
  "D093-9999" //Sensacion termica. Pin D9, dato 3
};

```

Figura 5.21 - Librerías y variables globales Arduino MEGA. Fragmento 2

Los pines analógicos tienen la ventaja de que pueden funcionar en cualquier momento tanto de entrada como de salida, por lo que no necesitan ser definidos. Únicamente se guarda en la variable asociacionA los pines analógicos que se van a usar, y en la variable correspondiente, los valores de los sensores analógicos (Figura 5.22).

```
int asociacionA[]={0}; //Si usamos sólo analog pins con analogRead, podemos poner 0,1... en vez de A0...
String valoresIN_A[]=
{
  "A030-9999" //Monoxido
};

```

Figura 5.22 - Librerías y variables globales Arduino MEGA. Fragmento 3

En última instancia, y de forma particular para la placa MEGA de Arduino, se define la variable dht necesaria para la obtención de datos del sensor DHT22 (Figura 5.23). Mediante parámetros se especifica tanto el pin asociado al sensor, como el modelo. Los dos modelos posibles son: DHT11 y DHT22.

```
36 DHT dht(asociacionD[7], DHT22); //DHTPIN, DHTTYPE
37
```

Figura 5.23 - Librerías y variables globales Arduino MEGA. Fragmento 4

A continuación (Figura 5.24), nos encontramos con la función setup. En esta función se define la dirección de esclavo en la comunicación I2C con la línea 39. En este proyecto todos los esclavos tienen la dirección número 2.

En la comunicación I2C el esclavo tiene que definir sus acciones, tanto para el caso en el que el maestro envíe datos, como en el caso de que solicite datos. El comportamiento en estos casos estará determinado por las funciones cambiarValor (línea 41) y solicitanDatos (línea 42) respectivamente.

Las dos órdenes restantes corresponden a la inicialización del sensor DHT22 (línea 40) y la configuración de los pines que se usarán durante el programa (línea 43).

```
38 void setup() {
39   Wire.begin(2);
40   dht.begin();
41   Wire.onReceive(cambiarValor); //Registramos la funcion a ejecutar cuando recivamos un evento
42   Wire.onRequest(solicitanDatos); //Funcion a ejecutar cuando el maestro nos solicite datos
43   configPinout();
44 }
45
```

Figura 5.24 - Función setup placa Arduino MEGA grupo 1

La función encargada de configurar los pines es configPinout (Figura 5.25). En esta función se recorren las tablas: tablaD y asociacionD, configurando para cada pin declarado en la tabla asociacionD, las propiedades correspondientes de tablaD.

```
137 void configPinout(){
138   int filasD = sizeof(tablaD)/sizeof(tablaD[0]); //Obtenemos el numero de filas de la tabla de pines Digitales
139
140   for(int i=0; i<filasD; i++){
141     if(tablaD[i][1]==true){
142       pinMode(asociacionD[i], OUTPUT);
143       if(tablaD[i][0]==true)
144         digitalWrite(asociacionD[i], HIGH);
145       else
146         digitalWrite(asociacionD[i], LOW);
147     }
148     else{
149       pinMode(asociacionD[i], INPUT);
150     }
151   }
152 }
```

Figura 5.25 - Función configPinout placa Arduino MEGA grupo 1

En el caso que la placa nodeMCU envíe datos al esclavo Arduino MEGA, se ejecutará la función cambiarValor (Figura 5.26).

En esta función se recibe la cadena de Bytes enviada por el maestro, y se separa en pin y valor. En este proyecto no se tiene ningún pin de salida analógico, por lo que el único caso contemplado de la condición switch es el caso de pin digital.

```
82 void cambiarValor(size_t howmany){
83   String cadena = "";
84   while(Wire.available()){
85     cadena += (char)Wire.read();
86   }
87   int separador = cadena.indexOf('-');
88   String pin = "";
89   if(cadena[1]=='0') //.toInt() no es capaz de traducir 04 a 4
90      pin = cadena.substring(2, separador-1); //Empezamos despues del caracter D hasta el caracter -
91   else
92     pin = cadena.substring(1, separador-1); //Empezamos despues del caracter D hasta el caracter -
93
94   separador = cadena.indexOf('-', separador); //Porque la cadena ahora es del tipo: XXXX-A-Y. Donde A es el codigo de la placa
95
96   String valor = cadena.substring(separador+3); //Empezamos desde el caracter siguiente a -
97
98   int aux=pin.toInt();
99   switch (cadena.charAt(0)){
100    case 'D':
101      switch (valor.toInt()){
102        case 1:
103          digitalWrite(pin.toInt(), HIGH);
104          break;
105        case 0:
106          digitalWrite(pin.toInt(), LOW);
107          break;
108      }
109      break;
110  }
111 }
112
```

Figura 5.26 - Función cambiarValor placa Arduino MEGA grupo 1

En el caso que el maestro esté solicitando los datos de los sensores del esclavo, se ejecutará la función solicitanDatos en la placa Arduino. En la Figura 5.27 se observa el método utilizado en este proyecto para recorrer las tablas valoresD y valoresA. Esta función debe estar sincronizada con el bucle de petición de datos de la función loop en la placa nodeMCU, enviándose exactamente los Bytes solicitados por nodeMCU.

```
113 void solicitanDatos(){
114   int valoresD = sizeof(valoresIN_D)/sizeof(valoresIN_D[0]);
115   int valoresA = sizeof(valoresIN_A)/sizeof(valoresIN_A[0]);
116
117   int contador=0;
118   while(contador<3){
119       if(cont_tx < valoresD){
120         Wire.write(valoresIN_D[cont_tx].c_str());
121         cont_tx++; //contador de todas las entradas enviadas
122         contador++; //Contador de entradas máximas por transmision
123       }
124
125      else if((cont_tx-valoresD) < valoresA){
126         Wire.write(valoresIN_A[cont_tx-valoresD].c_str());
127         cont_tx++;
128         contador++;
129       }
130      else{
131         cont_tx=0;
132         contador=3;
133      }
134   }
135 }
```

Figura 5.27 - Función solicitanDatos placa Arduino MEGA grupo 1

Dentro del bucle infinito del programa, únicamente se obtendrán los datos de los sensores y se actualizarán las variables que almacenan dichos datos. En este fragmento de código (Figura 5.28), se observa cómo se espera a que se haya cumplido el período de solicitud de datos.

La obtención de los datos de los sensores se realiza en las líneas 50, 51, 52 y 53. Los valores del sensor

DHT22 pueden ajustarse directamente a la longitud deseada de 4 caracteres, esto incluye dígitos y símbolos, con lo que se actualizan las cadenas que almacenan dichos valores.

```
46 void loop() {
47   if((unsigned long)(millis() - millisPrevio) >= interval){
48     millisPrevio=millis();
49     //------------------ DHT22 --------------------
50     float hum = dht.readHumidity();
51     float temp = dht.readTemperature();
52     float sen = dht.computeHeatIndex(temp, hum, false);
53     int monox = analogRead(asociacionA[0]);
54
55
56     valoresIN_D[0]= "+D091-" + String (temp,1);  //Esto da 5 caracteres: 42.50. Poniendo ",1", cogemos solo un decimal
57     valoresIN_D[1] = "+D092-"+ String (hum,1);
58     valoresIN_D[2] = "+D093-"+ String (sen,1);
59
```

Figura 5.28 - Función loop Arduino MEGA grupo 1. Fragmento 1

El sensor de monóxido de carbono puede aportar valores más amplios, con lo que el resultado debe adaptarse a una longitud de cuatro caracteres. La forma de conseguir esto es la ilustrada en la Figura 5.29.

```
60     //------------------- MQ7 ----------------------
61     if(monox<10){ //1 dígito
62       String ceros = "000";
63       ceros += String(monox);
64      valoresIN_A[0] = "+A030-" + ceros;
65     }
66     else if(monox<100){ //2 dígitos
67       String ceros = "00";
68       ceros += String(monox);
69       valoresIN_A[0] = "+A030-" + ceros;
70     }
71     else if(monox<1000){ //3 digitos
72       String ceros = "0";
73       ceros += String(monox);
74       valoresIN_A[0] = "+A030-" + ceros;
75     }
76     else{
77       valoresIN_A[0] = "+A030-" + String(monox);
78     }
79   }
80 }
```

Figura 5.29 - Función loop Arduino MEGA grupo 1. Fragmento 2

5.3.2.3 Sensores y actuadores

Este apartado se ha reservado para la explicación del código de los sensores y actuadores utilizados en el proyecto documentado.

5.3.2.3.1 Sensor DHT22

Para obtener los datos del sensor de temperatura y humedad utilizado en este proyecto se emplea, dependiendo del dato a medir, una de las funciones de la variable de tipo DHT.

```
36 DHT dht(asociacionD[7], DHT22); //DHTPIN, DHTTYPE
```

Figura 5.30 - Declaración variable tipo DHT

Dada la definición de la variable tipo DHT de la Figura 5.30, dónde se especifica, mediante parámetros, el pin de la placa Arduino conectado a la salida DATA del sensor y el modelo del sensor, pueden obtenerse los distintos datos que aporta el sensor DHT22, según lo recogido en la Figura 5.31

```
float hum = dht.readHumidity();
float temp = dht.readTemperature();
float sen = dht.computeHeatIndex(temp, hum, false);
```

Figura 5.31 - Obtención datos sensor DHT22

Los datos arrojados por el sensor DHT22 son todos de tipo *float*. En el caso de querer obtener el dato de sensación de temperatura, tendremos que especificar con qué valores de temperatura y humedad tiene que realizar el cálculo. El tercer argumento indica si queremos que el dato esté en grados Celsius o Fahrenheit. En la Figura 5.31 el tercer argumento tiene el valor "false" para obtener grados Celsius

5.3.2.3.2 Sensor MQ-7

En el caso del sensor de Monóxido de Carbono, el valor obtenido es un entero. Este sensor no se basa en funciones para la obtención de los datos, sino que se realiza leyendo la lectura del pin analógico y/o digital (Figura 5.32).

La lectura del pin digital consiste en ver, mediante el regulador incorporado, si el valor de monóxido pasa el umbral establecido; mientras que la salida analógica del sensor nos informa del valor de monóxido en el aire.

```
int monox = analogRead(asociacionA[0]);
```

Figura 5.32 - Obtención datos sensor MQ7

5.3.2.3.3 Sensor de apertura MC38

Para detectar si un elemento móvil se encuentra abierto o cerrado se ha usado el sensor MC38. Este sensor, al ser un elemento crítico por ser un dispositivo que aporta datos de seguridad, estará conectado directamente a la placa nodeMCU. El período de muestreo del valor es independiente del resto de sensores no críticos, cuyo intervalo de muestreo puede ser significativamente mayor.

La lectura del sensor MC38 se realiza mediante el conversor de nivel a través de uno de los pines digitales de la placa nodeMCU usándose para ello la lectura normal de un pin digital (Figura 5.33).

```
String valor="";
valor+=digitalRead(D0);
```

Figura 5.33 - Obtención datos sensor MC38

5.3.2.3.4 Actuador módulo relé

El módulo relé es capaz de actuar con tensión de hasta 230 Voltios. Se controla mediante un pin digital de la placa Arduino. Escribiendo nivel alto o bajo en este pin hacemos que el módulo relé conmute hacia un lado u otro, con lo que la forma de controlar este tipo de actuadores es simplemente cambiando el voltaje del pin Arduino al que está conectado.

Un aspecto a tener en cuenta es determinar la posición del conmutador cuando desde el Arduino se le da nivel alto y bajo, ya que en conmutadores de distinto fabricante podría no ser el mismo.

En este proyecto se actúa sobre el módulo relé según lo ordenado desde openHAB. Para ello, se obtiene el pin digital sobre el que se actúa a partir del *topic* del paquete MQTT. Este *topic*, que viene dado como cadena de caracteres, se convierte a un entero mediante la función "toInt()". El segundo parámetro especifica si el pin dado como primer parámetro se pone a nivel alto o bajo. En el caso de la Figura 5.34 se está poniendo a nivel alto.

```
digitalWrite(pin.toInt(), HIGH);
```

Figura 5.34 - Administración módulo relé

5.3.2.3.5 Sensor ACS712

El caso del sensor de corriente ACS712 puede que sea el más complejo de los vistos en este proyecto.

Para calcular la corriente y potencia con este sensor, se han declarado una serie de variables globales que nos ayudarán a una lectura lo más precisa posible. En la Figura 5.35 se observan las variables globales utilizadas para este sensor.

```
4  //Definimos variables para el sensor ASC712
5  float sensibilidad = 0.100;
6  float ruido = 0.000;
7  const int sensorIntensidad = A1;
8  float valorReposo = 2.50;
9  float intensidadPico=0;
10 float potencia=0;
11 float tensioneDeRed = 230.0;
```

Figura 5.35 - Variables globales sensor ACS712

- sensibilidad: valor de la sensibilidad del modelo del sensor.
- ruido: valor de la corriente media cuando el aparato que se está midiendo se encuentra apagado. Esta medida podrá cambiar de un circuito a otro.
- sensorIntensidad: recoge el pin Arduino usado para la lectura del sensor ACS712
- valorReposo: nivel de voltaje que aporta el sensor con una corriente de 0 Amperios; es decir, con el elemento a medir estando desconectado de la corriente. El valor teórico es de 2.5 Voltios, aunque este nivel puede variar en la práctica.
- intensidadPico: valor máximo de la corriente durante el intervalo de muestreo.
- potencia: potencia consumida por el equipo por dónde circula la corriente.
- tensiónDeRed: nivel de tensión del circuito. En este proyecto se están controlando las medidas de una bombilla conectada a la red eléctrica doméstica. La tensión alterna de un domicilio en España está regulada a 230V.

La forma de medir la corriente que circula a través del sensor se basa en una medición de lecturas durante un intervalo de tiempo. Cuanto mayor sea este intervalo, mayor será la precisión. En la función leerCorriente (Figura 5.36) se realiza dicha medición.

```
124 void leerCorriente(){
125   float valorVoltajeSensor;
126   float corriente=0;
127   long tiempo=millis();
128   float intensidadMaxima=0;
129   float intensidadMinima=0;
130   while(millis()-tiempo<500){//realizamos mediciones durante 0.5 segundos
131     valorVoltajeSensor = analogRead(sensorIntensidad)*(5.0 / 1023.0);//lectura del sensor en voltios
132     corriente=0.9*corriente+0.1*((valorVoltajeSensor-valorReposo)/sensibilidad);
133     if(corriente>intensidadMaxima){
134       intensidadMaxima=corriente;
135     }
136     if(corriente<intensidadMinima){
137       intensidadMinima=corriente;
138     }
139   }
140   intensidadPico = (((intensidadMaxima-intensidadMinima)/2)-ruido);
141   potencia = intensidadPico*0.707*230.0; //Intensidad RMS = Ipico/(2^1/2) , P=I*V watts.
142
143 }
```

Figura 5.36 - Función leerCorriente sensor ACS712

Para realizar la medición anteriormente comentada, se hace uso de las siguientes variables:

- valorVoltajeSensor: salida del sensor.
- corriente: variable usada para guardar el valor de corriente de una lectura.
- tiempo: su uso es controlar el período de muestreo.
- intensidadMaxima: almacena el valor más alto de corriente de todas las lecturas en una medición.
- intensidadMinima: valor más bajo de corriente en una medición.

El código reflejado en la Figura 5.36 muestra que, mientras no haya concluido el tiempo establecido para una medición, se realizarán lecturas de la corriente. En este caso, el período de muestreo es de 500 milisegundos.

Durante este tiempo se leerán las lecturas de tensión aportadas por el sensor. El sensor da 2.5 voltios para una corriente de 0 Amperios y un máximo de 5 V para la corriente máxima. El conversor analógico/digital de Arduino usará 1023 muestras para transformar la tensión analógica en un voltaje digital que, según lo mencionado, para el sensor ACS712, estará en el rango 2.5V-5V.

En la línea 132 de la Figura 5.36 se está aplicando un filtro para maximizar la precisión, con este filtro se está cogiendo una media de 10 medidas de corriente.

Finalmente se calcula la intensidad de pico y la potencia, dónde la potencia es el resultado de multiplicar la corriente eficaz por la tensión.

Se ha de destacar que el valor enviado a openHAB es la corriente eficaz, calculada como el producto de la corriente de pico por 0.707.

5.4 Administración openHABian

En esta sección se tratará en detalle la administración de openHABian realizada en este proyecto. Esto incluye la administración de los servicios ya instalados openHAB y mosquitto.

5.4.1 Localización de ficheros

El primer paso para la correcta administración de un servicio es saber localizar los ficheros de configuración. En este proyecto se administran dos servicios cuyos ficheros de configuración son tratados en esta subsección.

5.4.1.1 Ficheros openHAB

openHAB ubica sus ficheros de configuración según la forma en la que se haya instalado su servicio. En el caso que nos ocupa (Tabla 5.1), los ficheros están localizados de la siguiente manera:

Tabla 5.1 - Localización ficheros de configuración openHAB

Funcionalidad	Ubicación
Directorio de aplicación openHAB	`/usr/share/openhab2`
Directorio de características adicionales o *add-on*	`/usr/share/openhab2/addons`
Directorio de configuración dispositivos e interfaces	`/etc/openhab2`
Archivo de configuración del servicio	`/etc/default/openhab2`
Directorio de logs	`/var/log/openhab2`

Directorio de aplicación openHAB: `/usr/share/openhab2`

En este directorio, Figura 5.37, se encuentran los archivos referentes a la configuración de la aplicación openHAB basados en la aplicación de Ecplise Smart Home.

```
[02:44:22] openhabian@openHABianPi:/usr/share/openhab2$ ls
addons  LICENSE.TXT  runtime  start_debug.sh  start.sh
[02:46:00] openhabian@openHABianPi:/usr/share/openhab2$ ls runtime/
bin  etc  lib  services.cfg  system
```

Figura 5.37 - Directorio de aplicación openHAB

Directorio de configuración dispositivos e interfaces: `/etc/openhab2`

Este directorio, Figura 5.38, será el más usado para la configuración de openHAB en este proyecto. Dentro de este directorio se definen las things, ítems, ...

```
[02:49:52] openhabian@openHABianPi:/etc/openhab2$ ls
html  icons  items  persistence  rules  scripts  services  sitemaps  sounds  things  transform
```

Figura 5.38 - Directorio de configuración dispositivos e interfaces

Actualmente ya se ha hecho uso de uno de estos subdirectorios. En el subdirectorio services, Figura 5.39, se

encuentran los archivos de configuración de los binding instalados, como es el caso del binding mqtt1

```
[03:03:40] openhabian@openHABianPi:/etc/openhab2/services$ ls
addons.cfg  dashboard.cfg  mqtt.cfg  mqtt-eventbus.cfg  readme.txt  runtime.cfg
```

Figura 5.39 - Subdirectorio services

Archivo de configuración del servicio: `/etc/default/openhab2`

En este archivo pueden configurarse diferentes opciones del servicio openHAB. Algunas de las opciones configurables por este fichero son: Puertos en los que el servidor web de openHAB atenderá peticiones HTTP/HTTPS, rutas de los directorios de configuración de openHAB, usuarios y grupos propietarios de openHAB.

Directorio de logs: `/var/log/openhab2`

En este directorio, Figura 5.40, se encuentran los dos archivos de logs que proporciona openHAB.

- events.log: informa de cualquier cambio en el valor de los ítems definidos.
- openhab.log: aporta información sobre los eventos del servicio openHAB.

```
[02:56:22] openhabian@openHABianPi:~$ cd /var/log/openhab2/
[02:56:30] openhabian@openHABianPi:/var/log/openhab2$ ls
audit.log  events.log  openhab.log  Readme.txt
```

Figura 5.40 - Directorio de logs

Directorio de características adicionales o add-on: `/usr/share/openhab2/addons`

openHAB tiene la ventaja de que es altamente modular, pudiendo instalar características adicionales si son necesarias. Estas características tienen sus ficheros de configuración en este directorio. En este proyecto no se hace uso de dicho directorio.

5.4.1.2 Ficheros mosquitto

En el caso del servidor mosquitto el número de archivos usados para su administración es bastante menor. En este proyecto se han utilizado dos archivos para la administración de Mosquitto:

Archivo de configuración: `/etc/mosquitto/conf.d/mosquitto.conf`

Este archivo ha sido utilizado para configurar tanto el nombre del servidor, como el protocolo usado (MQTT), y el puerto de escucha del servidor.

Archivo de logs: `/var/log/mosquitto/mosquitto.log`

Toda la información relevante con el servidor Mosquitto es recogida en este archivo. Este fichero nos servirá para la administración del servicio, comprobando el correcto funcionamiento del servidor o para buscar información acerca de algún comportamiento anómalo.

5.4.2 Definición de things

Las *thing* son declaradas en openHAB a través de archivos en el directorio `/etc/openhab2/things`. Se pueden declarar *things* en archivos diferentes, lo que ayuda a la organización de los mismos.

Los archivos usados para la declaración de *things* deben tener la extensión "`.things`".

Este proyecto tiene como peculiaridad que los dispositivos aportan datos mediante el protocolo MQTT. Estos datos están ligados a un *topic* concreto, no a una dirección IP, con lo que no es necesario declarar una *thing* en

el directorio *things*. La conexión entre elemento software y físico puede hacerse directamente en la declaración del ítem, por esta razón no es necesario crear ningún archivo con extensión ".`things`" en este proyecto.

5.4.3 Definición de ítems

Los ítems son declarados dentro del directorio mediante archivos con extensión ".`items`". Al igual que en el caso de las *thing*, los ítems pueden definirse en archivos separados.

En este proyecto se han creado dos archivos de ítems, uno por cada placa nodeMCU (Figura 5.41).

```
[03:21:54] openhabian@openHABianPi:/etc/openhab2/items$ ls
placa1.items  placa2.items  readme.txt
```

Figura 5.41 - Directorio /etc/openhab2/items

Dentro de estos archivos únicamente se definen los ítems que podrán ser mostrados en los *sitemaps*. La definición de los ítems sigue una sintaxis específica (Figura 5.42):

```
itemtype itemname "labeltext [stateformat]" <iconname> (group1, group2, ...) ["tag1", "tag2", ...] {bindingconfig}
```

Figura 5.42 - Sintaxis item

El orden de los campos debe mantenerse, siendo obligatorios los campos itemtype e itemname. El resto de campos son opcionales. En la definición de un ítem, los distintos campos pueden ir separado por uno o más espacios, incluso mediante tabulaciones.

El campo itemtype ya fue abordado en la subsección 2.2.2. itemname es el nombre que se usará para referirnos al ítem en concreto, por lo que debe ser único, y no sólo en el archivo que está siendo definido; debe ser único de entre todos los archivos ".`items`"

Cuando hayamos incluido este ítem en el *sitemap* correspondiente y éste se cargue en la interfaz de usuario, el texto flotante que hace referencia al ítem es el campo labeltext. Este campo puede incluir también un stateformat, dónde se indica en qué formato debe representarse el valor del ítem.

Para ayudar al labeltext, puede incluirse una imagen que ayude a ilustrar qué información aporta el ítem que se está visualizando. openHAB incluye numerosos iconos que pueden ser usados, los cuales están recogidos en [19].

Con el fin de facilitar la tarea de administrar los ítems, openHAB pone a disposición del usuario los group. De esta forma, el sexto campo en la declaración de un ítem nos permite agrupar ítems. Un ítem puede estar asociado a varias categorías, siempre que éstas estén separadas por comas.

Como se documentó en la subsección 2.2.2, un grupo es también un tipo de ítem. Esto viene a decir que podemos agrupar grupos dentro de otros. Cabe destacar que antes de asociar un ítem a un grupo, el grupo debe estar creado.

Las etiquetas o tags permiten al usuario caracterizar al ítem. Éstas etiquetas pueden ser usadas por los *add-ons* para interactuar con los ítems. En este proyecto no se ha definido ningún ítem con etiquetas al considerar que no aportaban ninguna utilidad en el desempeño del proyecto.

El último campo en la definición de un ítem es el bindingconfig. En este campo se indica la asociación dedicada entre el ítem y la *thing*. Es el que nos permitirá comunicarnos con el objeto físico vía software.

En la Figura 5.43 se ilustra la definición de un ítem de tipo grupo o *Group*. Este grupo está unívocamente identificado en todo el servicio openHAB por la palabra "Temperatura". El texto que se mostrará cuando se quiera representar este ítem es "Temperatura", ya que es el tercer campo de la definición. Así mismo, se ha estipulado que el icono que represente a este grupo es el icono dado por "temperature".

```
Group Temperatura "Temperatura" <temperature> (Salon)
```

Figura 5.43 - Definición ítem tipo Group

Por último, tenemos el campo group. Con este campo estamos indicando que, cuando se represente el grupo "Salon", también se mostrarán todos los ítems incluidos dentro del grupo "Temperatura".

En el archivo placa1.items se define el ítem “Salon_temp” (Figura 5.44). Este ítem es de tipo numérico, con lo que la representación del ítem será un número. Como se observa en el campo labeltextFigura 5.44, el número a representar tendrá un decimal, y estará representará grados Celsius.

El ítem “Salon_temp” pertenece a dos grupos según la definición de la Figura 5.44. El grupo “Temperatura” estaba incluido en el grupo “Salon” con lo que, a efectos generales, el ítem “Salon_temp” se encuentra en tres grupos: “Temperatura”, “Sensores” y “Salon”.

```
Number Salon_temp "Temperatura [%.1f ºC]" <temperature> (Temperatura, Sensores) {mqtt="<[mosquitto:D091-1:state:
default]" }
```

Figura 5.44 - Definición ítem “Salon_temp”

El último campo del ítem es la asociación dedicada, que, en este proyecto, se ha utilizado el llamado mqtt1. La forma de utilizar esta asociación es la siguiente:

- Primero, se indica el protocolo: MQTT.
- A continuación del carácter “=” se incluye carácter “mayor/menor” (>/<), para posteriormente indicar entre corchetes la orden asociada al carácter mayor/menor. Se utilizará el carácter “menor” (<) para indicar que el comando asociado es entrante a openHAB. En caso contrario se utilizará el comando “mayor” (>).
- La sintaxis de la orden es:
    - En primer lugar, el nombre del servidor MQTT seguido del carácter de separación “:”. Con la configuración hecha hasta ahora, el nombre del servidor es “mosquitto”.
    - En segundo lugar, se especifica el *topic* del mensaje MQTT.
    - Seguidamente, se hace una diferencia entre la acción a realizar de entre los dos tipos existentes: actualizar estado (state), para mensajes entrantes, o enviar un paquete MQTT (command), para mensajes salientes.
        - En el caso de la acción “state”, después del carácter de separación se indica el valor inicial. En el caso de no querer un valor inicial, se indica mediante el valor “default”.
        - Si la acción es enviar un paquete, entre los caracteres de separación debe especificarse el valor del ítem en openHAB, y el valor que se quiere enviar a la red.

En la Figura 5.44, el mensaje entrante actualizará el valor del ítem “Salon_temp”, el cual no tiene valor inicial; mientras que en la Figura 5.45, por cada actualización del valor del ítem, se enviarán paquetes PUBLISH dónde, si el valor del ítem es “ON”, el *payload* del paquete será “1”, y si el valor del ítem es “OFF”, en el paquete se enviará un “0”

```
Switch Pasillo_luz1 "Luz pasillo 1" <light> (Pasillo, Luces) {mqtt=">[mosquitto:D040-1:command:ON:1],>[mosquitto
:D040-1:command:OFF:0]" }
```

Figura 5.45 - Definición ítem "Luz pasillo 1"

5.4.4 Definición de reglas

En la administración de los datos que aportan los sensores, podemos querer ciertos comportamientos según los datos recibidos, como puede ser la activación de un ventilador si se detecta humo en una habitación. Para estas situaciones openHAB pone a disposición del administrador las “reglas”.

Las reglas, al igual que los ítems, se encuentran definidas en un archivo, aunque en este caso la extensión del archivo debe ser “`.rules`” y estar ubicado en el directorio `/etc/openhab2/rules`.

Dentro de un mismo fichero pueden definirse varias reglas, las cuales están compuestas en tres partes:

La primera parte de una regla, Figura 5.46, es el nombre de la misma, y debe ser único de entre todas las reglas. El nombre de la regla debe estar entrecomillado y precedido de la palabra reservada “rule”.

```
rule "Monoxido Carbono"
```

Figura 5.46 - Nombramiento de regla openHAB

El segundo campo de una regla, Figura 5.47, viene dado por la palabra reservada "when", y debe especificar cuándo se hace uso de esta regla. En el proyecto que aquí se documenta se ha creado una regla que se lanzará cada vez que el ítem "Cocina_monoxido" cambie de estado.

```
when
        Item Cocina_monoxido received update
```

Figura 5.47 - Condición regla openHAB

Por último, viene el comportamiento de openHAB una vez se haya cumplido la condición. El comportamiento debe especificarse entre las palabras reservadas "then" y "end". La regla utilizada en este proyecto, Figura 5.48, es usada para convertir los datos aportados por el sensor de monóxido a un texto que será actualizado según el nivel del mismo, de forma que sea visualmente más amigable al usuario.

```
then
        if(Cocina_monoxido.state <=150){
                Cocina_nivel_monoxido.postUpdate("BAJO")
        }
        else if(Cocina_monoxido.state > 150 && Cocina_monoxido.state < 1000){
                Cocina_nivel_monoxido.postUpdate("MEDIO")
        }
        else{
                Cocina_nivel_monoxido.postUpdate("ALTO")
        }
end
```

Figura 5.48 - Comportamiento regla openHAB

5.4.5 Elaboración de sitemaps

Los *sitemaps* son archivos con extensión ".sitemap" que preparan la representación visual de los ítems. En el directorio de openHAB podemos tener varios *sitemp* creados, pero únicamente uno de ellos será el que se muestre en la interfaz de usuario. En este proyecto el archivo que servirá para la configuración de la representación visual es "casa.sitemap" Figura 5.49.

```
[05:09:44] openhabian@openHABianPi:/etc/openhab2/sitemaps$ ls
casa.sitemap  readme.txt
```

Figura 5.49 - Directorio /etc/openhab2/sitemaps

Al igual que ocurre con los ficheros anteriormente vistos, este tipo de archivo tiene una sintaxis propia. Todo archivo con extensión "`.sitemap`" debe comenzar con un elemento *sitemap*, tal y cómo se muestra en la Figura 5.50

```
sitemap <sitemapname> label="<title of the main screen>" {
    [all sitemap elements]
}
```

Figura 5.50 - Sintaxis sitemaps

Tras la palabra reservada "sitemap", se escribe el sitemapname, el cual debe ser igual que el nombre del archivo. En el campo label se indica un título para la pestaña de la interfaz de usuario. Los elementos para representar los ítems están recogidos en la Tabla 2.3 - Elementos de un sitemap, y tendrán que ir obligatoriamente entre los caracteres de llaves:"{}"

En la elaboración del sitemap de este proyecto, cuyo código se encuentra recogido en el Anexo C, se han utilizado diferentes *Frame* para tener más organizada la interfaz de usuario. Dentro de estos *Frames* nos hemos ayudado de los grupos creados en los ficheros placa1.items y placa2.items, de forma que el código de casa.sitemap queda bastante claro y conciso.

Un elemento que destacar de este código es el utilizado para la videovigilancia. La videovigilancia propuesta en este proyecto se basa en la reutilización de dispositivos tales cómo móviles, tabletas, ... a los que

actualmente no se les está dando ningún uso. Mediante la instalación de la aplicación "IP Webcam" se convierte el dispositivo en un servidor de imágenes tomadas desde la cámara del mismo.

El servidor instalado en nuestro teléfono móvil tiene asignada la dirección IP 192.168.1.129 y puerto de escucha 8282, Figura 5.51. La dirección IP es la primera dirección del rango DHCP dinámico, mientras que el puerto de escucha es configurable por la aplicación.

Como método de seguridad, para poder acceder a las imágenes compartidas por el servidor, se requiere de un usuario y contraseña configurados mediante la aplicación del servidor de vídeo. En este proyecto se ha usado una tasa de refresco de imagen de 6000 milisegundos

```
Frame label="Camara"{

        Image url="http://192.168.1.129:8282/video?user=camara&pwd=seguridad" label="Camara principal" refresh=6000
}
```

Figura 5.51 - Configuración videovigilancia sitemap

5.4.6 Comprobación de logs

Una de las tareas básicas para la administración de un servicio es la comprobación de errores o eventos. Los servicios reportan información de estas situaciones mediante cadenas de texto, también llamados logs, en archivos con extensión ".`log`"

5.4.6.1 Mosquitto

En el caso del servidor mosquitto, el archivo con la información del servicio se encuentra en el directorio `/var/log/mosquitto`, y el nombre de dicho archivo es `mosquitto.log`.

Dada la información suministrada por la Figura 5.52, puede observarse cómo el servicio mosquitto arranca con la configuración dada en el fichero `/etc/mosquitto/mosquitto.conf`. Una vez tomada la configuración, abre los puertos de escucha especificados en dicho archivo, en este caso el puerto que configuramos fue el 1883.

Podemos observar también en la Figura 5.52 el comportamiento del servicio ante una nueva conexión. Según esta información, la placa 1 ha pedido conexión con el servidor MQTT, a lo que este responde mediante un paquete CONNACK. Posteriormente al inicio de sesión, la placa 1 se ha suscrito a varios *topics* con la QoS 0, a lo que el servidor ha aceptado enviando paquetes SUBACK.

```
[16:18:17] openhabian@openHABianPi:/var/log/mosquitto$ cat mosquitto.log
1558275424: mosquitto version 1.4.10 (build date Wed, 17 Oct 2018 19:03:03 +0200) starting
1558275424: Config loaded from /etc/mosquitto/mosquitto.conf.
1558275424: Opening ipv4 listen socket on port 1883.
1558275424: Opening ipv6 listen socket on port 1883.
1558275432: New connection from 192.168.1.3 on port 1883.
1558275432: New client connected from 192.168.1.3 as placa1 (c1, k15).
1558275432: Sending CONNACK to placa1 (0, 0)
1558275432: Received SUBSCRIBE from placa1
1558275432:     D080-1 (QoS 0)
1558275432: placa1 0 D080-1
1558275432: Sending SUBACK to placa1
1558275432: Received SUBSCRIBE from placa1
1558275432:     D070-1 (QoS 0)
1558275432: placa1 0 D070-1
1558275432: Sending SUBACK to placa1
1558275432: Received SUBSCRIBE from placa1
1558275432:     D060-1 (QoS 0)
1558275432: placa1 0 D060-1
1558275432: Sending SUBACK to placa1
1558275432: Received SUBSCRIBE from placa1
1558275432:     D050-1 (QoS 0)
1558275432: placa1 0 D050-1
1558275432: Sending SUBACK to placa1
1558275432: Received SUBSCRIBE from placa1
1558275432:     D040-1 (QoS 0)
1558275432: placa1 0 D040-1
1558275432: Sending SUBACK to placa1
1558275432: Received SUBSCRIBE from placa1
1558275432:     D030-1 (QoS 0)
1558275432: placa1 0 D030-1
```

Figura 5.52 - Log de inicio Mosquitto y suscripción placa 1

El proyecto constaba de tres dispositivos conectados al servidor MQTT, dos placas nodeMCU, y openHAB. En la Figura 5.53 puede observarse la conexión de openHAB al servidor MQTT y la suscripción a alguno de los *topics* suscritos por dicho servicio.

```
1558275484: New connection from 192.168.1.2 on port 1883.
1558275484: New client connected from 192.168.1.2 as paho70680773409 (c1, k60).
1558275484: Sending CONNACK to paho70680773409 (0, 0)
1558275484: Received SUBSCRIBE from paho70680773409
1558275484:     A001-2 (QoS 0)
1558275484: paho70680773409 0 A001-2
1558275484: Sending SUBACK to paho70680773409
1558275484: Received SUBSCRIBE from paho70680773409
1558275484:     A002-2 (QoS 0)
1558275484: paho70680773409 0 A002-2
1558275484: Sending SUBACK to paho70680773409
1558275484: Received SUBSCRIBE from paho70680773409
1558275484:     D001-2 (QoS 0)
1558275484: paho70680773409 0 D001-2
1558275484: Sending SUBACK to paho70680773409
```

Figura 5.53 - Logs de inicio de sesión openHAB a mosquitto y suscripciones

En la subsección 4.2.3 se habló de los paquetes de control MQTT entre los que figuran los paquetes PINGRESQ y PINGRESP, cuya finalidad era informar de la conectividad ente cliente y servidor. Estos paquetes quedan reflejados en el archivo de logs de mosquitto, como se refleja en la Figura 5.54, dónde la placa 1 informa de su presencia al servidor y éste le responde.

```
1558275447: Received PINGREQ from placa1
1558275447: Sending PINGRESP to placa1
```

Figura 5.54 - Logs mosquitto paquetes PINGREQ y PINGRESP

La finalidad del protocolo MQTT es la comunicación de mensajes desde un cliente MQTT que publique información en ciertos *topics*, hasta otro cliente MQTT que esté publicado a dichos *topics*. En la Figura 5.55 puede verse cómo, si no existe ningún cliente suscrito a un *topic*, los paquetes PUBLISH no son reenviados a nadie, mientras que, como se ilustra en la Figura 5.56, cuando existe un cliente suscrito a un *topic*, el servidor MQTT, inmediatamente después de recibir un paquete PUBLISH y comprobar que existen suscriptores de dichos *topics*, manda paquetes PUBLISH a dichos equipos.

```
1558275432: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
1558275432: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D092-1', ... (4 bytes))
1558275432: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D093-1', ... (4 bytes))
1558275432: Received PUBLISH from placa1 (d0, q0, r0, m0, 'A000-1', ... (4 bytes))
1558275439: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
1558275439: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D092-1', ... (4 bytes))
1558275439: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D093-1', ... (4 bytes))
1558275439: Received PUBLISH from placa1 (d0, q0, r0, m0, 'A000-1', ... (4 bytes))
1558275446: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
1558275446: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D092-1', ... (4 bytes))
1558275446: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D093-1', ... (4 bytes))
1558275446: Received PUBLISH from placa1 (d0, q0, r0, m0, 'A000-1', ... (4 bytes))
```

Figura 5.55 - Log envío paquetes PUBLISH sin suscriptores

```
1558275488: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
1558275488: Sending PUBLISH to paho70680773409 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
1558275488: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D092-1', ... (4 bytes))
1558275488: Sending PUBLISH to paho70680773409 (d0, q0, r0, m0, 'D092-1', ... (4 bytes))
1558275488: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D093-1', ... (4 bytes))
1558275488: Sending PUBLISH to paho70680773409 (d0, q0, r0, m0, 'D093-1', ... (4 bytes))
1558275488: Received PUBLISH from placa1 (d0, q0, r0, m0, 'A000-1', ... (4 bytes))
1558275488: Sending PUBLISH to paho70680773409 (d0, q0, r0, m0, 'A000-1', ... (4 bytes))
1558275492: Received PINGREQ from placa1
1558275492: Sending PINGRESP to placa1
1558275495: Received PUBLISH from placa1 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
1558275495: Sending PUBLISH to paho70680773409 (d0, q0, r0, m0, 'D091-1', ... (4 bytes))
```

Figura 5.56 - Log envío paquetes PUBLISH con suscriptores

En este proyecto no se abarca la posibilidad de que un dispositivo MQTT se desconecte (no se han programado las placas para desconectarse del servidor MQTT ya que no queremos este comportamiento). La única opción de que un dispositivo se desconecte es debido a un comportamiento anómalo, el cual puede deberse a diferentes causas, pero todas ellas se verán reflejados de la misma manera en el archivo de logs de mosquitto, ilustrada con la Figura 5.57

```
1558275566: Client placa1 has exceeded timeout, disconnecting.
1558275566: Socket error on client placa1, disconnecting.
```

Figura 5.57 - Desconexión MQTT placa 1

5.4.6.2 openHAB

Tal y cómo se habló en el apartado 5.4.1.1, el directorio de logs de openHAB se encuentran en el directorio `/var/log/openhab2/`. Dentro de este directorio encontramos dos archivos con extensiones ".log": `events.log` y `openhab2.log`, los cuales contendrán información acerca de los valores de los ítems y del servicio openHAB.

Dada la configuración actual de openHAB, en la Figura 5.58 se muestra un arranque correcto del servicio.

```
2019-07-09 00:45:55.229 [INFO ] [el.core.internal.ModelRepositoryImpl] - Loading model 'casa.sitemap'
2019-07-09 00:45:56.582 [INFO ] [.dashboard.internal.DashboardService] - Started Dashboard at http://192.168.1.2:8080
2019-07-09 00:45:56.585 [INFO ] [.dashboard.internal.DashboardService] - Started Dashboard at https://192.168.1.2:8443
2019-07-09 00:45:57.560 [INFO ] [arthome.ui.paper.internal.PaperUIApp] - Started Paper UI at /paperui
2019-07-09 00:45:57.917 [INFO ] [penhab.io.transport.mqtt.MqttService] - MQTT Service initialization completed.
2019-07-09 00:45:57.921 [INFO ] [t.mqtt.internal.MqttBrokerConnection] - Starting MQTT broker connection 'mosquitto'
```

Figura 5.58 Logs openhab2.log

mientras que, un fragmento del archivo `events.log` se muestra en la Figura 5.59

```
2019-07-09 00:51:07.997 [ome.event.ItemCommandEvent] - Item 'Pasillo_luz2' received command OFF
2019-07-09 00:51:08.015 [vent.ItemStateChangedEvent] - Pasillo_luz2 changed from ON to OFF
2019-07-09 00:51:08.288 [vent.ItemStateChangedEvent] - CorrienteEntrada changed from 0.11 to 0.10
2019-07-09 00:51:08.292 [vent.ItemStateChangedEvent] - PotenciaEntrada changed from 25.5 to 23.2
2019-07-09 00:51:08.407 [vent.ItemStateChangedEvent] - Salon_temp changed from 25.9 to 26.0
2019-07-09 00:51:08.413 [vent.ItemStateChangedEvent] - Salon_hum changed from 61.0 to 61.1
```

Figura 5.59 - Logs events.log

5.4.7 Interfaz de usuario

En esta subsección se va a mostrar mediante capturas, la interfaz resultante de la configuración realizada en este proyecto con todos los dispositivos y sensores en funcionamiento.

Por razones obvias se han omitido las figuras de videovigilancia, además se muestran únicamente los menús: "general" (Figura 5.60), "Luces" (Figura 5.61) y "Todos los sensores" (Figura 5.62), omitiéndose los menús por habitaciones, ya que los sensores mostrados en las habitaciones serán alguno de los mostrados en los otros menús, dependiendo de la configuración.

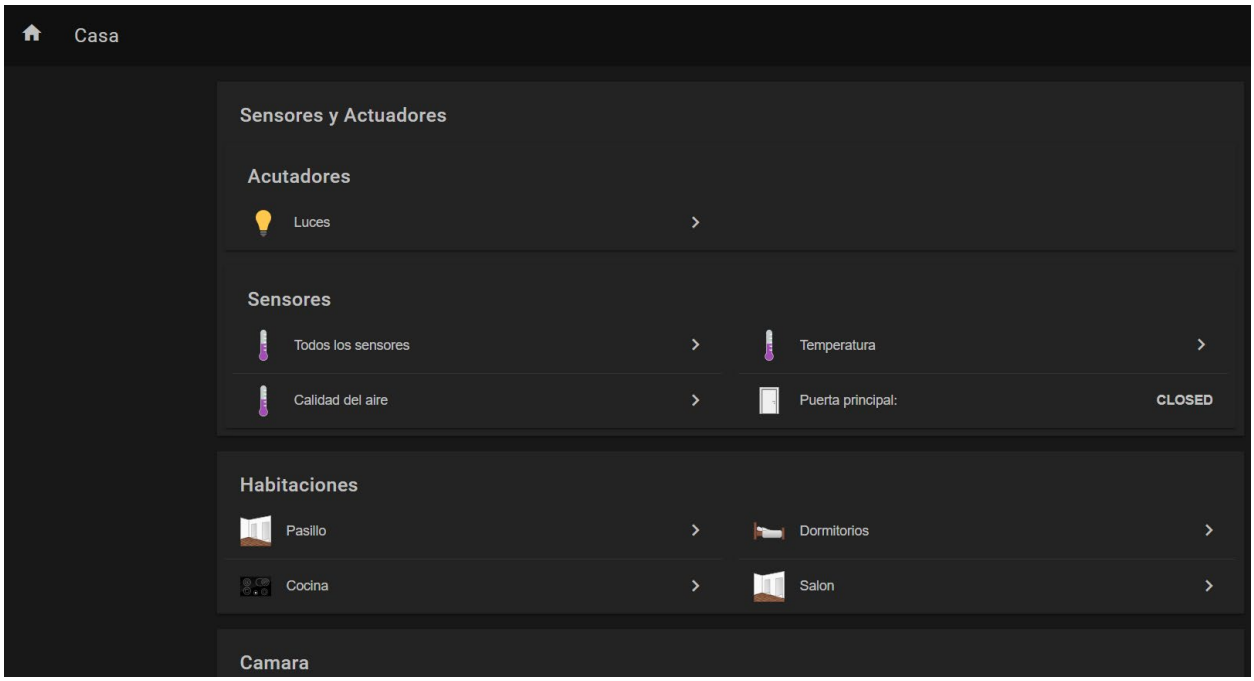


Figura 5.60 - Interfaz de usuario menú "general"

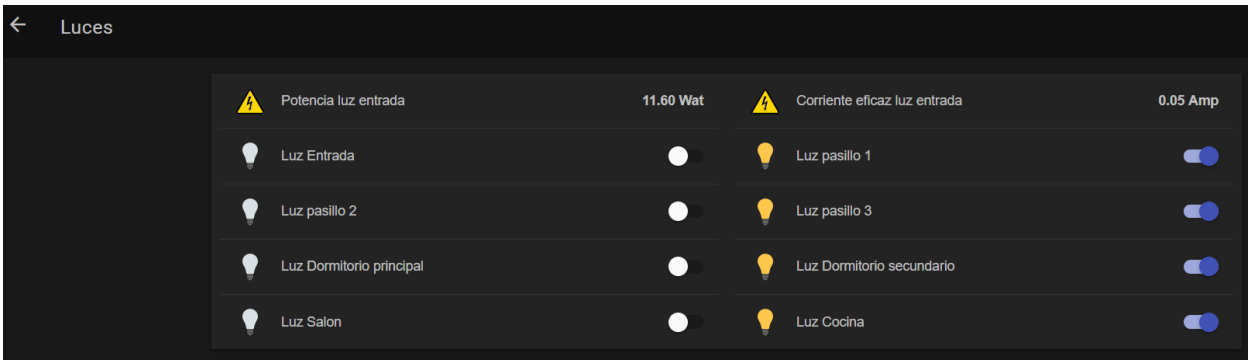


Figura 5.61 - Interfaz de usuario menú "Luces"

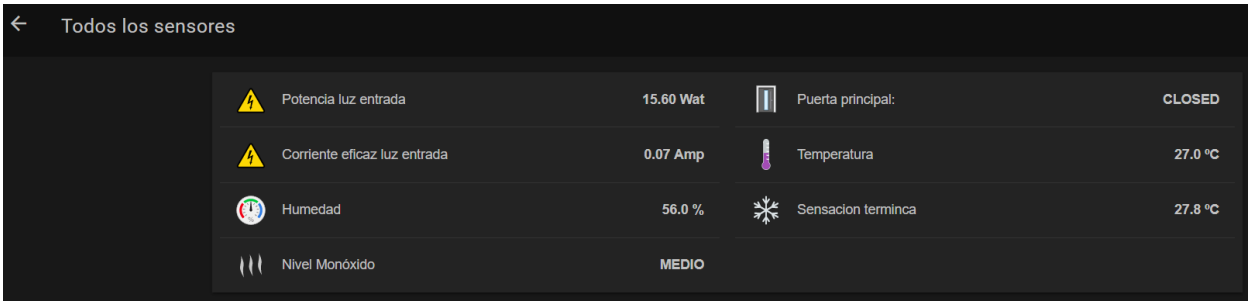


Figura 5.62 - Interfaz de usuario menú "Todos los sensores"

6 CONCLUSIONES Y LÍNEAS DE FUTURO

Las personas siempre han buscado la sencillez y reducir esfuerzos a la hora de realizar tareas. Debido a estos, y otros fines, surgió la automatización que, unido a la ambición del ser humano de controlar su entorno, ha dado lugar a los dispositivos IoT.

Cada día parece más claro que los dispositivos y plataformas IoT no son una tecnología pasajera, han llegado para quedarse; muestra de ello son las cifras de volumen de negocio. En el año 2021 se estima el volumen de negocio generado en 520 billones de dólares, frente a los 235 generados en el año 2017, según la revista Forbes (Forbes, IoT Market Predicted).

En el presente documento se han tratado dos protocolos esenciales en la comunicación entre dispositivos IoT, el protocolo I2C y el protocolo MQTT, dónde este último ha ido ganando más protagonismo en este ámbito desde su creación en IBM hasta llegar a convertirse en un estándar abierto de OASIS y, finalmente, en un estándar ISO.

Este proyecto se ha realizado con la firme idea de adentrarse en el mundo de la domótica, evitando ciertos mitos económicos que recaen sobre esta tecnología. De esta forma, se ha diseñado y desarrollado un sistema domótico, de bajo coste, con el que ha sido posible controlar las luces de la vivienda, monitorizar los niveles de temperatura, humedad y monóxido de carbono además del consumo eléctrico de un dispositivo. Asimismo, se han incorporado elementos de seguridad tales como videovigilancia y sensores de apertura de puertas y ventanas.

Se han usado un número limitado de sensores y actuadores Arduino, esto se ha hecho con el único fin de ejemplarizar un sistema domótico a escala reducida; pero, con la configuración planteada y desarrollada en este proyecto, se podría aumentar perfectamente el número de grupos Arduino - nodeMCU y los sensores utilizados. La gran ventaja de haber utilizado Arduino para la automatización de la vivienda es la enorme cantidad de sensores existentes en el mercado para las plataformas Arduino. Esta cantidad prácticamente ilimitada de sensores y actuadores proporcionan al usuario una total personalización de su sistema domótico y, además, sin invertir una gran cantidad de recursos monetarios.

Como se ha visto en el desarrollo de este documento, el límite del sistema domótico lo marca el administrador del mismo; pero, con la configuración aquí realizada, el ámbito de este sistema queda circunscrito dentro de la LAN.

Uno de los esfuerzos de openHAB, ha sido que la administración de los datos proporcionados por los sensores y enviados al sistema recaiga en el usuario que administra el sistema, no en servicios y empresas externas que puedan poner en riesgo la confidencialidad de dichos datos. Por esta razón, a los creadores de openHAB siempre les ha gustado decir que su servicio es un servicio intranet. Sin embargo, la posibilidad de compartir los datos fuera de la red de área local permite un enorme abanico de posibilidades y, por esta razón, se desarrolló openCloud.

openCloud es un servicio en la nube desarrollado por los creadores de openHAB que permite sincronizar los datos con el servicio intranet openHAB. De esta forma, configurando una cuenta en openCloud y sincronizándola con openHAB, podremos acceder a los datos de nuestro servicio openHAB desde cualquier parte.

openHAB no es sólo un servicio que recoge datos de dispositivos, también es un integrador de soluciones domóticas. Esto nos permite ampliar aún más nuestro sistema domótico, pudiendo integrar dispositivos de

Google, Amazon o Xiaomi destinados a la domótica y ser controlados directamente desde un único servicio.

Por último, se considera interesante dotar al servicio de openHAB de la posibilidad de realizar acciones mediante comandos de voz. Esta funcionalidad permitiría gestionar los dispositivos integrados en openHAB sin necesidad alguna de navegar por la interfaz, tal y como funcionan los dispositivos Google Home y Alexa desarrollados por Google y Amazon respectivamente.

Realizar este proyecto ha resultado una labor más que interesante que, además, me ha permitido afianzar y profundizar en los conocimientos adquiridos durante estos años de aprendizaje. Durante el tiempo que ha abarcado la elaboración del sistema domótico recogido en este documento, he podido comprobar el enorme futuro que tienen por delante tanto los dispositivos IoT como la domótica.

REFERENCIAS

[1] CEDOM, Asociación Española de Domótica e Inmótica. CEDOM: Qué es Domótica. http://www.cedom.es/sobre-domotica/que-es-domotica

[2] openHAB, open Home Automation Bus. Documentation. https://www.openhab.org/docs/

[3] openHAB, open Home Automation Bus. https://www.openhab.org/

[4] Arduino, Guide: Introduction. https://www.arduino.cc/en/Guide/Introduction

[5] OASIS, MQTT. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[6] NXP, I2C-Bus Specification. https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[7] Atmel, ATmega2560/V. https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[8] Espressif Systems, ESP8266 datasheet. https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

[9] EINSTRONIC, nodemCU. https://einstronic.com/wp-content/uploads/2017/06/NodeMCU-ESP8266-ESP-12E-Catalogue.pdf

[10] Raspberry Pi, Raspberry Pi 3 Datasheet. https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM_3p0.pdf

[11] OASIS, MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[12] SONGLE RELAY. http://www.circuitbasics.com/wp-content/uploads/2015/11/SRD-05VDC-SL-C-Datasheet.pdf

[13] Adafruit, DHT22 sensor datasheet. https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

[14] Winsensor, MQ-7 datasheet. https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MQ-7%20Ver1.3%20-%20Manual.pdf

[15] Synacorp, MC38 datasheet. http://blogmasterwalkershop.com.br/arquivos/datasheet/Datasheet%20MC-38.pdf

[16] Allegro MicroSystems, ACS712. https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf

[17] Arduino, Arduino IDE. https://www.arduino.cc/en/main/software

[18] Marginally Clever, nodeMCU setup. https://www.marginallyclever.com/2017/02/setup-nodemcu-drivers-

arduino-ide/

[19] openHAB, icons. https://www.openhab.org/docs/configuration/iconsets/classic/

ÍNDICE DE CONCEPTOS

GLOSARIO

IoT: Internet of Things xi
openHAB: Open Home Automation Bus 2
HTTP: Hypertext Transfer Protocol 3
OSGi: Open Services Gateway initiative 3
S.O: Sistema operativo 4
OSM: OpenStreetMap 7
URL: Uniform Resource Locator 7
DIP: Dual-Inline-Package 9
PWM: Pulse-Width Modulation 9
SMD: Surface Mount 10
DC: Direct Current 10
SRAM: Static Random Access Memory 10
EEPROM: Electrically Erasable Programmable Read-Only Memory 10
LED: Light-Emitting Diode 10
SPI: Serial Peripheral Interface 11
UART: Universal Asynchronous Receiver/Transmitter 11
SS: Slave Selected 11
MOSI: Master Out Slave In 11
MISO: Master In Slave Out 11
SCK: Serial Clock 11
I2C: Inter-Integrated Circuit Communication 12
SDA: Serial Data 12
SCL: Serial Clock 12
ROM: Read-Only Memory 12
AC: Alternating Current 16
CO: Monóxido de Carbono 17
SnO2: Dióxido de Estaño 17
SoC: System on Chip 19
USB: Universal Serial Bus 19
MQTT: Message Queuing Telemetry Transport 23
MSB: Most Significant Byte 26
OSI: Open Systems Interconnection 26

ANEXO A

En este anexo se adjunta el código correspondiente al grupo 1: Arduino MEGA - nodeMCU

Código placa Arduino MEGA

```
#include <Wire.h>
#include "DHT.h"

const int interval = 5000;
unsigned long millisPrevio=0;

int cont_tx=0;

boolean tablaD[][2]= //Tabla pines digitales. primero si HIGH=true, segundo
si output=true
{
  {false, true}, //Luz pasillo 3 = D2.
  {false, true}, //Luz pasillo 2 = D3.
  {false, true}, //Luz pasillo 1 = D4.
  {false, true}, //Luz cocina = D5.
  {false, true}, //Luz salon = D6.
  {false, true}, //Luz dormitorio secundario = D7.
  {false, true}, //Luz dormitorio principal = D8.
  {false, false} //Sensor de temperatura DHT22 (3 datos)= D9
};

int asociacionD[]={2, 3, 4, 5, 6, 7, 8, 9};

String valoresIN_D[] =
{
  "D091-9999", //Temperatura. Pin D9, dato 1
  "D092-9999", //Humedad. Pin D9, dato 2
  "D093-9999" //Sensacion termica. Pin D9, dato 3
};

int asociacionA[]={0}; //Si usamos sólo analog pins con analogRead, podemos
poner 0,1... en vez de A0...
String valoresIN_A[]=
{
  "A000-9999" //Monoxido
};

DHT dht(asociacionD[7], DHT22); //DHTPIN, DHTTYPE

void setup() {
  Wire.begin(2);
  dht.begin();
  Wire.onReceive(cambiarValor); //Registramos la funcion a ejecutar cuando
recivamos un evento
  Wire.onRequest(solicitanDatos); //Funcion a ejecutar cuando el maestro
nos solicite datos
  configPinout();
}

void loop() {
```

```
  if((unsigned long)(millis() - millisPrevio) >= interval){
    millisPrevio=millis();
    //------------------ DHT22 ----------------------
    float hum = dht.readHumidity();
    float temp = dht.readTemperature();
    float sen = dht.computeHeatIndex(temp, hum, false);
    int monox = analogRead(asociacionA[0]);


    valoresIN_D[0]= "+D091-" + String (temp,1);  //Esto da 5 caracteres:
42.50. Poniendo ",1", cogemos solo un decimal
    valoresIN_D[1] = "+D092-"+ String (hum,1);
    valoresIN_D[2] = "+D093-"+ String (sen,1);

    //------------------- MQ7 ----------------------
    String ceros = "";
    if(monox<10){ //1 dígito
      ceros = "000";
    }
    else if(monox<100){ //2 dígitos
      ceros = "00";
    }
    else if(monox<1000){ //3 digitos
      ceros = "0";
    }
    ceros += String(monox);
    valoresIN_A[0] = "+A000-" + ceros;
  }
}

void cambiarValor(size_t howmany){
  String cadena = "";
  while(Wire.available()){
    cadena += (char)Wire.read();
  }
  int separador = cadena.indexOf('-');
  String pin = "";
  if(cadena[1]=='0') //.toInt() no es capaz de traducir 04 a 4
     pin = cadena.substring(2, separador-1); //Empezamos despues del
caracter D hasta el caracter -
  else
    pin = cadena.substring(1, separador-1); //Empezamos despues del caracter
D hasta el caracter -

  separador = cadena.indexOf('-', separador); //Porque la cadena ahora es
del tipo: XXXX-A-Y. Donde A es el codigo de la placa

  String valor = cadena.substring(separador+3); //Empezamos desde el
caracter siguiente a -

  int aux=pin.toInt();
  switch (cadena.charAt(0)){
    case 'D':
      switch (valor.toInt()){
        case 1:
          digitalWrite(pin.toInt(), HIGH);
          break;
        case 0:
```

```
            digitalWrite(pin.toInt(), LOW);
            break;
        }
        break;
    }
}

void solicitanDatos(){
    int valoresD = sizeof(valoresIN_D)/sizeof(valoresIN_D[0]);
    int valoresA = sizeof(valoresIN_A)/sizeof(valoresIN_A[0]);

    int contador=0;
    while(contador<3){
        if(cont_tx < valoresD){
          Wire.write(valoresIN_D[cont_tx].c_str());
          cont_tx++; //contador de todas las entradas enviadas
          contador++; //Contador de entradas máximas por transmision
        }

       else if((cont_tx-valoresD) < valoresA){
          Wire.write(valoresIN_A[cont_tx-valoresD].c_str());
          cont_tx++;
          contador++;
        }
       else{
          cont_tx=0;
          contador=3;
       }
    }
}

void configPinout(){
    int filasD = sizeof(tablaD)/sizeof(tablaD[0]); //Obtenemos el numero de
filas de la tabla de pines Digitales

    for(int i=0; i<filasD; i++){
      if(tablaD[i][1]==true){
        pinMode(asociacionD[i], OUTPUT);
        if(tablaD[i][0]==true)
          digitalWrite(asociacionD[i], HIGH);
        else
          digitalWrite(asociacionD[i], LOW);
      }
      else{
        pinMode(asociacionD[i], INPUT);
      }
    }
}
```

Código placa nodeMCU:

```
#include <Wire.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

/* Variables para la conexion Wifi */
const char* ssid = "JAZZTEL_5DB3";
const char* password = "96828h8C47868CaF882F";
const char* mqtt_server = "192.168.1.2";
WiFiClient clientWifi;
PubSubClient clientMQTT(clientWifi);

/* Variables para el intervalo de peticion de datos y envio a openHAB*/
const int interval = 7000;
unsigned long millisPrevio=0;
int valoresIN = 4; //Necesario indicar cuantos bytes tiene que recibir


void setup() {
  Wire.begin(D1,D2); //sda, scl. Comunicacion i2c

  setup_wifi();
  clientMQTT.setServer(mqtt_server, 1883);
  clientMQTT.setCallback(callback);
  reconnectMQTT();
}

void loop() {
  /* Comprobacion de conexion a la red */
  if (WiFi.status() != WL_CONNECTED)
    setup_wifi();

  if (!clientMQTT.connected()) {
    reconnectMQTT();
  }
  clientMQTT.loop();

  /*Peticion de datos si ha pasado el tiempo indicado */
  if((unsigned long)(millis() - millisPrevio) >= interval){
   millisPrevio=millis();

   /*i2c maximo tx 32Bytes */
   int cont_rx =0;  //Cuenta las entradas recibidas totales (cada entrada
ocupa 10B)
   int entradas_pide=0; //Numero de entradas (cada una ocupa 10Bytes) que
se van a pedir en la transmision

   while(cont_rx!=valoresIN){ //Comprobamos si ya hemos pedido todas las
entradas
      if((valoresIN-cont_rx)>=3){ //Si aun nos quedan por pedir mas de 30B
        cont_rx += 3; //Sumamos los 30B que vamos a pedir al contador
total de Bytes recibidos
        entradas_pide = 3; //Indicamos que vamos a pedir 30Bytes
      }
      else{
        entradas_pide = valoresIN-cont_rx; //Indicamos que vamos a pedir
las entradas que faltan.
        cont_rx += valoresIN-cont_rx; //Sumamos las entradas que faltan al
contador de entradas recibidas
```

```
      }
     Wire.requestFrom(2, entradas_pide*10); //Pedimos al esclavo numero
dos que nos mande los Bytes indicados
     String datos = "";
     while(Wire.available()){ //Recibimos los bytes
        datos += (char)Wire.read(); //Leemos bytes. La cadena recibida
sera una concatenacion
                                    //de la cadena: +XXXX-YYYY, donde
XXXX es el topic e YYYY es el valor del sensor
     }
     int separadorCadena = datos.indexOf('+'); //

     while(separadorCadena>=0){ //Mientras que nos falte por leer mas
signos "+"
      int separadorTopic = datos.indexOf('-', separadorCadena);
//Buscamos el siguiente signo "-"
      String topic = datos.substring(separadorCadena+1, separadorTopic);
//Primero es inclusive, segundo no.

//Nos saltamos el primer elemento (D o A) hasta el signo "-" - 1.
      topic+="-1"; //Añadimos al topic el codigo de la placa que envia
los datos.
      separadorCadena = datos.indexOf('+', separadorTopic); //Buscamos el
siguiente signo "+" para tomarlo como indice referencia
      String valor = "";
      if(separadorCadena>=0){ //Si queda alguna cadena mas por leer...
        valor = datos.substring(separadorTopic+1, separadorCadena);
//Leemos a partir del segundo elemento (nos saltamos el primero (D o A)
                                                                  //has
ta el separador de cadena -1
      }
      else{ //Si no quedan mas cadenas por leer...
        valor = datos.substring(separadorTopic+1); //Leemos hasta el
final.
      }
      clientMQTT.publish(topic.c_str(), valor.c_str()); //Publicamos
directamente.
     }
    }
 }
}

void topicsSuscritos(){//El -1 del final es el codigo de la placa. Cada
una tendrá un codigo diferente.
  clientMQTT.subscribe("D080-1"); //luz dormitorio principal
  clientMQTT.subscribe("D070-1"); //luz dormitorio secundario
  clientMQTT.subscribe("D060-1"); //luz salon
  clientMQTT.subscribe("D050-1"); //luz cocina
  clientMQTT.subscribe("D040-1"); //luz pasillo 1
  clientMQTT.subscribe("D030-1"); //luz pasillo 2
  clientMQTT.subscribe("D020-1"); //luz pasillo 3
}

void setup_wifi() {
  // Nos conectamos a la red Wifi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```
  }
}

void reconnectMQTT() {
  // Bucle hasta que reconecta
  while (!clientMQTT.connected()) {
    if (clientMQTT.connect("placa1")) {
      topicsSuscritos(); // Una vez conectado, volvemos a suscribirnos
    }
    else {
      // Esperamos 5 segundos para volver a conectar
      delay(5000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  if (topic[0]=='D') {//actuamos sobre pines digitales: D/pinout-valor
    String valor="";
    valor +=topic;
    valor+='-';
    valor+=(char)payload[0];//Ej: D/1-1 (Poner en HIGH el pin digital numero
1
    Wire.beginTransmission(2);  // inicia una transmicion al dispositivo #2
    Wire.write(valor.c_str()); //Mandamos valor
    Wire.endTransmission(); //Terminamos transmision
  }
  else {//pin analogico. No tenemos ninguna salida analogica
  }
}
```

ANEXO B

En este anexo se adjunta el código correspondiente al grupo 2: Arduino UNO - nodeMCU

Código placa Arduino UNO:

```
#include <Wire.h>
const int interval = 3000; //Intervalo muestreo sensor ASC217
unsigned long millisPrevio=0;
//Definimos variables para el sensor ASC217
float sensibilidad = 0.100;
float ruido = 0.000;
const int sensorIntensidad = A1;
float valorReposo = 2.50;
float intensidadPico=0;
float potencia=0;
float tensioneDeRed = 230.0;
int cont_tx=0;

boolean tablaD[][2]= //Tabla pines digitales. primero si HIGH=true,
segundo si output=true
{
  {false, true}, //Luz entrada = D2. A nivel bajo, con los cables como
están estaría apagada.
};

int asociacionD[]={2};
String valoresIN_D[] =
{

};


int asociacionA[]={0}; //Si usamos sólo analog pins con analogRead,
podemos poner 0,1... en vez de A0...
String valoresIN_A[]=
{
  "+A011-9999", //Corriente sensor ASC217 - Entrada
  "+A012-9999" //Potencia sensor ASC217 - Entrada
};
void setup() {
  Wire.begin(2);
  Wire.onReceive(cambiarValor); //Registramos la funcion a ejecutar cuando
recivamos un evento
  Wire.onRequest(solicitanDatos); //Funcion a ejecutar cuando el maestro
nos solicite datos
  configPinout();
}
void loop() {
  if((unsigned long)(millis() - millisPrevio) >= interval){
    millisPrevio=millis();
    leerCorriente();
    String aux = "";
    aux=intensidadPico*0.707; //Irms = intensidad pico*0.707
    valoresIN_A[0]="+A011-"+aux.substring(0,4);
    aux=potencia;
```

```
    valoresIN_A[1]="+A012-"+aux.substring(0,4);
  }
}
void cambiarValor(size_t howmany){
  String cadena = "";
  while(Wire.available()){
    cadena += (char)Wire.read();
  }
  int separador = cadena.indexOf('-');
  String pin = "";
  if(cadena[1]=='0') //.toInt() no es capaz de traducir 04 a 4
     pin = cadena.substring(2, separador-1); //Empezamos despues del: D
hasta el caracter -
  else
    pin = cadena.substring(1, separador-1); //Empezamos despues del: D
hasta el caracter -

  separador = cadena.indexOf('-', separador); //Porque la cadena ahora es
del tipo: XXXX-A-Y. Donde A es el codigo de la placa

  String valor = cadena.substring(separador+3); //Empezamos desde el
caracter siguiente a -
  int aux=pin.toInt();
  switch (cadena.charAt(0)){
    case 'D':
      switch (valor.toInt()){
        case 1:
          digitalWrite(pin.toInt(), HIGH);
          break;
        case 0:
          digitalWrite(pin.toInt(), LOW);
          break;
      }
      break;
  }
}


void solicitanDatos(){
  int valoresD = sizeof(valoresIN_D)/sizeof(valoresIN_D[0]);
  int valoresA = sizeof(valoresIN_A)/sizeof(valoresIN_A[0]);
  int contador=0;
  while(contador<3){
      if(cont_tx < valoresD){
        Wire.write(valoresIN_D[cont_tx].c_str());
        cont_tx++; //contador con todas las entradas enviadas
        contador++; //Contador de entradas máximas por transmision
      }

     else if((cont_tx-valoresD) < valoresA){
        Wire.write(valoresIN_A[cont_tx-valoresD].c_str());
        cont_tx++;
        contador++;
      }
     else{
        cont_tx=0;
        contador=3;
     }
```

```
  }
}


void configPinout(){
  int filasD = sizeof(tablaD)/sizeof(tablaD[0]); //Obtenemos el numero de
filas de la tabla de pines Digitales

  for(int i=0; i<filasD; i++){
    if(tablaD[i][1]==true){
      pinMode(asociacionD[i], OUTPUT);
      if(tablaD[i][0]==true)
        digitalWrite(asociacionD[i], HIGH);
      else
        digitalWrite(asociacionD[i], LOW);
    }
    else{
      pinMode(asociacionD[i], INPUT);
    }
  }
}
void resetearValores(){
  valoresIN_A[0] = "+A001-";
  valoresIN_A[1] = "+A002-";
}
void leerCorriente(){
  float valorVoltajeSensor;
  float corriente=0;
  long tiempo=millis();
  float intensidadMaxima=0;
  float intensidadMinima=0;
  while(millis()-tiempo<500){//realizamos mediciones durante 0.5 segundos
    valorVoltajeSensor = analogRead(sensorIntensidad)*(5.0 /
1023.0);//lectura del sensor en voltios
    corriente=0.9*corriente+0.1*((valorVoltajeSensor-
valorReposo)/sensibilidad);
    if(corriente>intensidadMaxima){
      intensidadMaxima=corriente;
    }
    if(corriente<intensidadMinima){
      intensidadMinima=corriente;
    }
  }
  intensidadPico = (((intensidadMaxima-intensidadMinima)/2)-ruido);
  potencia = intensidadPico*0.707*230.0; //Intensidad RMS = Ipico/(2^1/2)
, P=I*V watts.

}
```

Código placa nodeMCU:

```
#include <Wire.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

/* Variables para la conexion Wifi */
const char* ssid = "JAZZTEL_5DB3";
const char* password = "96828h8C47868CaF882F";
const char* mqtt_server = "192.168.1.2";
WiFiClient clientWifi;
PubSubClient clientMQTT(clientWifi);


const int interval = 7000;
unsigned long millisPrevio=0;

const int interval_Seguridad =3000;
unsigned long millisPrevio_Seguridad=0;
boolean puerta_abierta=false;

int valoresIN = 2; //Necesario indicar cuantos bytes tiene que recibir

void setup() {
  Wire.begin(D1,D2); //sda, scl. Comunicacion i2c
  pinMode(D0,INPUT_PULLUP);
  setup_wifi();
  clientMQTT.setServer(mqtt_server, 1883);
  clientMQTT.setCallback(callback);
  reconnectMQTT();
}

void loop() {
  /* Comprobacion de conexion a la red */
  if (WiFi.status() != WL_CONNECTED)
    setup_wifi();

  if (!clientMQTT.connected()) {
    reconnectMQTT();
  }
  clientMQTT.loop();

  if((unsigned long)(millis() - millisPrevio) >= interval){
   millisPrevio=millis();

      /*i2c maximo tx 32Bytes */
   int cont_rx =0;  //Cuenta las entradas recibidas totales (cada entrada
ocupa 10B)
   int entradas_pide=0; //Numero de entradas (cada una ocupa 10Bytes) que
se van a pedir en la transmision

   while(cont_rx!=valoresIN){ //Comprobamos si ya hemos pedido todas las
entradas
      if((valoresIN-cont_rx)>=3){ //Si aun nos quedan por pedir mas de 30B
        cont_rx += 3; //Sumamos los 30B que vamos a pedir al contador
total de Bytes recibidos
        entradas_pide = 3; //Indicamos que vamos a pedir 30Bytes
      }
      else{
        entradas_pide = valoresIN-cont_rx; //Indicamos que vamos a pedir
```

```
las entradas que faltan.
        cont_rx += valoresIN-cont_rx; //Sumamos las entradas que faltan
al contador de entradas recibidas
      }
     Wire.requestFrom(2, entradas_pide*10); //Pedimos al esclavo numero
dos que nos mande los Bytes indicados
     String datos = "";
     while(Wire.available()){ //Recibimos los bytes
        datos += (char)Wire.read(); //Leemos bytes. La cadena recibida
sera una concatenacion
                                    //de la cadena: +XXXX-YYYY, donde
XXXX es el topic e YYYY es el valor del sensor
     }
     int separadorCadena = datos.indexOf('+'); //

     while(separadorCadena>=0){ //Mientras que nos falte por leer mas
signos "+"
      int separadorTopic = datos.indexOf('-', separadorCadena);
//Buscamos el siguiente signo "-"
      String topic = datos.substring(separadorCadena+1, separadorTopic);
//Primero es inclusive, segundo no.

//Nos saltamos el primer elemento (D o A) hasta el signo "-" - 1.
      topic+="-2"; //Añadimos al topic el codigo de la placa que envia
los datos.
      separadorCadena = datos.indexOf('+', separadorTopic); //Buscamos el
siguiente signo "+" para tomarlo como indice referencia
      String valor = "";
      if(separadorCadena>=0){ //Si queda alguna cadena mas por leer...
        valor = datos.substring(separadorTopic+1, separadorCadena);
//Leemos a partir del segundo elemento (nos saltamos el primero (D o A)
                                                                  //has
ta el separador de cadena -1
      }
      else{ //Si no quedan mas cadenas por leer...
        valor = datos.substring(separadorTopic+1); //Leemos hasta el
final.
      }
      clientMQTT.publish(topic.c_str(), valor.c_str()); //Publicamos
directamente.
     }
    }
   }

   if((unsigned long)(millis() - millisPrevio_Seguridad) >=
interval_Seguridad){
    //Los topics D0YX donde Y = 0 o 1 y X = 0-9 estan reservados para las
conexiones directas a la placa nodeMCU (seguridad)
    millisPrevio_Seguridad=millis();
    String valor="";
    valor+=digitalRead(D0);
    String topic="D001-2";
    if(valor=="0" && puerta_abierta==true){
      puerta_abierta=false;
      valor="CLOSED";
      clientMQTT.publish(topic.c_str(), valor.c_str());

    }
```

```
    else if (valor=="1" && puerta_abierta==false){
      puerta_abierta=true;
      valor="OPEN";
      clientMQTT.publish(topic.c_str(), valor.c_str());
    }
   }
}

void topicsSuscritos(){
  clientMQTT.subscribe("D020-2"); //Interruptor rele
}

void setup_wifi() {
  // We start by connecting to a WiFi network
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void reconnectMQTT() {
  // Bucle hasta que reconecta
  while (!clientMQTT.connected()) {
    if (clientMQTT.connect("placa2")) {
      topicsSuscritos(); // Una vez conectado, volvemos a suscribirnos
    }
    else {
      // Esperamos 5 segundos para volver a conectar
      delay(5000);
    }
  }
}
void callback(char* topic, byte* payload, unsigned int length) {
  if (topic[0]=='D') {//actuamos sobre pines digitales: D/pinout-valor
  String valor="";
  valor +=topic;
  valor+='-';
  valor+=(char)payload[0];//Ej: D/1-1 (Poner en HIGH el pin digital numero 1
  Wire.beginTransmission(2);  // inicia una transmicion al dispositivo #2
  Wire.write(valor.c_str()); //Mandamos valor
  Wire.endTransmission(); //Terminamos transmision
  }
  else {//pin analogico. No tenemos ninguna salida analogica
  }
}
```

ANEXO C

En este anexo se adjunta el código correspondiente a la administración de openHABian

Código archivo addons.cfg:

```
# The installation package of this openHAB instance
# Note: This is only regarded at the VERY FIRST START of openHAB
# Note: If you want to specify your add-ons yourself through entries
below, set the package to "minimal"
# as otherwise your definition might be in conflict with what the
installation package defines.
#
# Optional. If not set, the dashboard (https://<yourserver>:8080/) will
ask you to choose a package.
#
# Valid options:
#   - minimal  : Installation only with dashboard, but no UIs or other
add-ons. Use this for custom setups.
#   - simple   : Setup for using openHAB purely through UIs - you need to
expect MANY constraints in functionality!
#   - standard : Default setup for normal users, best for textual setup
#   - expert   : Setup for expert users, especially for people migrating
from openHAB 1.x
#   - demo     : A demo setup which includes UIs, a few bindings, config
files etc.
#
# See https://www.openhab.org/docs/configuration/packages.html for a
detailed explanation of these packages.
#
package = expert

# Access Remote Add-on Repository
# Defines whether the remote openHAB add-on repository should be used for
browsing and installing add-ons.
# This not only makes latest snapshots of add-ons available, it is also
required for the installation of
# any legacy 1.x add-on. (default is true)
#
#remote = true

# Include legacy 1.x bindings. If set to true, it also allows the
installation of 1.x bindings for which there is
# already a 2.x version available (requires remote repo access, see
above). (default is false)
#
legacy = true

# A comma-separated list of bindings to install (e.g. "binding =
sonos,knx,zwave")
binding = mqtt1, network

# A comma-separated list of UIs to install (e.g. "ui = basic,paper")
#ui =

# A comma-separated list of persistence services to install (e.g.
```

```
"persistence = rrd4j,jpa")
#persistence =

# A comma-separated list of actions to install (e.g. "action =
mail,pushover")
#action =

# A comma-separated list of transformation services to install (e.g.
"transformation = map,jsonpath")
#transformation =

# A comma-separated list of voice services to install (e.g. "voice =
marytts,freetts")
#voice =

# A comma-separated list of miscellaneous services to install (e.g. "misc
= myopenhab")
#misc =
```

Código archivo mosquitto.conf (Únicamente se ha incluido la sección configurada):

```
# Config file for mosquitto
#
# See mosquitto.conf(5) for more information.
#
# Default values are shown, uncomment to change.
#
# Use the # character to indicate a comment, but only if it is the
# very first character on the line.
# =================================================================
# Default listener
# =================================================================

# IP address/hostname to bind the default listener to. If not
# given, the default listener will not be bound to a specific
# address and so will be accessible to all network interfaces.
# bind_address ip-address/host name
#bind_address

# Port to use for the default listener.
port 1883

# The maximum number of client connections to allow. This is
# a per listener setting.
# Default is -1, which means unlimited connections.
# Note that other process limits mean that unlimited connections
# are not really possible. Typically the default maximum number of
# connections possible is around 1024.
#max_connections -1

# Choose the protocol to use when listening.
# This can be either mqtt or websockets.
# Websockets support is currently disabled by default at compile time.
# Certificate based TLS may be used with websockets, except that
# only the cafile, certfile, keyfile and ciphers options are supported.
protocol mqtt

# When a listener is using the websockets protocol, it is possible to
serve
# http data as well. Set http_dir to a directory which contains the files
you
# wish to serve. If this option is not specified, then no normal http
# connections will be possible.
#http_dir

# Set use_username_as_clientid to true to replace the clientid that a
client
# connected with with its username. This allows authentication to be tied
to
# the clientid, which means that it is possible to prevent one client
# disconnecting another by using the same clientid.
# If a client connects with no username it will be disconnected as not
# authorised when this option is set to true.
# Do not use in conjunction with clientid_prefixes.
# See also use_identity_as_username.

#use_username_as_clientid
```

Código del archivo mqtt.cfg

```
#
 Define your MQTT broker connections here for use in the MQTT Binding or
MQTT
# Persistence bundles. Replace <broker> with an ID you choose.
#

# URL to the MQTT broker, e.g. tcp://localhost:1883 or
ssl://localhost:8883
mosquitto.url=tcp://192.168.1.2:1883

# Optional. Client id (max 23 chars) to use when connecting to the broker.
# If not provided a random default is generated.
#<broker>.clientId=<clientId>

# Optional. True or false. If set to true, allows the use of clientId
values
# up to 65535 characters long. Defaults to false.
# NOTE: clientId values longer than 23 characters may not be supported by
all
# MQTT servers. Check the server documentation.
#<broker>.allowLongerClientIds=false

# Optional. User id to authenticate with the broker.
#<broker>.user=<user>

# Optional. Password to authenticate with the broker.
#<broker>.pwd=<password>

# Optional. Set the quality of service level for sending messages to this
broker.
# Possible values are 0 (Deliver at most once),1 (Deliver at least once)
or 2
# (Deliver exactly once). Defaults to 0.
#<broker>.qos=<qos>

# Optional. True or false. Defines if the broker should retain the
messages sent to
# it. Defaults to false.
#<broker>.retain=<retain>

# Optional. True or false. Defines if messages are published
asynchronously or
# synchronously. Defaults to true.
#<broker>.async=<async>

# Optional. Defines the last will and testament that is sent when this
client goes offline
# Format: topic:message:qos:retained <br/>
#<broker>.lwt=<last will definition>
```

Código archivo placa1.items

```
Group Pasillo "Pasillo" <corridor>
Group Cocina "Cocina" <kitchen>
Group Luces "Luces" <lightbulb>
Group Salon "Salon" <corridor>
Group Dormitorios "Dormitorios" <bedroom>
Group Temperatura "Temperatura" <temperature> (Salon)
Group Calidad_aire "Calidad del aire" <temperature> (Cocina)
Group Sensores "Todos los sensores" <temperature>

Switch Pasillo_luz1 "Luz pasillo 1" <light> (Pasillo, Luces)
{mqtt=">[mosquitto:D040-1:command:ON:1],>[mosquitto:D040-1:command:OFF:0]"
}
Switch Pasillo_luz2 "Luz pasillo 2" <light> (Pasillo, Luces)
{mqtt=">[mosquitto:D030-1:command:ON:1],>[mosquitto:D030-1:command:OFF:0]"
}
Switch Pasillo_luz3 "Luz pasillo 3" <light> (Pasillo, Luces)
{mqtt=">[mosquitto:D020-1:command:ON:1],>[mosquitto:D020-1:command:OFF:0]"
}

Switch Dorm_prin_luz "Luz Dormitorio principal" <light> (Dormitorios,
Luces) {mqtt=">[mosquitto:D080-1:command:ON:1],>[mosquitto:D080-
1:command:OFF:0]"}
Switch Dorm_sec_luz "Luz Dormitorio secundario" <light> (Dormitorios,
Luces) {mqtt=">[mosquitto:D070-1:command:ON:1],>[mosquitto:D070-
1:command:OFF:0]" }
Switch Salon_luz "Luz Salon" <light> (Salon, Luces)
{mqtt=">[mosquitto:D060-1:command:ON:1],>[mosquitto:D060-1:command:OFF:0]"
}
Switch Cocina_luz "Luz Cocina" <light> (Cocina, Luces)
{mqtt=">[mosquitto:D050-1:command:ON:1],>[mosquitto:D050-1:command:OFF:0]"
}

Number Salon_temp "Temperatura [%.1f °C]" <temperature> (Temperatura,
Sensores) {mqtt="<[mosquitto:D091-1:state:default]" }
Number Salon_hum "Humedad [%.1f %%]" <humidity> (Temperatura, Sensores)
{mqtt="<[mosquitto:D092-1:state:default]" }
Number Salon_sensacion "Sensacion terminca [%.1f °C]" <snow> (Temperatura,
Sensores) {mqtt="<[mosquitto:D093-1:state:default]" }
Number Cocina_monoxido "Monoxido de carbono [%.1f %%]" <smoke>
{mqtt="<[mosquitto:A000-1:state:default]" }


String Cocina_nivel_monoxido "Nivel Monóxido [%s]" <smoke> (Calidad_aire,
Sensores)
```

Código archivo placa2.items:

```
Number CorrienteEntrada "Corriente eficaz luz entrada [%.2f Amp]" <energy>
(Luces, Sensores) {mqtt="<[mosquitto:A011-2:state:default]" }
Number PotenciaEntrada "Potencia luz entrada [%.2f Wat]" <energy> (Luces,
Sensores) {mqtt="<[mosquitto:A012-2:state:default]" }

Switch LuzEntrada "Luz Entrada" <light> (Luces) {mqtt=">[mosquitto:D020-
2:command:ON:1],>[mosquitto:D020-2:command:OFF:0]" }

Contact SensorPuerta "Puerta principal: [%s]" <frontdoor> (Sensores)
{mqtt="<[mosquitto:D001-2:state:default]"}
```

Código archivo casa.sitemap

```
sitemap casa label="Casa"
{
    Frame label="Sensores y Actuadores"{
        Frame label="Acutadores"{
            Group item=Luces
        }

        Frame label="Sensores"{
            Group item=Sensores
            Group item=Temperatura
            Group item=Calidad_aire
            Text item=SensorPuerta icon="door"

        }
    }

    Frame label="Habitaciones"{
        Group item=Pasillo
        Group item=Dormitorios
        Group item=Cocina
        Group item=Salon
    }
    Frame label="Camara"{

        Image
url="http://192.168.1.129:8282/video?user=camara&pwd=seguridad"
label="Camara principal" refresh=6000
    }

}
```

Código archivo reglas.rules

```
rule "Monoxido Carbono"

when
    Item Cocina_monoxido received update
then
    if(Cocina_monoxido.state <=150){
        Cocina_nivel_monoxido.postUpdate("BAJO")
    }
    else if(Cocina_monoxido.state > 150 && Cocina_monoxido.state < 1000){
        Cocina_nivel_monoxido.postUpdate("MEDIO")
    }
    else{
        Cocina_nivel_monoxido.postUpdate("ALTO")
    }
end
```