UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Diseño e implementación de un dispositivo IoT de bajo coste para la medida de la calidad del aire

TRABAJO FIN DE GRADO

Junio - 2023

AUTORES: Diego Nescolarde López

DIRECTOR/ES: Eduardo Yubero Funes

Nuria Galindo Corral

AGRADECIMIENTOS

A mis tutores tanto a Eduardo Yubero como a Nuria Galindo. Los dos me han dado la oportunidad de realizar este proyecto. A Eduardo que me ha proporcionado su total implicación y dedicación, así como la atención prestada siempre que la he necesitado. A Nuria que sin su ayuda no podría haber diseñado y fabricado ninguno de los prototipos.

A mi familia por estar siempre ahí con su apoyo y ayuda incondicional. A mi padre por ser un ejemplo de motivación y trabajo constante. A mi madre por siempre motivarme a estudiar y confiar en mí aun cuando pasaba por peores momentos. A mi hermana que, a pesar de nuestras diferencias, siempre he podido contar con ella.

A mi pareja por estar a mi lado, ayudarme siempre que lo he necesitado y apoyarme tanto en los buenos momentos como en los malos.

A mis amigos que han sido siempre un pilar fundamental para mí durante todo este tiempo.

A mis compañeros de clase que me han acompañado y ayudado estos años. Por todas las horas en la sala de estudio, por las risas y momentos en la cafetería.

A todos ellos, gracias.

RESUMEN

En el siguiente proyecto se ha diseñado un dispositivo que permite obtener en tiempo real la concentración de partículas atmosféricas, la concentración de dióxido de carbono (CO_2) y algunos parámetros meteorológicos como la temperatura, humedad y presión. Para ello, se ha realizado primero, un estudio de mercado que ha permitido escoger los mejores sensores disponibles en función de su relación calidad/precio. Todos estos sensores están conectados a un microcontrolador con conexión wifi, lo que permite almacenar los datos en la nube en tiempo real. También, se ha realizado un prototipo, mediante una PCB, que incluye una tarjeta microSD y un sistema RTC para hacer un guardado continuo de los datos. Además, será posible visualizar las medidas en una pantalla y mediante el uso de botones será posible modificar la configuración de éste. Por último, se ha diseñado e impreso una carcasa, con una impresora 3D, para finalizar el prototipo.

PALABRAS CLAVE: IoT, aerosoles, contaminación, bajo coste, prototipo.

ABSTRACT

A device will be designed to obtain in real time the concentration of atmospheric particles, CO_2, temperature, humidity and pressure. To this end, the best sensors available on the market will be chosen according to their quality/price ratio. All these sensors will be connected to a microcontroller with wifi connection, which will allow the data to be stored in the cloud in real time. A prototype will also be made, using a PCB, which will include a microSD card and an RTC system to store the data. In addition, it will be possible to visualise the measurements from a screen and with some buttons it will be possible to modify the configuration of this. Finally, a case will be designed and printed with a 3D printer to finalise the prototype.

KEYWORDS: IoT, aerosol, low-cost, prototype.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

ABREVIATURAS

µC – Microcontrolador

BOM – *Bill of materials*

CO_2 – Dióxido de carbono

CS – *Chip Select*

ESD – *Electrostatic discharges*

I2C – *Inter-Integrated Circuit*

IoT – *Internet of Things*

MAE - *Mean Absolute Error*

MISO – *Master Input Slave Input*

MOSI – *Master Output Slave Input*

NTP – *Network Time Protocol*

NDIR - Infrarrojo no dispersivo

PA – Pascales

PCB – *Printed Circuit Board*

PM – *Particulate Matter*

PWM – *Pulse Width Modulation*

RMSE -*Root Mean Squared Error*

RH – Humedad relativa

RTC – *Real Time Clock*

RX – *Receiver Bus*

SCL – *System Clock*

SCLK – *System Clock*

SDA – *System Data*

SMD – *Surface-Mount Device*

SoC – *System on Chip*

SPI – *Serial Peripheral Interface*

THT - *Through-Hole Technology*

TTL – *Transistor-transistor logic*

TVS – *Transient Voltage Suppressor*

TX – *Transmisor Bus*

UART – *Universal Asynchronous Receiver-Transmitter*

ESQUEMA TEMÁTICO

En este apartado se va a analizar el proyecto desde una visión global, para entender cuáles van a ser los contenidos que se van a encontrar en cada apartado de la memoria.

Esta memoria comienza con una introducción donde se explica la necesidad de este proyecto, continua con el estado de arte actual en este ámbito y, por último, se explican los fundamentos teóricos que se consideran relevantes del proyecto.

A continuación, se explica el objetivo del proyecto y la planificación seguida para cumplir dichos objetivos.

Posteriormente, en el apartado de material se pueden encontrar los esquemas tanto, el base como el final, del dispositivo. Se presenta un estudio de mercado que permitió realizar la selección de los sensores, se describen los componentes necesarios y las aplicaciones que se utilizan para llevar a cabo el proyecto.

A continuación, en el apartado de desarrollo del dispositivo, se explican todos los pasos necesarios para programar, diseñar y montar el prototipo. También, hay un apartado con el diseño y fabricación de una carcasa 3D para el dispositivo. Además, se analiza el consumo del dispositivo.

Por otro lado, en el apartado de presupuesto se incluye el coste de fabricación del prototipo.

Por último, se analizan los resultados obtenidos y se obtienen conclusiones del proyecto añadiendo posibles mejoras y líneas futuras.

Al final del proyecto se incluyen anexos con el código del dispositivo y el diseño de la placa. También, se encuentra la bibliografía consultada para la realización del trabajo.

1. INTRODUCCIÓN

Tradicionalmente, la medida de la calidad del aire se ha realizado mediante instrumentación de alto coste económico y de gran tamaño. Esto impedía el acceso público a medidas de contaminantes y dificultaba el poder obtener una representación espacial de la contaminación en, por ejemplo, ciudades debido a la imposibilidad de adquirir numeroso equipamiento.

Actualmente, con el avance de los señores de bajo coste, los microcontroladores y el incremento en el internet de las cosas (IoT) han ido poco a poco apareciendo dispositivos que permiten la medida de contaminantes a un precio mucho más reducido. Sin embargo, los dispositivos disponibles de "precio reducido" en el mercado, siguen representando un coste muy elevado para la gran mayoría de familias. Además, muchos de estos dispositivos de bajo coste no ofrecen medidas suficientemente precisas y fiables de la calidad del aire.

Existen muchos contaminantes distintos que informan de la calidad del aire actual, aunque en este trabajo se va a centrar en dos de ellos: la materia particulada (PM) y el dióxido de carbono (CO_2). La materia particulada, tiene una importancia crítica por su impacto en el clima, la visibilidad, el ciclo biogeoquímico, la reactividad atmosférica y la salud humana. Mientras que el segundo, el CO_2 es uno de los parámetros que indica la saturación del ambiente y por tanto del grado de ventilación en interiores, y además es un indicador sobre el efecto invernadero y su aumento se relaciona con el cambio climático. Además de los contaminantes mencionados, también es interesante la medida de algunas variables meteorológicas como la temperatura, humedad y presión, que aportarán una información extra muy valiosa para la interpretación de las concentraciones de contaminantes.

1.1. ESTADO DEL ARTE

En este apartado, se expone, desarrolla y explica que son y porqué es importante medir el PM y CO_2. Además, se analizan ejemplos de captadores tradicionales de referencia que permiten la medida de estos contaminantes. Por otro lado, se evalúa las alternativas ya disponibles en el mercado de captadores de bajo coste relacionados con el diseñado en este proyecto. Por último, se explica en qué consiste el internet de las cosas (IoT).

1.1.1. MATERIA PARTICULADA (PM)

La materia particulada, abreviado como 'PM' del inglés *Particulate Matter*, es una mezcla de partículas sólidas y gotas minúsculas líquidas suspendidas en el aire que se pueden inhalar y que pueden llegar a provocar serios problemas de salud. El PM incluye a partículas de diferentes tamaños, propiedades ópticas y composición, aunque generalmente se divide en categorías según el tamaño de estas. Estas categorías utilizan la siguiente nomenclatura PMx donde *x* indica el diámetro aerodinámico de las partículas que pasarían a través de un cabezal que recogiera con una efectividad del 50%.

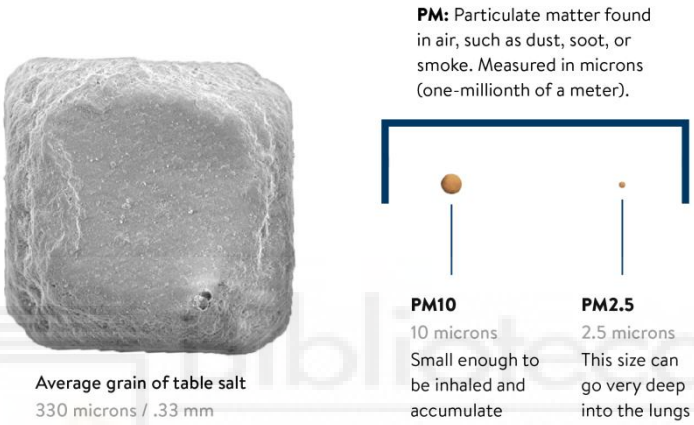


Figura 1 – Rango de tamaño de partículas [1]

En la figura 1 se compara el tamaño de un grano de sal y dos partículas, una de PM10 y otra de PM2.5. Debido al tamaño tan pequeño de estas partículas es posible inhalarlas y que se dirijan hacia los pulmones, lo que puede provocar con el tiempo problemas respiratorios y otros efectos perjudiciales para la salud. Las partículas inferiores en tamaño son incluso capaces de llegar al torrente sanguíneo pudiendo producir problemas cardiovasculares.

La principal fuente de emisión antropogénica, de esta materia particulada suspendida en el aire, son los motores de combustión que están presentes en vehículos de todo tipo, instalaciones industriales, etc. Respecto a las fuentes naturales destacarían las emisiones de aerosol marino y las de transporte de larga distancia de desiertos como el del Sahara.

La Organización Mundial de la Salud (OMS), en el año 2021, endureció las concentraciones límites recomendadas para distintos contaminantes atmosféricos. La concentración máxima promedio anual de partículas en suspensión con un diámetro de 2,5 micras recomendada pasa de ser 25 a 15 $\mu g/m^3$. [2].

Los efectos perjudiciales para la salud asociados con la exposición a partículas son significativos. A modo de ejemplo, un informe del Ministerio de Sanidad de España [3], indica que la contaminación atmosférica provoca más de 3,2 millones de muertes al año, en todo el mundo, y al menos 13.915 muertes anuales en España. A partir de estos datos es evidente la necesidad de la medida, control y reducción de este contaminante.

1.1.2. DIÓXIDO DE CARBONO (CO_2)

El dióxido de carbono, fórmula química CO_2, es un compuesto de carbono y oxígeno. Este compuesto está íntimamente relacionado con el efecto invernadero y, por lo tanto, con la subida de la temperatura terrestre y el cambio climático.

La concentración actual es de alrededor de 420 ppm, partes por millón, muy superior a los 280 ppm que había antes de la industrialización. Este valor está aumentando año tras año debido a la contaminación, como se puede ver en la siguiente figura (Fig. 2):

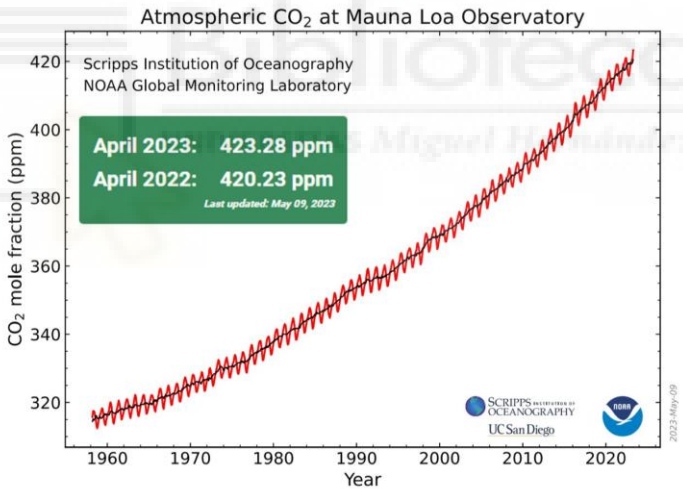


Figura 2 – Aumento del CO_2 atmosférico medio en el observatorio de Mauna Loa en Hawaii [4]

La principal fuente de emisión antropogénica, del dióxido de carbono procede de la combustión del carbón, petróleo y gas de centrales eléctricas, automóviles e instalaciones industriales.

El CO_2, además de ser un indicativo del cambio climático, es muy útil para conocer la ventilación en espacios interiores cerrados y, por lo tanto, para evaluar la probabilidad de contagio por virus o bacterias. Esta información ha sido utilizada ampliamente durante los años de pandemia como indicador de probabilidad de contagiarse por el virus SARS-COV2 (coronavirus). El coronavirus se propaga a través de partículas que se originan a través de la

respiración, tos o habla de una persona contagiada. Mantener niveles de CO_2 bajos permite asegurar una ventilación adecuada y libre de posibles patógenos contagiosos. Concentraciones altas de CO_2, por encima de 800-1000 ppm, nos indicarán de la necesidad de ventilación de la sala.

1.1.3. CAPTADORES TRADICIONALES

Actualmente la infraestructura disponible para la medición de la concentración de partículas en el aire y de CO_2 se realiza en distintas estaciones a nivel regional para poder comprobar si se cumple con las regulaciones de la calidad del aire (Fig. 3). Estas estaciones permiten conocer la situación en un punto concreto que se escoge con la idea de que sea lo más representativo posible del entorno. Sin embargo, los valores de estos contaminantes varían en el espacio y el tiempo con una resolución muy grande, por lo que parte de la información se pierde no pudiendo determinar zonas donde la contaminación es más elevada y, por lo tanto, donde la población podría estar más expuesta. El coste de estas estaciones de medida es extremadamente alto y su tamaño es también importante. Además, requieren de un mantenimiento costoso.

Figura 3 – Estación tradicional de medida de contaminación [5]

Por otro lado, existen captadores tradicionales portátiles los cuales permiten medir uno o varios valores contaminantes ocupando un espacio mucho menor que el de las estaciones y que permitiría su desplazamiento a distintos lugares. Los precios de estos captadores continuos superan, en el caso de partículas atmosféricas, los 25.000 €.

Estos captadores son una referencia en el ámbito de la medida de contaminantes. En este apartado se han seleccionado dos dispositivos del mercado para mostrar un ejemplo de las alternativas disponibles y se van a describir las características de estos.

El primero es un captador de dióxido de carbono llamado Serinus 31 de Acoem (Fig. 4):

Figura 4 – Serinus 31 - Captador tradicional de dióxido de carbono [6]

Este es un captador de referencia en el ámbito de la medida de dióxido de carbono (CO_2). Este dispositivo permite medir el CO_2 mediante el método NDIR, infrarrojo no dispersivo. Cuando la radiación infrarroja interactúa con las moléculas de gas, éstas absorben la luz infrarroja provocando la vibración de las moléculas de gas. Los sensores de gas NDIR (infrarrojos no dispersivos) detectan la disminución de la luz infrarroja transmitida, que es proporcional a la concentración de gas. El Serinus 31 es un captador tradicional con un peso de 21,3 kg y un tamaño considerablemente grande. Necesita ser colocado en un rack y no puede estar al aire libre, obligando la instalación de una cabina protectora.

Las especificaciones del Serinus 31 son (Tab.1):

Serinus 31	
Fabricante	Acoem
Rango efectivo	0 – 3000 ppm
Ruido	< 0,2 ppm o 0,1 % de la concentración
Error por linealidad	< 1% en el fondo de escala
Precisión	0,2 ppm o 0,5% de la medida
Rango de temperaturas	0 a 40°C

Tabla 1 – Especificaciones del Serinus 31

El segundo captador que se quiere destacar permite medir materia particulada y es el modelo GRIMM 1109 (Fig. 5):

Figura 5 – GRIMM 1109 – Captador tradicional de partículas [7]

En este caso, el GRIMM 1109, es un captador de referencia en el ámbito de la medida de la materia partícula (PM). Este dispositivo permite medir la materia particulada mediante el método de dispersión de luz. El aire muestreado se conduce directamente a la celda de medición a través de la entrada de aerosoles u otras entradas de aire diseñadas a medida, por ejemplo, para velocidades de viento elevadas o sobrepresión. Las partículas presentes en el aire de muestra se detectan mediante dispersión de luz en el interior de la célula de medición. Se cuenta el pulso de luz de dispersión de cada partícula y la intensidad de su señal de luz de dispersión se clasifica en función de un determinado tamaño de partícula. Este es un captador tradicional con un peso de 2,4 kg y un tamaño considerablemente grande.

Las especificaciones del GRIMM 1109 son (Tab.2):

GRIMM 1109	
Fabricante	GRIMM
Rango de medidas	0.25 a 32 µm, 31 canales
Concentración de partículas	1 a 2.000.000 partículas/litro
Rango efectivo	0,1 a 100.000 µg/m³
Error (Reproducibilidad)	±3% para todo el rango de medida

Tabla 2 – Especificaciones del GRIMM 1109

Estos dos dispositivos son una referencia en el mercado debido a que tienen una precisión y fiabilidad muy alta. Sin embargo, el precio, tamaño y peso son factores que imposibilitan la recomendación de su compra para ámbitos no profesionales y mucho menos en situaciones en las que queramos tener muchas medidas simultáneas en distintos emplazamientos.

1.1.4. CAPTADORES DE BAJO COSTE

En los últimos años han surgido nuevas tecnologías a precios y tamaños mucho menores que los de los captadores tradicionales. Estos captadores que rondan precios inferiores a los mil euros se les llaman de bajo coste o *low-cost* y permiten tener una resolución espaciotemporal mucho más alta e implementarlos fácilmente a gran escala.

En el mercado actual se dispone de algunos dispositivos que miden la concentración de partículas en el aire y/o el CO_2 con un tamaño reducido y precio medianamente bajo.

Se han seleccionado dos dispositivos del mercado para mostrar un ejemplo de las alternativas disponibles y se van a describir las características de estos. Este mercado está en crecimiento continuo y aparecen continuamente dispositivos capaces de realizar estas medidas.

El primer dispositivo que se va a describir y mostrar sus especificaciones es el Purple Air (Fig. 6).

Figura 6 – Purple Air - Flex Air Quality Monitor [8]

Este dispositivo utiliza dos sensores de partículas para proporcionar redundancia y ayudar a determinar la salud y fiabilidad del sensor. Además de tener estos dos sensores, dispone de uno que mide temperatura, humedad y presión. Tiene un microcontrolador con conexión wifi para subir los datos a internet y cuenta con la posibilidad de almacenar datos en una microSD y un RTC, "Real-Time Clock", que le permite mantener la hora ante apagones o pérdidas de conexión.

Este dispositivo es resistente al agua con posibilidad de estar tanto en interiores como exteriores, se alimenta con cable, no tiene baterías y habría que comprar el cargador aparte.

Su precio es de 269$, unos 250€, un precio que, posiblemente, resulte un poco elevado para usuarios que simplemente quieran ser conscientes de la concentración de partículas en una sala o una habitación de su casa. Solo mide la concentración de material particulado, no dando información de ningún otro contaminante ni del grado de ventilación en interiores. Además, como se verán en el estudio de mercado, el sensor utilizado no es la mejor opción disponible en el mercado.

Las especificaciones del PurpleAir Flex Air Quality Monitor son (Tab. 3):

	Flex Air Quality Monitor	
(2) PMS6003 (Sensor de PM)	Fabricante	Plantower
	Rango de medidas	0.3, 0.5, 1.0, 2.5, 5.0, & 10 µm
	Eficiencia de cuenta	50% para 0.3µm & 98% para ≥ 0.5µm
	Rango efectivo	0 a 500 µg/m³
	Error máximo	10 µg/m³ → 0 a 100 µg/m³ ±10% → 100 a 500 µg/m³
BME688 (Sensor de temperatura, humedad y presión)	Fabricante	Bosch
	Rango de temperaturas	-40 a 85°C
	Rango de presión	300 a 1100 hPa, ± 0,25 %
	Rango de humedad	0% a 100%, ± 3% RH
RTC	Modelo	DS3231
Microcontrolador	Modelo	ESPWROOM02D

Tabla 3 – Especificaciones del Purple Air Flex Air Quality Monitor. Se presentan las características de los sensores que forman el dispositivo en sí [8]

El segundo dispositivo a destacar es el AQ Mesh (Fig. 7).

Figura 7 – Sistema de monitoreo de la calidad del aire AQ Mesh [9]

Es un dispositivo modulable el cual puede medir hasta 6 gases y también PM, ruido, velocidad del viento y su dirección. Además, también se puede elegir el modo de alimentación del dispositivo, se tiene la opción con batería, panel solar o con fuente de alimentación por cable. Para todas las configuraciones disponibles se dispone de además de sensor de temperatura, humedad y presión.

Este dispositivo puede configurarse para medir los siguientes valores (Fig. 8):

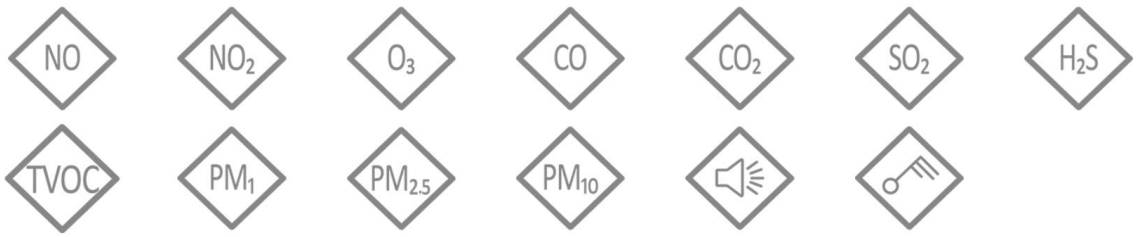


Figura 8 – Posibles medidas del AQ Mesh [9]

Las especificaciones del AQ Mesh son las de la tabla 4:

	AQ Mesh			
	Unidades	Rango	Correlación	Precisión
PM_1	µg/m³	0-100,000 µg/m³	>0.9	5 µg/m³
$PM_{2.5}$	µg/m³	0-150,000 µg/m³	>0.9	5 µg/m³
PM_4	µg/m³	0-225,000 µg/m³	>0.9	5 µg/m³
PM_{10}	µg/m³	0-250,000 µg/m³	>0.85	5 µg/m³
CO_2	ppm	0-5,000 ppm	>0.9	50 ppm
Temperatura	°C or °F	-20 a 100°C	>0.9	2°C
Presión	hPa	500 a 1500 hPa	>0.9	5 hPa
Humedad	%	0 a 100%	>0.9	5% RH

Tabla 4 – Especificaciones del AQ Mesh. La columna de correlación está calculada con respecto a un equipo de referencia [9]

Los datos de correlación de la tabla 4 vienen dados comparando los sensores del dispositivo con un dispositivo de medición de referencia. El fabricante no da ninguna información de los sensores utilizados para la medida de contaminantes.

Para el caso en el que se quiera comprar este dispositivo con una configuración básica (alimentación por cable, medida de PM y CO_2, sin la medida de ningún gas adicional) el coste será de 5.040 €. Como se puede observar el precio de este dispositivo es muy elevado, mucho más que el mostrado anteriormente, Purple Air.

Además de los dos dispositivos anteriormente mencionados, existen otras alternativas en el mercado que dan información sobre la calidad del aire o que miden concentraciones de contaminantes. Aunque para todos estos dispositivos se puede encontrar unos puntos en común: un precio superior a los 250€ y una fiabilidad de medida no siempre asegurada.

1.1.5. INTERNET DE LAS COSAS (IoT)

Gracias a la llegada de los microcontroladores, un circuito integrado que contiene todos los componentes de un ordenador, su bajo coste y la aparición de las comunicaciones de gran ancho de banda han permitido que ahora existan millones de dispositivos conectados a

internet. Esto significa que cada dispositivo del día a día como, por ejemplo, los cepillos de dientes, neveras, aspiradores y coches pueden recopilar información y de esta manera informar a los usuarios y actuar consecuentemente. Podría darse el caso en el que, por ejemplo, nuestra nevera detectase que nos hemos quedado sin leche y que automáticamente añadiese la leche al carrito de la compra.

Internet de las cosas (Internet of Things, IoT) describe la red de objetos físicos que llevan incorporados sensores, software y otras tecnologías con el fin de conectarse e intercambiar datos con otros dispositivos y sistemas a través de Internet.

Aunque el concepto de IoT existe desde hace años, una serie de avances han permitido que se desarrolle vertiginosamente en los últimos años:

- Sensores de bajo coste y bajo consumo: la disponibilidad de sensores asequibles está haciendo posible llevar a cabo muchos proyectos con tecnología IoT.
- Conectividad: los avances tecnológicos en los protocolos de red han facilitado la conexión de sensores a la nube. Algunos ejemplos serían: bluetooth, NFC o WIFI.
- Plataformas en la nube: gracias al aumento de estas plataformas en la nube que permiten acceder a la infraestructura que necesitan escalar sin que el usuario tenga que realizar la gestión completa.
- Aprendizaje automático y análisis: con los avances en el aprendizaje automático y la analítica, junto a la posibilidad de acceder a cantidades enormes de datos almacenados en la nube, las empresas pueden obtener información más rápida y fácilmente.
- Inteligencia artificial conversacional: gracias a la popularidad de las redes neuronales, ha sido posible la incorporación del procesamiento del lenguaje natural (PLN) a dispositivos IoT, como los asistentes personales digitales Alexa, Cortana y Siri, y los han hecho atractivos, asequibles y viables para uso doméstico.

1.2. FUNDAMENTOS TEÓRICOS

En este apartado se explican los conceptos básicos que se consideran importantes mencionar para comprender en profundidad el desarrollo del proyecto y todos sus apartados.

1.2.1. SENSORES LOW-COST

SENSOR LOW-COST PM

La gran mayoría de los sensores de bajo coste que permiten medir los valores de PM en el aire, lo hacen mediante la dispersión de la luz a través de estas partículas. El funcionamiento de estos sensores es similar al de los captadores tradicionales, pero miniaturizando todos los componentes.

Las partículas son normalmente absorbidas mediante un ventilador y se les hace pasar a través de un haz de luz, habitualmente un láser. La dispersión de la luz por las partículas es capturada por un fotodiodo. Posteriormente, el fotodiodo pasa la información al convertidor fotoeléctrico. La señal modulada del convertidor es utilizada para determinar la concentración de partículas (µg/m³).

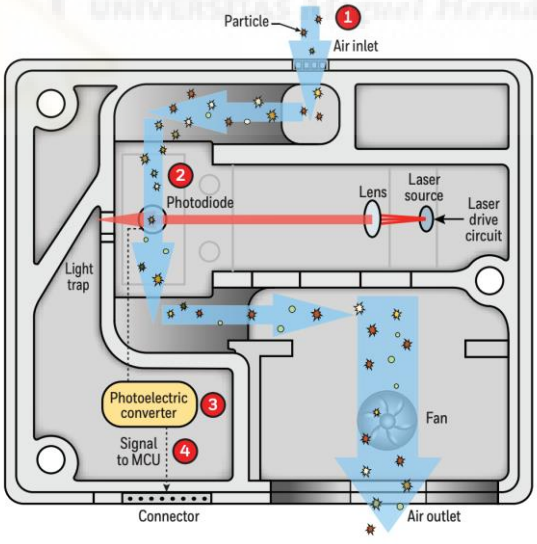


Figura 9 – Funcionamiento interno de un sensor de PM de bajo coste [10]

En la figura 9 se puede observar el interior de un sensor de partículas de bajo coste, las partes que lo conforman para así comprender un poco mejor el funcionamiento de este.

Utilizar el método de dispersión de luz tiene una serie de limitaciones importantes a la hora de realizar las mediciones de la concentración de PM. La gran mayoría de estas limitaciones

aparecen ante el cambio en las condiciones ambientales, estas pueden ser especialmente problemáticas para sensores de gama baja.
Estos sensores suelen presentar problemas sobre todo cuando aumenta la humedad relativa y las partículas absorben cierta cantidad de agua lo que puede cambiar su tamaño y masa, además de su forma, índice de refracción y concentración.

Por lo tanto, es importante ser conscientes de la posible dependencia de las medidas con la temperatura, humedad y presión. Para compensar esta posible desventaja, será importante realizar técnicas de post procesado de los datos incorporando los valores de los sensores meteorológicos para tratar de mejorar la precisión de las medidas cuando se produce una variación de las condiciones ambientales.

SENSOR LOW-COST CO_2

En cuanto a los sensores low-cost de CO_2, el método más utilizado para hacer la medida es mediante NDIR, que significa de infrarrojo no dispersivo.

Los sensores que utilizan el principio NDIR (Fig. 10) para medir CO_2 funcionan usando una lámpara infrarroja (IR) que apunta hacia un sensor de infrarrojos, dentro de una cavidad que tiene uno o más orificios donde circula el gas.

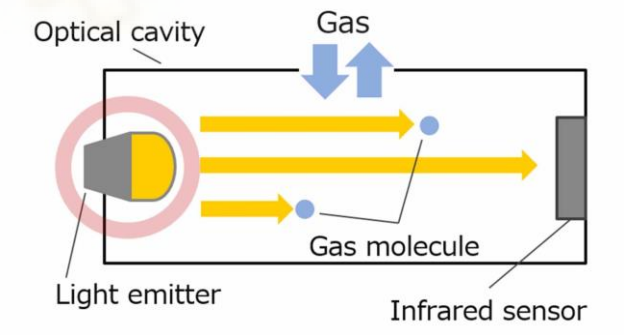


Figura 10 – Funcionamiento de un sensor NDIR [11]

Gracias a que la banda de radiación infrarroja es muy cercana a la banda de absorción del CO_2, es posible identificar la molécula de CO_2.

A medida que la luz infrarroja pasa a través de la cavidad estrecha de aire, las moléculas de gas CO_2 absorben la banda específica de luz IR mientras dejan pasar el resto de las longitudes de onda de luz.

Finalmente, el sensor de infrarrojos detecta la cantidad restante de luz infrarroja que no fue absorbida por las moléculas de CO_2. La cantidad de partes por millón, ppm, se calcula en función de la lectura del sensor de infrarrojos.

Los sensores de infrarrojos normalmente utilizados para sensores low-cost suelen tener una considerable sensibilidad a la temperatura. Por lo tanto, para el caso de los sensores de mayor calidad se suele encontrar su correspondiente compensación.

SENSORES LOW-COST METEOROLÓGICOS

Existen muchos tipos de sensores meteorológicos, aunque en este proyecto se va a centrar solamente en sensores de temperatura, humedad y presión. No obstante, existen sensores para medir la radiación, precipitación, velocidad del viento y dirección, etc.

Para el caso de los sensores de temperatura su funcionamiento consiste en utilizar sensores resistivos. La principal característica de las resistencias que utilizan estos sensores es que son variables en función de la temperatura, lo que permite medirla mediante un circuito de acondicionamiento. En la siguiente figura 11 se puede ver un ejemplo de sensores resistivos:

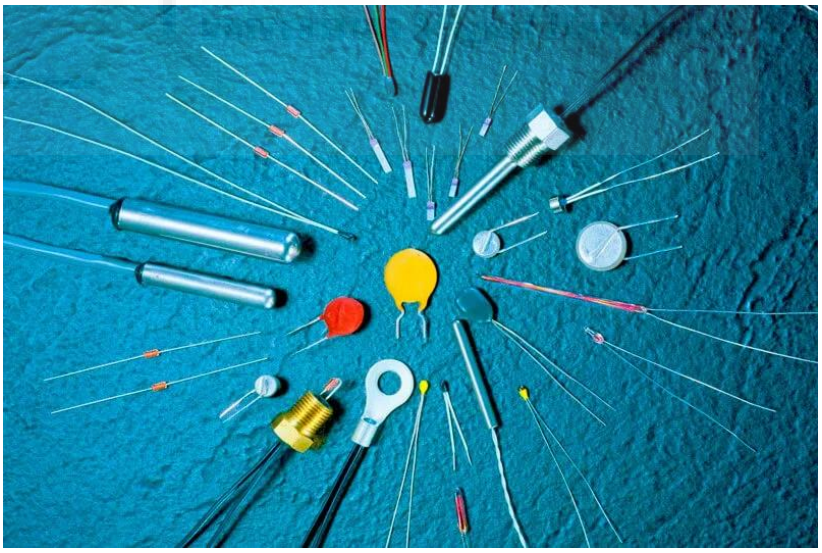


Figura 11 – Sensores de temperatura resistivos [12]

Por otro lado, dentro de los sensores de humedad existen muchos tipos en función del método físico que utilicen para realizar la medida. El sensor de humedad más utilizado es el capacitivo, este tipo de sensor aprovecha la variación en la capacidad cuando se produce cambio en la permitividad eléctrica del material que tenemos entre las placas del

condensador. En la figura 12 de a continuación se pueden ver las distintas partes del condensador:

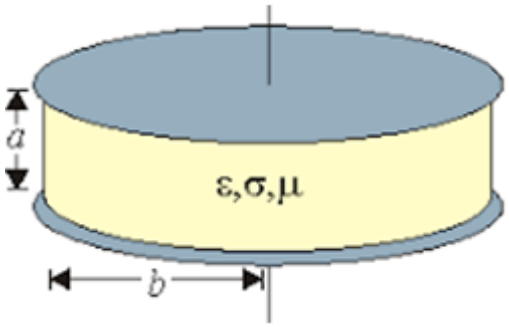


Figura 12 – Sensores de humedad por condensadores variables [13]

Por último, para el caso de los sensores de presión existe una situación parecida a la que se tiene en el caso de los sensores de humedad. Se dispone de muchos tipos de sensores de presión en función de la tecnología que utilizan para medir. Según el tipo de aplicación, rango de medida y precisión necesitada es más común utilizar una tecnología u otra. Aunque los métodos más utilizados son mediante galgas extensiométricas y condensadores variables. En ambos métodos se utiliza una membrana que modifica las propiedades estables cuando existe variaciones en la presión. Para el caso de la galga extensiométrica modifica la resistividad. Mientras que para el caso del condensador variable modifica la capacidad.

En la siguiente (Fig. 13) se observa un sensor de presión por membrana:

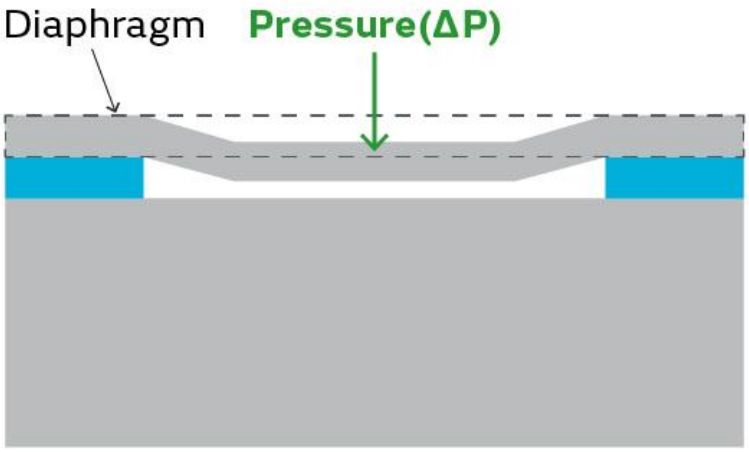


Figura 13 - Sensores de presión por membrana. En este caso en la imagen se muestra un sensor de presión capacitivo por membrana. Este sensor tiene una membrana que se deforma cuando sobre ella se aplica una variación en la presión. Esta microdeformación reduce la distancia entre las dos placas del condensador modificando la capacidad del sensor [14]

1.2.2. MICROCONTROLADORES

Los microcontroladores o, abreviadamente µC, son dispositivos que gracias a la evolución en la microelectrónica han conseguido integrar las funcionalidades de un microprocesador y de múltiples periféricos, en un solo chip.

Desde el primer microcontrolador, alrededor de los 70, se han producido grandes mejoras y estos han evolucionado mucho. Desde entonces el precio se ha visto reducido de manera considerable y se ha visto un incremento en las aplicaciones IoT. Uno de los microcontroladores, o gama de microcontroladores, que más se han utilizado históricamente es el llamado *Peripheral Interface Controller* (PIC). Estos destacaban por su bajo precio, aunque resultaban complejos de utilizar o programar. Más adelante aparecieron los dispositivos Arduino que permitieron acercar el mundo de la electrónica a más usuarios gracias a su sencillez y a la gran comunidad que tiene detrás que ha facilitado elaborar proyectos muy interesantes y de muy diversa índole. Actualmente, existen muchos otros fabricantes y modelos que incorporan microcontroladores a un precio relativamente bajo y, además, la complejidad de programar éstos ha ido reduciéndose poco a poco, permitiendo dejar de necesitar un lenguaje tipo ensamblador de bajo nivel y baja abstracción para utilizar lenguajes de alto nivel como C/C++ con mayor abstracción, que facilitan mucho más su programación.

Para aplicaciones de tipo IoT ha sido importante la aparición de microcontroladores de bajo coste con módulo WIFI y bluetooth integrado. Entre otros voy a destacar el ESP8266 y ESP32 los cuales han permitido hacer proyectos IoT a un precio muy bajo. Además, gracias a ser compatibles con Arduino ha permitido utilizar todas las herramientas y librerías ya desarrolladas.

Programación

Para hablar de la programación de un microcontrolador se debe entender la estructura base de todo código orientado a la programación de microcontroladores.

En primer lugar, se debe saber que todo programa escrito en C está formado por varios módulos llamados funciones, donde la función principal se llama *main*. En los microcontroladores es la función que se ejecuta en el arranque.

Para los microcontroladores existe otra función importante, que recibe distinto nombre en función del fabricante: *loop(), for(;;), while(1)* o *while(true)*. Esta función es la parte del código que va a estar ejecutándose en bucle e indefinidamente.

Antes de cualquier función, en el código del programa se tienen las declaraciones y directivas: las librerías y archivos que se van a incluir en el código, las definiciones de macros y/o constantes, declaraciones de tipos de datos y la declaración de variables globales.

Además, como el código de los microcontroladores también se suele poder escribir en C++, un lenguaje orientado a objetos y, por lo tanto, es posible que se tengan declaraciones de objetos para los componentes del dispositivo como sensores o periféricos, como la memoria o la pantalla.

Se debe saber que para comprobar el funcionamiento del microcontrolador se puede mostrar por pantalla información en el ordenador con, por ejemplo, el serial de Arduino o similares como la aplicación *hyperterminal*. De esta manera, se puede comprobar si se está produciendo algún error o si por ejemplo los sensores se están leyendo correctamente.

1.2.3. PROTOCOLOS DE COMUNICACIÓN PARA µC

Para los microcontroladores (o µC) el tipo de transferencia más utilizada es la serie, serial o secuencial. La principal ventaja que se puede encontrar en este tipo de transferencia es que necesita menos líneas de transmisión que una transferencia en paralelo (Fig. 14). Además, este método elimina algunos problemas como sería la sincronización, necesaria en las transferencias en paralelo.

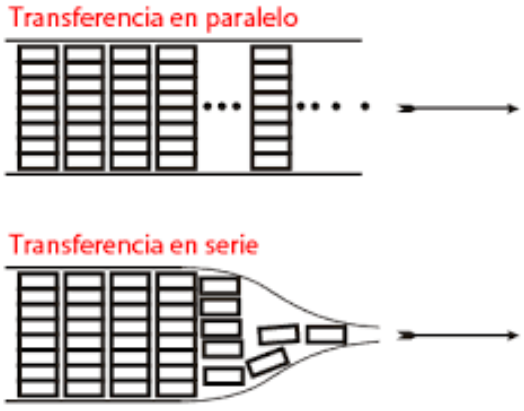


Figura 14 - Transferencia en serie y paralelo [15]

Es cierto que la transferencia en serie tiene menor rendimiento en la transmisión que una transferencia en paralelo a la misma frecuencia. Por ello, para las transferencias en serie se suele utilizar frecuencias más altas para compensar esto.

Entre los protocolos de comunicación disponibles, los más utilizados en ámbitos de IoT son el I2C, el UART y el SPI. Estos tres protocolos utilizan transferencia en serie y sus características son las siguientes:

I2C

El término I2C o I^2C proviene de las siglas inglesas "*Inter-Integrated Circuit*" [16] y es un protocolo de comunicación digital muy utilizado en dispositivos electrónicos (Fig. 15).

Las características de este protocolo son:

- Es de tipo serie, síncrono y *half-duplex*. Por lo tanto, necesita una señal de reloj y la comunicación es posible en ambas direcciones, pero solo se puede transmitir en una a la vez.
- Utiliza dos líneas de comunicación para transmitir los datos entre los dispositivos: SCL (Serial Clock) y SDA (Serial Data). La línea SCL es la línea de reloj que sincroniza la transmisión de datos, y la línea SDA es la línea que transporta los datos.
- Define dos tipos de dispositivos: el dispositivo maestro y el dispositivo esclavo. El dispositivo maestro es el que inicia la comunicación y controla el bus, mientras que los dispositivos esclavos responden a las solicitudes del dispositivo maestro.
- Utiliza direcciones únicas para identificar a los dispositivos conectados al bus. Estas direcciones permiten que el dispositivo maestro se comunique con el dispositivo esclavo correcto. Por lo tanto, este protocolo de comunicación permite la transmisión de datos con solamente dos líneas a múltiples dispositivos.

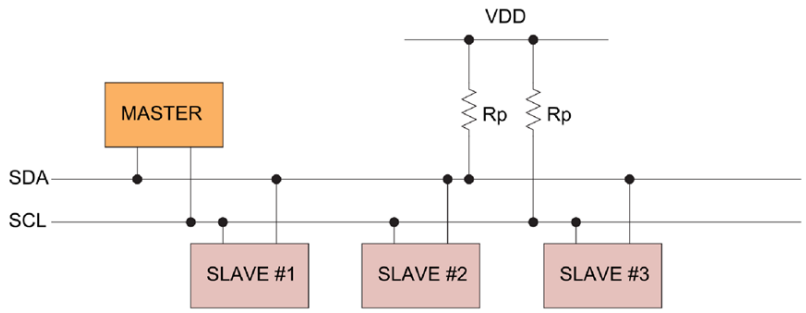


Figura 15 – Diagrama del protocolo I2C. Se observan las resistencias pull-up necesarias para la comunicación mediante el protocolo I2C. El funcionamiento de estas resistencias se explica en el siguiente apartado. [17]

UART

El protocolo UART (Fig. 16), cuyo acrónimo proviene de "Universal Asynchronous Receiver/Transmitter" [18], se utiliza para comunicar datos entre dispositivos. A diferencia de otros protocolos de comunicación, éste es asíncrono y, por lo tanto, no necesita un reloj y esto facilita trabajar con él.

Sus principales características son:

- Tiene dos líneas de datos, una para transmitir (TX) y otra para recibir (RX). Los datos que se transmiten desde el puerto TX del emisor hasta que se reciben en el RX del receptor.
- Esta transmisión es de tipo asíncrona, por lo tanto, no necesita ninguna señal de reloj para sincronizar la transmisión.
- No tiene ningún dispositivo esclavo o maestro, la comunicación es punto a punto.
- Tiene un bit de paridad opcional para detectar errores, y un bit o dos de parada para indicar si es el final de paquete.
- Es importante que ambos dispositivos UART tengan definida la misma velocidad en baudios, que indica la cantidad de bits transmitidos por segundo.

La comunicación UART permite tres formas de transmitir la información entre dispositivos:

- Símplex: transmite los datos en una dirección, unidireccional.
- Half-duplex: transmite los datos en ambas direcciones, bidireccional, pero no simultáneamente.
- Full-duplex: permite transmitir los datos en ambas direcciones simultáneamente, bidireccional.

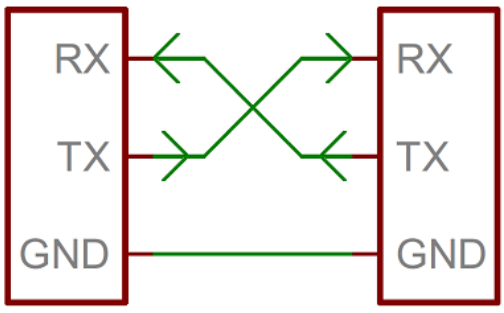


Figura 16 – Diagrama de conexión de dispositivos mediante el protocolo UART [19]

SPI

Las siglas SPI provienen del inglés "*Serial Peripheral Interface*" [20] o en castellano, interfaz de periféricos en serie (Fig. 17). Es comúnmente utilizado para algunos periféricos como sensores, memorias, pantallas LCD, etc... Este protocolo de comunicación es muy utilizado para estos periféricos debido a que éste protocolo SPI es relativamente simple y bastante eficiente, facilitando un comunicación rápida y fiable. Además, éste permite una comunicación *full-duplex*, que permite transmitir los datos en ambas direcciones simultáneamente, bidireccional.

Las principales características son:

- Este protocolo es síncrono, serie y de tipo *full-duplex*.
- Este protocolo consiste en la comunicación de dos o más dispositivos de los cuales uno es maestro y el resto se les llama esclavos. Este dispositivo maestro controla la transmisión de datos y selecciona el dispositivo esclavo con el que comunicarse, mientras que el resto de los dispositivos esclavos esperan a ser seleccionados para comunicarse.
- Tiene cuatro líneas para la transmisión de datos. La primera es una línea de selección SS (*Slave Select*) que es utilizada por el maestro para seleccionar el esclavo a comunicarse. La segunda es SCK (*Serial Clock*) que es un reloj proporcionado por el dispositivo maestro. El tercer es MISO (*Master In Slave Out*) es la entrada de datos del maestro y la salida del esclavo, línea donde los esclavos responden al maestro. El cuarto y último es MOSI (*Master Output Slave In*) es la salida de datos del maestro y la entrada del esclavo, línea donde el maestro envía datos a los esclavos.

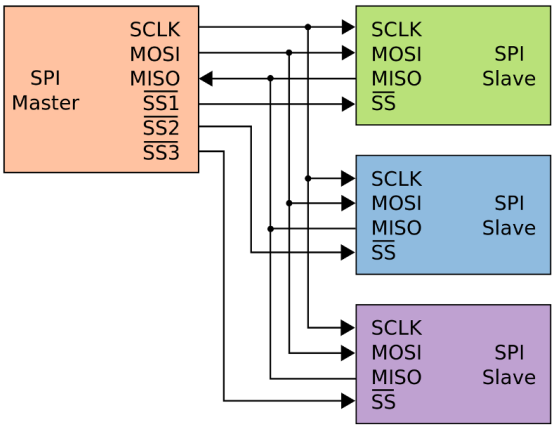


Figura 17 - Diagrama del protocolo SPI. Conexión del maestro con varios esclavos [21]

1.2.4. RESISTENCIAS PULL-UP Y PULL-DOWN

Las resistencias *pull-up* o *pull-down* se utilizan para mantener la señal a un nivel conocido y tienen múltiples aplicaciones.

Para algunos protocolos de comunicación es necesario disponer de algunas resistencias de *pull-up* que mantienen la señal a un nivel conocido, en este caso a alto +3,3 V o +5 V (son los niveles lógicos más conocidos) cuando el dispositivo se encuentra en reposo. Entre los protocolos que necesitan estas resistencias de tipo *pull-up* tenemos el protocolo I2C, como se veía en la figura 13, y algunos casos del SPI, como por ejemplo para utilizar una microSD [22].

Además, existen otras aplicaciones de las resistencias *pull-up* y *pull-down* como para el uso de botones. La función de estos botones es mantener el nivel lógico en un estado alto o bajo cuando el botón se mantenga sin pulsar. Cuando el mismo se pulse, pondrá la señal al nivel lógico opuesto (Fig. 18).

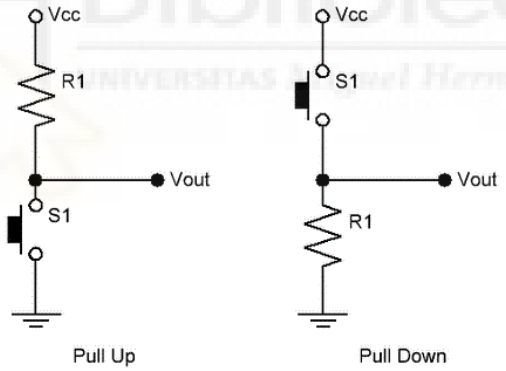


Figura 18 – Resistencia pull-up y pull-down [23]

1.2.5. MQTT

El protocolo llamado MQTT de sus siglas en inglés *"Message Queing Telemetry Transport"* [24] (Fig. 19) es un protocolo de comunicación punto a punto que consiste en un servicio de publicación y subscrición (pub-sub).

Para comunicarse mediante este servicio MQTT es necesario seguir los siguientes pasos: primero el cliente MQTT establece una conexión con el agente MQTT; segundo y último, una vez conectado, el cliente puede publicar su información o subscribirse para recibir información o ambas.

Este protocolo es muy utilizado en el ámbito de IoT para subir los datos obtenidos por dispositivos a la nube o incluso para recibir información de la nube. Este protocolo se ha convertido en un elemento muy importante en el ámbito del IoT debido a sus características principales como su sencillez o que no es necesario disponer de un dispositivo muy potente, con mucha autonomía o con un gran ancho de banda. Además, este protocolo es muy seguro y debido a que está altamente testado se considera robusto y fiable.

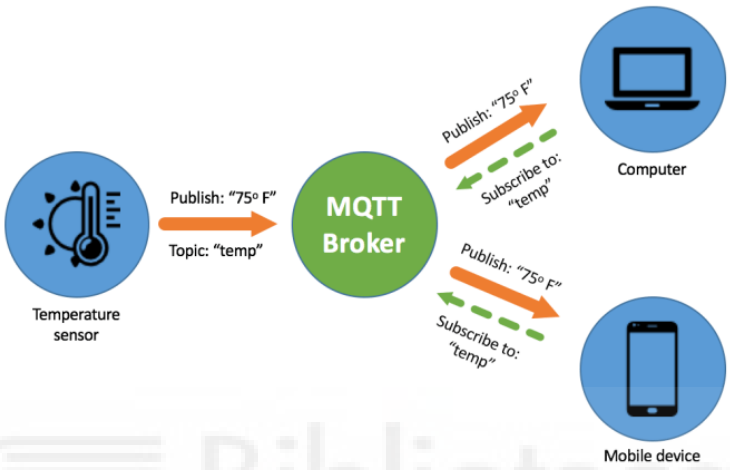


Figura 19 – Diagrama de comunicación MQTT - El sensor publica el valor medido dentro de un topic, todos los clientes suscritos a ese topic recibirán la información, en este ejemplo es la temperatura medida por el sensor [25]

1.2.6. DISEÑO DE LA PCB

Para este proyecto se ha diseñado una PCB, *printed circuit board* en inglés, que permite darle un aspecto más profesional al proyecto y tener el dispositivo en un espacio más reducido. Aunque, antes de realizar el diseño de una PCB se deben tener unos conceptos básicos en cuenta.

En primer lugar, los componentes se dividen en dos grupos en función del tipo de tecnología de sus pines (Fig. 20):

- Componentes de inserción (*Through-Hole* o THT): son componentes que para el posicionamiento sobre la placa impresa sus pines necesitan atravesar la misma.
- Componentes superficiales (SMD, *surface-mounted device*): son componentes cuyos pines quedan apoyados en la superficie de la PCB.

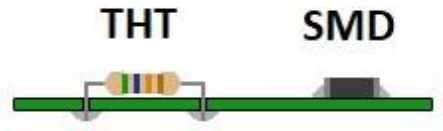


Figura 20 – THT vs SMD [26]

Los componentes tipo SMD van a permitir reducir el tamaño de la placa debido a que, por lo general, ocupan mucho menos espacio que sus equivalentes tipo THT.

En segundo lugar, existen unos elementos básicos dentro de la PCB aparte de los componentes. Estos componentes básicos son identificables en la figura 21 y son: pista, pad, vía y planos.

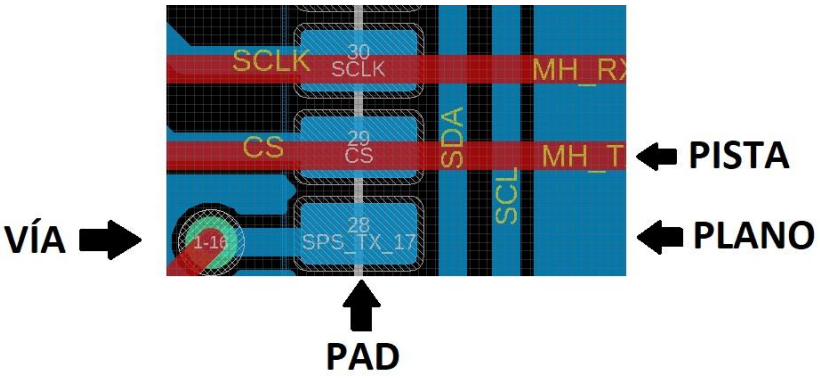


Figura 21 – Explicación PCB

Las pistas son los "cables" o conexiones dentro de la PCB que conectan los pines de los componentes. Estos pueden ser de mayor o menor tamaño, en función de esto permitirán mayor o menor corriente y tendrán mayor o menor resistencia. Una cosa que se debe tener en cuenta es que se deben evitar los ángulos de 90º y se ha de tratar de utilizar siempre ángulos de 45º.

Los planos son superficies que realizan una función similar a la de las pistas, el conexionado de los pines de los componentes en la PCB. Estos planos son muy utilizados para, por ejemplo, realizar la conexión de todos pines de masa de los componentes.

Los pads son las superficies o agujeros, en función de la tecnología de los componentes, donde se posicionan los pines del componente. En el caso de componentes THT los pads son agujeros metalizados donde se puede colocar los pines del componente. Mientras que en el caso de SMD son superficies metálicas expuestas para colocar los pines del componente.

Las vías son pequeños agujeros metalizados en la PCB que permiten comunicar pistas y planos que se encuentran en diferentes capas. No es recomendable colocar vías en los pads de los componentes.

2. OBJETIVOS Y PLANIFICACIÓN

2.1. OBJETIVOS

La calidad del aire es un factor esencial para tener una vida saludable. Especialmente cuando nos encontramos en un entorno cerrado la gran mayoría del tiempo, es importante controlar atentamente la composición química del aire ambiente, para de esta manera verificar que los niveles de contaminación atmosférica no superan los límites recomendados por la OMS [2].

Sin embargo, en la mayoría de los casos no se cuenta con estaciones de medición del aire debido al alto coste que estas presentan. Especialmente en hogares y espacios cerrados, donde se pasa la gran mayoría del día, es recomendable disponer de estos dispositivos de medida que permitan monitorizar estos valores. Además, aunque existen dispositivos en el mercado, estos no proporcionan suficiente precisión de medida a un precio asequible.

Por ello, el principal objetivo es tratar de acercar al público en general un dispositivo económico que sea accesible a todos los bolsillos, con el cual se pueda conocer instantáneamente la calidad del aire y que, además, esta información sea accesible desde cualquier dispositivo. El dispositivo que pretende cumplir estos objetivos se ha denominado **AirKnowledge**. Como se puede imaginar el nombre hace referencia al conocimiento que nos va a otorgar sobre el aire que respiramos.

Para que el proyecto sea viable su realización será necesario cumplir con diversos requisitos:

- Mantener un coste económico reducido, no superando los 120 euros.
- Obtener medidas fiables.
- Obtener la medición de varios parámetros desde un único dispositivo.
- Poder consultar los datos de forma remota.
- Posibilidad de configurar el modo de guardado: en la nube o en microSD.
- Obtener un dispositivo compacto y elegante, para ello se diseñará una placa de circuito impreso, o PCB en inglés, y carcasa a medida.

2.2. PLANIFICACIÓN

Para el correcto desarrollo de un proyecto es importante conocer previamente las tareas a realizar, esto permite tener una visión global del proyecto desglosado en todas las actividades y cómo se relacionan entre sí. Además de esto, si se realiza una buena planificación, se podrá distinguir los distintos recursos materiales, económicos y de personal de esta forma poder planificar el proyecto para cumplir con los plazos y objetivos esperados.

Para la fabricación de este dispositivo se definieron las distintas actividades y tiempos esperados para su realización. Una vez definidas las actividades, se representaron mediante un diagrama de Gantt y de esta manera, se pudo visualizar su evolución a lo largo del tiempo y obtener de esta manera una estimación de la duración del proyecto y las actividades críticas. Estas actividades críticas, son las actividades más importantes porque en caso de sufrir algún retraso afectarían al coste y el tiempo de todo el proyecto.

En la siguiente tabla, tabla 5, se puede observar las distintas actividades del proyecto:

ID	Actividad	Duración (días)	Dependencia
A	Estado del arte	7	-
B	Estudio de mercado	7	-
C	Compra del material y entrega	10	-
D	Montaje provisional	7	C
E	Programación sensores	25	A, B
F	Programación microSD y RTC	15	A, B
G	Programación pantalla	7	E, F
H	Programación interfaz de usuario	8	E, F
I	Programación Wifi y plataformas IoT	5	E, F
J	Funcionamiento del bucle	5	G, H, I
K	Esquemático PCB	15	G, H, I
L	Diseño PCB	7	K
M	Montaje y soldado	4	L
N	Diseño carcasa	4	L
Ñ	Fabricación carcasa	1	N
O	Análisis consumos	2	M

Tabla 5 – Lista de actividades del proyecto

Si se representan dichas actividades en el tiempo y se tienen sus dependencias en cuenta obtendremos el siguiente diagrama de Gantt, se puede observar en la siguiente figura (Fig. 22). Para utilizar dicho diagrama hemos utilizado el programa Gantt Proyect [27]. En él se puede observar el camino crítico formado por las siguientes actividades:

$$\mathbf{A/B \rightarrow E \rightarrow H \rightarrow K \rightarrow L \rightarrow N \rightarrow Ñ}$$

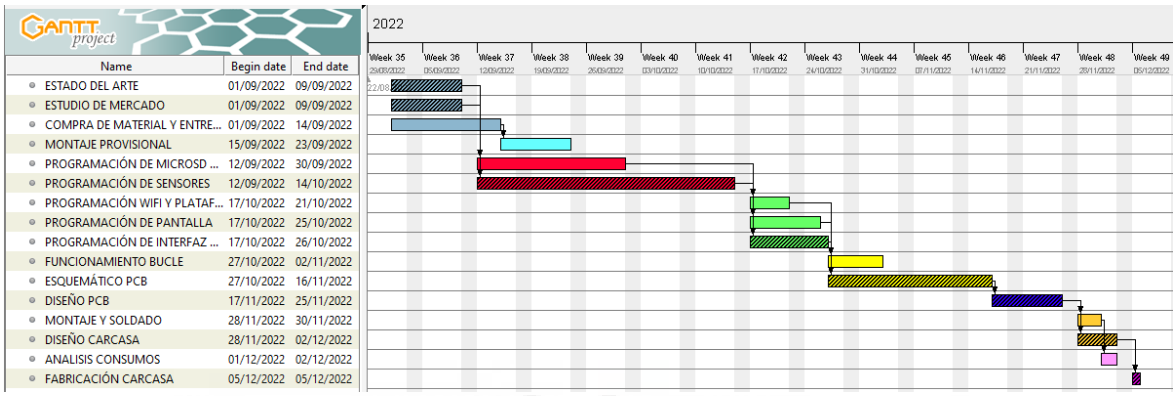


Figura 22 – Gantt Project – En este diagrama podemos ver las distintas actividades divididas en filas. La duración y dependencia es visible en barras de colores. Además, se puede sacar el camino crítico observando las actividades sombreadas.

Si se suma la duración de cada una de las actividades del camino crítico se obtiene la duración estimada del proyecto. En este caso, se obtuvo una duración de 96 días, es decir, 3 meses para el desarrollo del proyecto.

A partir de la información obtenida del diagrama de Gantt se deben tomar distintas decisiones sobre cómo planificar el proyecto. En las actividades críticas se deben centrar los recursos disponibles con el fin de para evitar problemas y que pueda provocar un aumento en la duración del proyecto y el coste de este.

No se debe olvidar al resto de actividades puesto que, aunque tengan una mayor holgura para atrasarse en el tiempo, siguen siendo importantes y se debe estar atentos para que no afecten al correcto desarrollo del proyecto.

3. MATERIAL

3.1. ESQUEMA

En este apartado se comenta, en primer lugar, cuál es el diagrama inicial o base del dispositivo que se quiere diseñar y sus elementos esenciales, en segundo lugar, cuál será el diagrama final del mismo y los elementos que se van a utilizar.

3.1.1. BASE

Para hacer el diseño del dispositivo IoT de este proyecto, se utilizará un esquema base (Fig. 23) el cual sea fácilmente escalable y adaptable a las distintitas características y demandas del posible cliente.

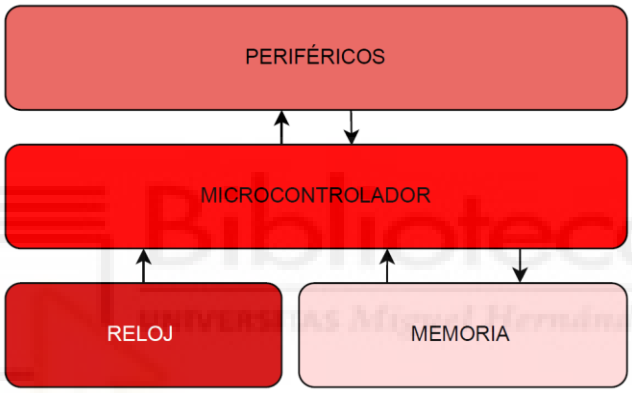


Figura 23 – Diagrama base

El esquema base parte de un microcontrolador con módulo wifi, interno o externo, para poder subir los datos a la nube y tener la posibilidad de acceder a estos desde cualquier dispositivo conectado a internet.

Aparte de esto, tiene una memoria donde poder almacenar los datos en ella para los casos en los no se disponga de una red wifi o para no perder la información en caso de caídas imprevistas de la red.

Junto a la memoria es necesario un periférico que permita mantener la hora actual para hacer el registro de los datos con la información temporal, y que sea capaz de mantener la hora ante pérdidas de la alimentación.

Por último, a este esquema se le puede incorporar cualquier periférico adicional para cumplir las especificaciones que exija el posible usuario final.

3.1.2. FINAL

Partiendo del esquema base, comentado en el apartado anterior, se define un esquema final (Fig. 24) que cumpla las especificaciones deseadas.

Por un lado, puesto que se busca un dispositivo que dé información relevante sobre la calidad del aire será necesario que disponga de un sensor de PM y otro de CO_2. Además, como no podría ser de otra manera, que tenga información meteorológica básica como temperatura, humedad y presión. Estos dos sensores son los que se incluirán en el diseño del esquema final.

La memoria utilizada es una microSD debido a su tamaño reducido, gran capacidad y facilidad de comunicación.

El dispositivo seleccionado que proporciona las funcionalidades de reloj y permite mantener la hora es un RTC, de las siglas en ingles de "*Real Time Clock*".

Por otro lado, se desea que el dispositivo muestre la información para, en caso de encontrarnos junto a él, poder ver lo que están midiendo los sensores de una manera fácil y rápida.

Por último, se quiere incorporar una interfaz mediante la que se pueda configurar el funcionamiento del dispositivo. Esto será posible mediante dos botones físicos que permiten cambiar entre los modos definidos del dispositivo. Se desea tener la opción de cambiar, por ejemplo, el modo de guardar los datos y que se pueda elegir si se quiere guardar la información solo en la memoria externa o en la nube o en ambas.

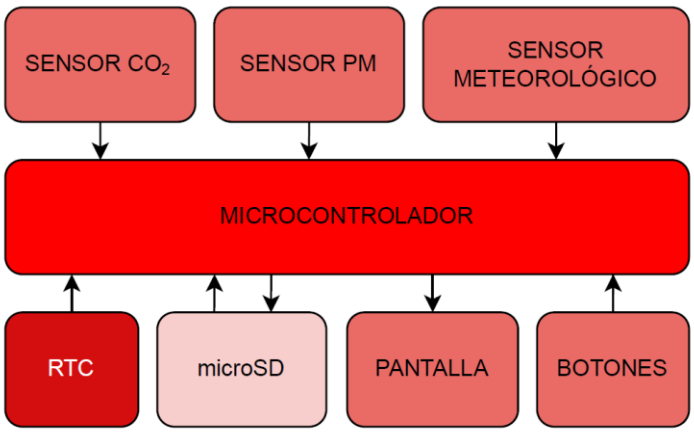


Figura 24 - Diagrama final

3.2. ESTUDIO DE MERCADO

3.2.1. COMPARATIVA DE SENSORES DE PM

En este apartado se va a presentar un estudio de mercado y comparativa de los distintos sensores que miden material particulado que podemos encontrar a la venta. Para el sensor de CO_2 no se ha realizado ya que existen menos opciones y el escogido presenta una resolución y precio óptimo. Los sensores que se compararon se caracterizan por ser de bajo coste y supuesta calidad en la medida. Se han preseleccionado los cuatro siguientes en función de la disponibilidad, precio y características (Tab. 6): ZH03B, SDS011, SPS30 y PMS7003.

Fabricante	Modelo	Precio (€)	Salidas
Winsen	ZH03B	~20	PM_1, $PM_{2.5}$, PM_{10}
Plantower	PMS7003	~20	PM_1, $PM_{2.5}$, PM_{10}
Sensirion	SPS30	~50	PM_1, $PM_{2.5}$, PM_4, PM_{10}
Nova	SDS011	~20	$PM_{2.5}$, PM_{10}

Tabla 6 – Lista de sensores

Esta comparativa consistió en diseñar e implementar un dispositivo siguiendo el esquema base para cada uno de los sensores preseleccionados y comparar la medida de los sensores con un sensor de referencia. El captador de referencia utilizado es el nombrado en el apartado de captadores llamado GRIMM 1109. Por otro lado, se utilizó cada sensor por duplicado y así se pudo intercomparar también entre ellos para comprobar la fiabilidad de estos.

Se han realizado las medidas en dos periodos: primero en el interior de un edificio, con un ambiente, en principio, con baja concentración de partículas y segundo en el exterior, donde se puede suponer, a priori, un ambiente con mayor concentración de partículas, al menos en momentos puntuales del día como en las horas punta de tráfico. Cada campaña tuvo una duración, de, al menos, una semana completa. El primer periodo de medición se hizo entre finales de abril y principios de mayo. Mientras que el segundo periodo de medición se hizo a mediados de mayo. Las medidas se han realizado cada 5 segundos para, posteriormente, realizar un promedio cada 10 minutos. Así conseguimos suavizar cualquier valor anómalo puntual al mismo tiempo que garantizamos una cantidad de datos suficiente para realizar las comparaciones.

Tras realizar las medidas se procedió a comparar los datos obtenidos. Pero antes se deben definir unas variables que permitirán comparar los sensores:

- Coeficiente de determinación (R^2): es el cuadrado de la correlación (R) e indica la proporción de la variación o desviación entre las medidas de los sensores comparados.
- Línea de tendencia: si forzamos a pasar por cero obtenemos una ecuación del tipo $y = m \cdot x$, donde m es la constante de la pendiente de la recta. Este valor indica el número por el que se debe multiplicar las medidas del sensor del eje x para obtener medidas más similares a las del sensor del eje y.

De estas dos variables podemos sacar la siguiente información: cuanto más cercana a uno sea la pendiente, los valores del sensor de bajo coste más se asemejarán al de referencia; mientras que cuanto más se acerque a uno el coeficiente de correlación a 1 significará que la bondad del ajuste es muy buena y que, por lo tanto, la correlación entre ambos sensores es significativa.

Estos valores comentados anteriormente son importantes, dan bastante información y son fácilmente calculables a partir de hoja de cálculo como Excel. Aunque los resultados del análisis de correlación pueden ayudar a discernir si dos de los sensores miden de igual forma, es necesario utilizar parámetros más específicos a la hora de comparar series de datos como son los errores que se definen a continuación [28] (Ec.1 y Ec.2):

- Error de raíz cuadrada media, RMSE, del inglés *"Root Mean Squared Error"*.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(c_{sensor\ low\ cost,i} - c_{ref,i})^2}{n}}$$

Ecuación 1 – Expresión para la obtención del RMSE

- Error absoluto medio, MAE, de las siglas de *"Mean Absolute Error"*.

$$MAE = \frac{\sum_{i=1}^{n}\left|c_{sensor\ low\ cost,i} - c_{ref,i}\right|}{n}$$

Ecuación 2 – Expresión para la obtención del MAE

Donde $c_{sensor\ low\ cost,i}$ y $c_{ref,i}$ se refiere a las concentraciones del sensor a comparar y el de referencia respectivamente, y n es el número de medidas.

Ambos términos, MAE y RMSE, dan información del error medio del modelo de predicción. Estos términos se utilizan para comparar modelos con valores reales. En este caso se utilizan para comparar valores de sensores de bajo coste con el valor del sensor de referencia. El resultado de estos términos puede oscilar entre cero y valores tan altos como las mismas concentraciones medidas. Aunque la principal diferencia y razón para utilizar ambos términos es que RMSE pondera mucho más los errores grandes y, por lo tanto, es mucho más crítico. Mientras que, el MAE es más permisivo con los errores grandes y valora el comportamiento general del sensor.

A continuación, se muestran las gráficas obtenidas representando los valores de la medida de los sensores con respecto a los valores medidos por el medidor de referencia para ambos periodos (Fig 25-26):

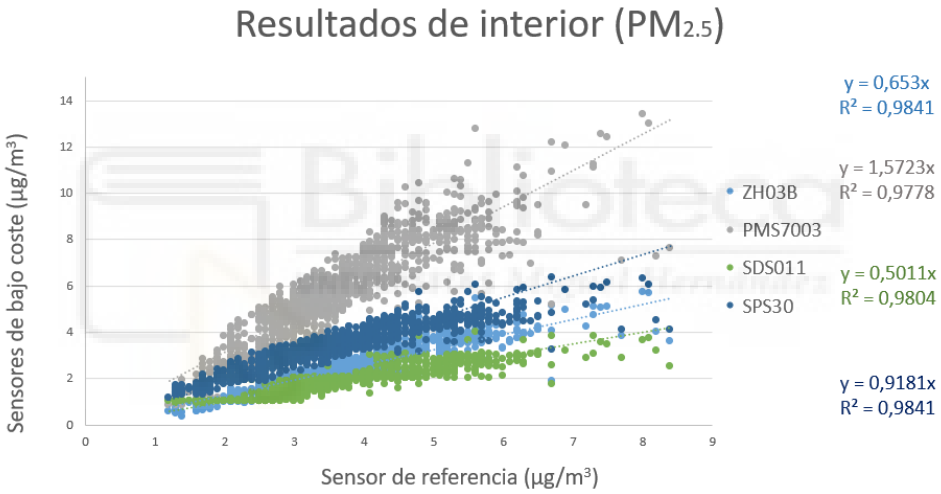


Figura 25 - Comparativa sensores - low-cost vs referencia – Interior

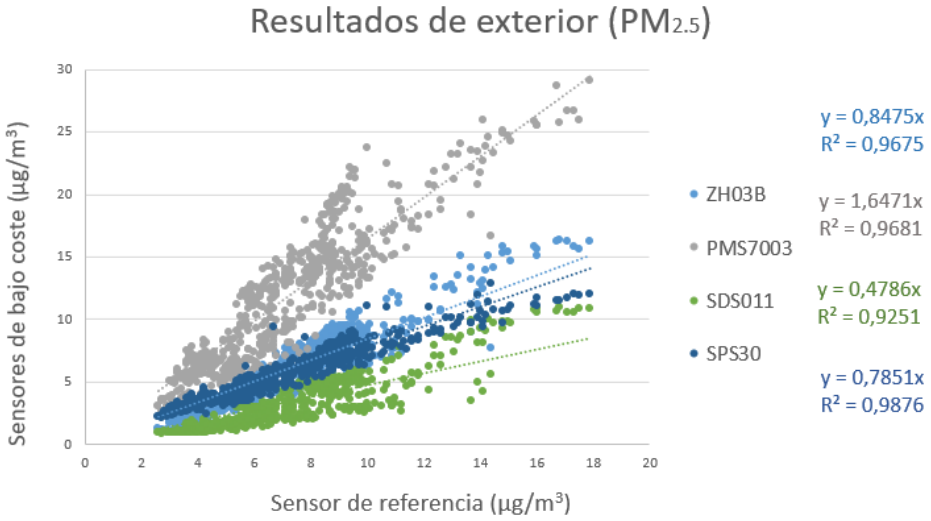


Figura 26 - Comparativa sensores - low-cost vs referencia – Exterior

De estas gráficas obtenemos, el valor del R^2 y la pendiente, resultados obtenidos mediante la relación entre el sensor de referencia y cada uno de los sensores de bajo coste. Se observa como los mejores resultados se obtienen para el sensor SPS30 puesto que tanto para interiores como para exteriores se obtienen valores de R^2 muy cercanos a 1. En relación con la pendiente, también es el sensor con valores más cercanos a uno si tenemos en cuenta las dos campañas. Después de este dispositivo el segundo que tiene un comportamiento más parecido al de referencia sería el sensor ZH03B.

Además, como se ha comentado anteriormente, se han duplicado los sensores de bajo coste midiendo la concentración de partículas con dos sensores del mismo tipo para, de esta manera, discernir si los valores dados por los mismos sensores son comparables. Gracias a esto podemos entender como de fiables son los sensores y si en caso de comprar en masa podríamos esperar que todos midiesen, al menos, de la misma forma (Fig. 27).

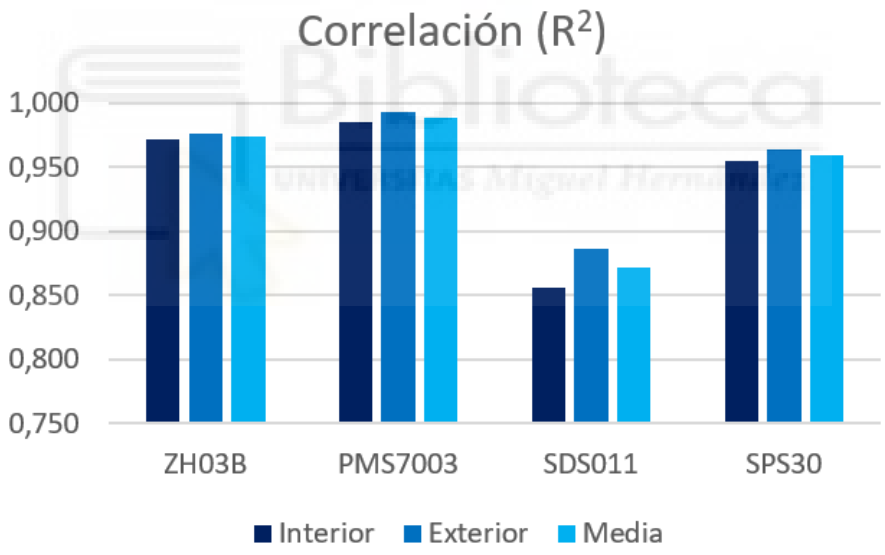


Figura 27 – Correlación y fiabilidad de los sensores

De la figura 27 podemos deducir que los sensores con mayor fiabilidad serían el PMS7003, el ZH03B y el SPS30. El sensor SDS011 presenta unos niveles de correlación inferiores al resto de los sensores.

Por último, los valores obtenidos por los errores MAE y RMSE son los siguientes:

SENSORES	MEDIDAS	MAE	RMSE
ZH03B	INTERIOR	1,2948	1,4962
	EXTERIOR	1,4641	1,8028
SDS011	INTERIOR	1,7905	2,0422
	EXTERIOR	3,9669	4,2082
SPS30	INTERIOR	0,4081	0,9197
	EXTERIOR	1,5328	1,8647
PMS7003	INTERIOR	2,0697	2,4296
	EXTERIOR	4,6324	5,7242

Tabla 7 – RMSE y MAE

Como podemos observar, en la tabla 7, los mejores resultados los obtenemos para los sensores SPS30 y ZH03B. Los valores obtenidos para los otros sensores (PMS7003 y SDS011) son más elevados sobre todo en la campaña de exteriores.

Finalmente, a partir de los resultados presentados anteriormente se puede decir que el sensor con mayor estabilidad y que se acerca más a los valores del captador de referencia es el sensor SPS30, de Sensirion, aunque el sensor ZH03B también sería una opción para tener en cuenta.

3.2.2. ELECCIÓN DE LOS SENSORES

SENSOR DE PM

Tras la comparativa realizada en el apartado anterior, podemos concluir que el sensor SPS30 (Fig. 28) es la mejor de las alternativas disponibles debido a que tiene muy buena relación entre la calidad de sus medidas y el precio. Además, ofrece esto a un tamaño muy reducido.

Figura 28 – SPS30 [29]

En la hoja de características [29] podemos ver como la comunicación con el microcontrolador puede hacerse tanto por I2C como por UART. En este proyecto será utilizado con el protocolo UART. Por otro lado, la tensión de alimentación debe ser a 5V. Las características principales por las cuales elegimos este sensor son (Tab. 8):

Tamaños de concentración de partículas	PM_1 $PM_{2.5}$ PM_4 PM_{10}
Rango de tamaño para la concentración de partículas	PM1.0 → 0,3 a 1,0 µg/m³ PM2.5 → 0,3 a 2,5 µg/m³ PM4 → 0,3 a 4,0 µg/m³ PM10 → 0,3 a 10 µg/m³
Rango de concentración	0 a 1000 µg/m³
Precisión de medida Concentración de partículas	±10 µg/m³ → 0 a 100 µg/m³ ±10% → 100 a 1000 µg/m³
Resolución de la medida	1 µg/m³

Tabla 8 – Características SPS30 [29]

SENSOR DE CO_2

Para el caso del sensor de CO_2, como se ha comentado antes, no se ha realizado ningún estudio de mercado debido a que no existe tanta cantidad sensores de bajo coste disponibles. Entre los pocos sensores CO_2 de bajo coste disponibles en el mercado, el que más destaca es MH-Z19B, un sensor que cuenta con unas características muy buenas para su reducido precio. Este sensor es de la empresa de origen chino llamada Winsen, bastante reconocida por la fabricación de sensores de bajo coste.

El MH-Z19B [30] (Fig. 29) es un sensor de pequeño tamaño que, mediante infrarrojos no dispersivos (NDIR), es capaz de detectar la concentración de CO_2 en el aire. Este sensor se caracteriza por tener alta sensibilidad y muy buena estabilidad ante variaciones en el

ambiente, gracias a por ejemplo disponer de un sensor interno de temperatura que le permite hacer la correspondiente compensación.

Figura 29 – MH-Z19B [31]

En la hoja de características [31] podemos ver como la comunicación con el microcontrolador tiene que hacerse mediante UART, el cual se ha explicado en el apartado de fundamentos teóricos, aunque también dispone de una salida analógica basada en pulsos PWM, *pulse width modulation*, que mediante la variación en el ancho de los pulsos informa de la cantidad de concentración de CO_2. Por otro lado, la tensión de alimentación debe ser de 5V.

Las características principales por las cuales elegimos este sensor son (Tab. 9):

Rango de medida	400 a 10000 ppm
Precisión de la medición	50 ppm + 5% de la medida.
Rango de trabajo temperatura	0 a 50ºC
Rango de trabajo humedad	0 a 90% RH

Tabla 9 – Características MH-Z19B [32]

SENSOR DE HUMEDAD, TEMPERATURA Y PRESIÓN ATMOSFÉRICA

Para el caso de las variables meteorológicas: temperatura, humedad y presión, se ha optado por buscar un sensor que ofrezca la medida de estos tres valores en un solo dispositivo.

En el mercado podemos encontrar distintas alternativas que midan los valores anteriormente descritos, siendo los fabricados por la empresa Bosch los que destacan por su estabilidad y fiabilidad. Entre los sensores disponibles de esta marca se puede destacar el sensor BME280 (Fig. 30). Este sensor tiene un precio reducido, una precisión elevada y un pequeño tamaño.

Figura 30 – BME280 [33]

Además de todo lo anterior, la comunicación con la que se puede comunicar el sensor resulta sencilla puesto que permite tanto unos protocolos de comunicación I2C como SPI, ambos explicados en el apartado de fundamentos teóricos.

Este sensor es muy utilizado en todo tipo de proyectos IoT que requieran la medición los valores de temperatura, humedad y presión. En especial, en aquellos que estén alimentados por batería, debido al bajo consumo que tiene este componente.

Las características principales por las cuales elegimos este sensor son (Tab. 10):

Temperatura	Rango de medida	-40 a 85°C
	Precisión de la medición	25 °C → ±0,5 ºC 0 a 65 °C → ±1,0 ºC -20 a 0 °C → ±1,25 ºC -40 a -20°C → ±1,5 ºC
Humedad	Rango de medida	0 a 100%
	Precisión de la medición	±3% RH
Presión Atmosférica	Rango de medida	300 a 1250 hPa
	Precisión de la medición	0 ~ 65 °C → ±1,0 hPa -40 ~ 0 °C → ±1,7 hPa

Tabla 10 - Características BME280 [34]

3.2.3. COMPARATIVA CON LOS DISPOSITIVOS DEL MERCADO

A continuación, se van a comparar las especificaciones de los sensores que se van a utilizar en este proyecto, llamado AirKnowledge, con respecto a los dispositivos nombrados en el estado del arte.

En primer lugar, el Purple Air:

Dispositivos	AirKnowledge	Purple Air
Temperatura	-40 a 85°C, ±1,0 ºC (0 a 65 °C)	-40 a 85°C, ±1,0 ºC (0 a 65 °C)
Presión	300 a 1250 hPa, ±1,0 hPa	300 a 1100 hPa, ±0.25 %
Humedad	0 a 100%, ±3% RH	0 a 100%, ±3% RH
CO_2	400 a 10000 ppm, ± (50 ppm + 5%)	No dispone de sensor CO_2
PMx	±10µg/m³ → 0 a 100 µg/m³ ±10% → 100 a 1000 µg/m³	±10µg/m³ → 0 a 100 µg/m³ ±10% → 100 a 500 µg/m³

Tabla 11 – Comparativa AirKnowledge vs Purple Air

En la tabla 11, se puede contrastar las características el dispositivo Air Purple y el dispositivo que se va a crear en este proyecto, AirKnowledge. Se observa que las especificaciones de ambos dispositivos son muy buenas, pero el AirKnowledge incluye adicionalmente un sensor de CO_2 el cual añade información muy relevante.

En segundo lugar, si se compara con el AQ Mesh:

Dispositivos	AirKnowledge	AQ Mesh
Temperatura	-40 a 85°C, ±1,0 ºC (0 a 65 °C)	-20 a 100°C, ±2,0 ºC
Presión	300 a 1250 hPa, ±1,0 hPa	500 a 1500 hPa, ±5hPa
Humedad	0 a 100%, ±3% RH	0 a 100%, ±5% RH
CO_2	400 a 10000 ppm, ± (50 ppm + 5%)	0-5,000 ppm, ±50 ppm
PMx	±10 µg/m³ → 0 a 100 µg/m³ ±10% → 100 a 1000 µg/m³	5 µg/m³ → 0 a 250 µg/m³

Tabla 12 – Comparativa AirKnowledge vs AQ Mesh

En la tabla 12, se puede contrastar las características el dispositivo AQ Mesh y el dispositivo que se va a crear en este proyecto, AirKnowledge. Las conclusiones que se pueden sacar es que no hay mucha diferencia en las especificaciones de ambos dispositivos. La precisión del AirKnowledge es mayor, excepto para las medidas del material particulado. Las pocas diferencias entre ambos dispositivos no justifican la diferencia de precio que va a suponer comprar uno u otro.

3.2.4. REGRESIÓN MULTILINEAL

Como se ha comentado anteriormente, en el apartado de funcionamiento de los sensores low-cost, los sensores de bajo coste tienen ciertas limitaciones puesto que la medida puede ser afectada por factores ambientales como la humedad y temperatura. Puesto que no podemos ignorar este hecho vamos a hacer una regresión multilineal con el objetivo de tratar de compensar los valores de concentración obtenidos ante las posibles variaciones de estas variables meteorológicas.

La regresión lineal múltiple permite determinar la ecuación de dependencia de la medida leída por el sensor de PM con factores meteorológicos. Mediante esta ecuación seremos capaces de compensar los valores PM con los valores medidos de temperatura y humedad. Esto permitirá que los valores que se obtengan puedan tener mayor precisión y menor dependencia ante cambios en la meteorología.

Para hacer la regresión multilineal existen muchas herramientas de software disponibles. En este caso se ha utilizado Matlab debido a su facilidad de uso. Con Matlab es posible hacer la regresión multilineal utilizando una función llamada **fitlm**. Deberemos darle como parámetros de entrada un vector con los valores medidos por el sensor de referencia, por un lado, y, por otro lado, una matriz con los valores medidos por el sensor de bajo coste, la humedad y temperatura.

En el anexo 6, se explica el procedimiento a seguir para obtener la ecuación de la regresión multilineal para el sensor SPS30.

La ecuación obtenida en la regresión lineal es la siguiente (Ec.3):

$$PM_{x,ajustada}\left(\frac{\mu g}{m^3}\right) = 1{,}0579 \cdot PM_{x,medida}\left(\frac{\mu g}{m^3}\right) + 0{,}02534 \cdot Temperatura\ (ºC) + 0{,}011566 \cdot Humedad\ (\%)$$

Ecuación 3 – Ecuación de la regresión multilineal

Esta ecuación obtenida (Ec.3), permitirá reducir la dependencia del sensor con los factores ambientas y mejorar los resultados obtenidos de la pendiente y de la correlación. Como se puede observar la dependencia con la temperatura es mayor que con la humedad.

Para ello se comparará los resultados obtenidos sin aplicar la corrección y los resultados obtenidos aplicando la corrección (Fig. 31-32):

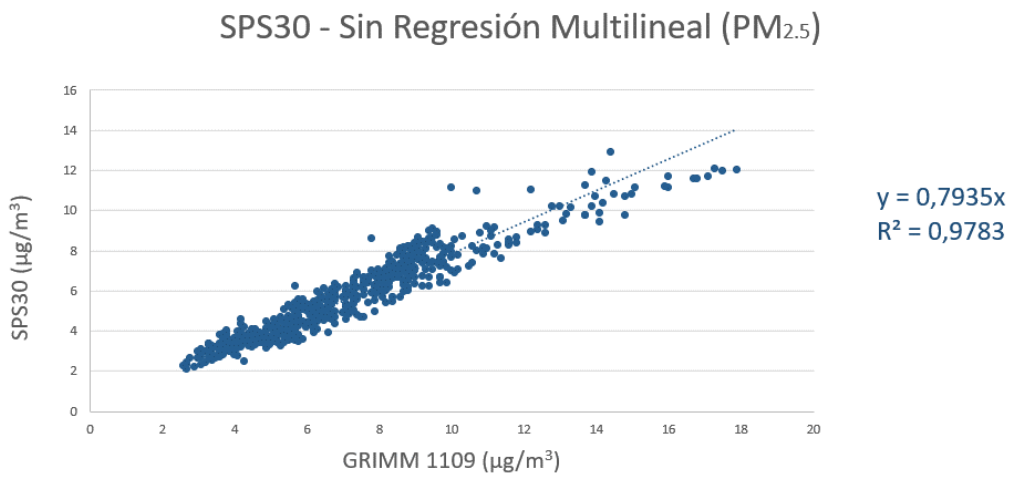


Figura 31 – Resultados sin regresión lineal

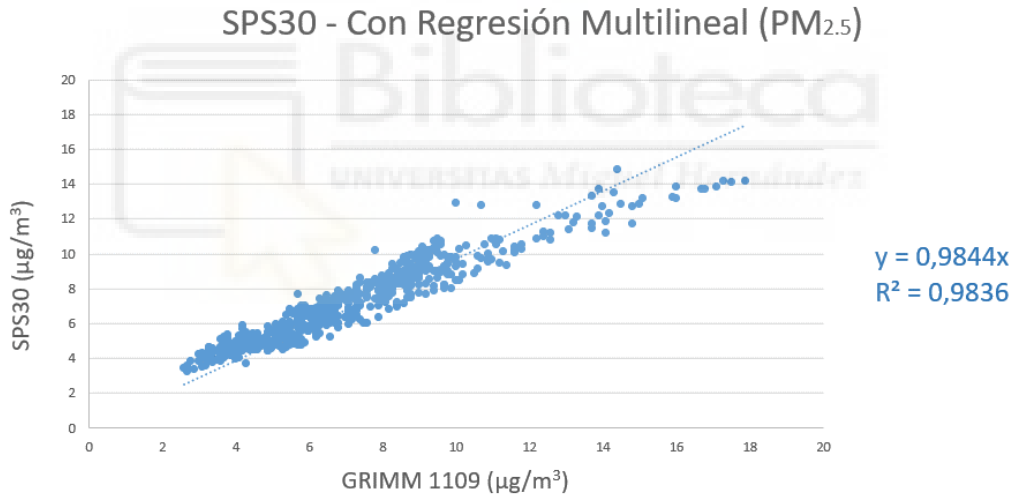


Figura 32 – Resultados con regresión lineal

En los resultados obtenidos en las dos últimas figuras (Fig. 31-32) se puede ver como existe una diferencia significativa cuando se aplica la ecuación obtenida (Ec. 3) con respecto a cuando no se aplica. Tanto el valor de la pendiente como el del cuadrado de la correlación han mejorado considerablemente acercándose a 1.

Por otro lado, se ha calculado el MAE y RMSE para el caso en el que no se ha aplicado regresión multilineal (SIN MR) y para el caso en el que sí se ha aplicado (CON MR).

SENSORES	MAE	RMSE
SPS30 SIN MR	1,4932	1,7868
SPS30 CON MR	0,6819	0,8898

Tabla 13 – MAE y RMSE – Regresión Multilineal

Se observa (Tab. 13) cómo tanto el valor del MAE como del RMSE mejora considerablemente y se reduce mucho su valor.

Como conclusión, una vez vistos los resultados obtenidos y verificada la eficacia de aplicar la regresión multilineal, se ha decidido aplicar la ecuación 3 en el programa del dispositivo que se va a diseñar para obtener unos valores, de la materia partícula, más precisos.

3.3. COMPONENTES

En este apartado se presentarán los componentes que han sido necesarios para llevar a cabo el proyecto y se justificará la elección de cada uno de ellos.

En un primer momento se va a realizar el prototipo en una placa de prototipado, o *protoboard* en inglés, para preparar el código y comprobar el funcionamiento correcto de todo el dispositivo. Posteriormente, una vez preparado el prototipo en la *protoboard*, se va a realizar una PCB, como se comentaba en el apartado de diseño PCB.

Los componentes utilizados se presentan en tres apartados diferentes: los componentes comunes, los componentes que se van a utilizar para el prototipo inicial en la placa de prototipado y, por otro lado, los componentes para la fabricación de la PCB.

3.3.1. COMUNES

Inicialmente, se van a nombrar los componentes que se encuentran presentes tanto en el prototipo como en la PCB. Los componentes comunes son los siguientes

SH1106

La pantalla elegida es la SH1106 (Fig. 33) y servirá para mostrar los datos medidos por los sensores e interactuar con el usuario por los menús del dispositivo.

Figura 33 – SH1106 [35]

Esta pantalla está construida mediante tecnología OLED (diodo orgánico de emisión de luz), monocromática, tiene un tamaño de 1,3”, se puede alimentar tanto a 3,3 V como a 5 V, tiene un precio bastante reducido y su principal ventaja, y el porqué de su elección, es el hecho de que se comunica mediante I2C. Este protocolo, como ya se había comentado antes, permite con solamente dos pines del microcontrolador comunicarnos con múltiples dispositivos.

BOTONES

Se utilizarán dos botones estándar de inserción en placa (Fig. 34). Éstos permitirán interactuar con el dispositivo, acceder a los menús y configurar algunos parámetros del dispositivo.

Figura 34 – Botón [36]

La elección de los botones no es muy relevante ya que cualquier botón permitiría realizar la misma función. Un punto a favor del botón elegido es el tacto que tiene y, además, la tecla que incluye tiene un tamaño cómodo para pulsarlo.

SENSORES

Los sensores que utilizaremos en este proyecto ya se han seleccionado y justificado en el segundo apartado del estudio de mercado. Estos son los siguientes:

- MHZ-19B: Sensor de CO_2 (Fig. 35).

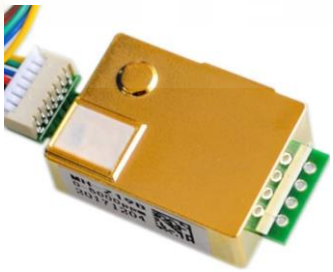


Figura 35 – Sensor CO2 – MHZ-19B [37]

- SPS30: Sensor PM (Fig. 36).

Figura 36 – Sensor de partículas – SPS30 [38]

- BME280: Sensor meteorológico (Fig. 37).

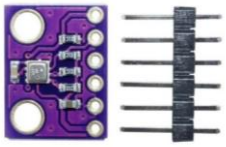


Figura 37 – Sensor atmosférico – BME280 [39]

3.3.2. PROTOTIPO EN PROTOBOARD

Para el caso del prototipo en *protoboard*, se va a necesitar componentes que ya tengan su propio módulo o placa de desarrollo que permita insertarlo en la placa de prototipado fácilmente.

Los componentes que se van a utilizar en este apartado serán los siguientes:

PROTOBOARD

Como no podría ser de otra forma, se va a necesitar mínimo una placa de prototipado (Fig. 38), o protoboard, para colocar los componentes en la placa, realizar el montaje y, posteriormente, la programación.

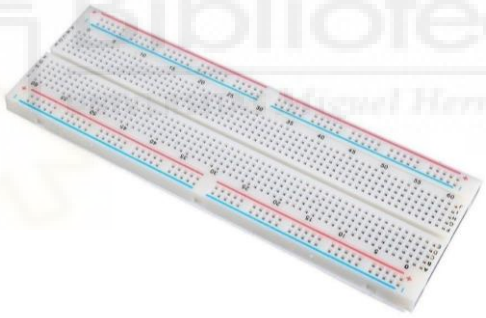


Figura 38 – Placa de prototipado [40]

CABLES

Los cables (Fig. 39) serán esenciales para realizar el conexionado de los componentes entre sí. Los que se muestran en la siguiente figura son de tipo Dupont que permiten hacer el conexionado de una manera más cómoda, puesto que no es necesario cortarlos ni pelarlos como en otros casos.

Figura 39 – Cables [41]

PLACA DE DESARROLLO ESP32

Como ya se comentó en el apartado de fundamentos teóricos, microcontroladores, actualmente en el mercado disponemos de unos microcontroladores de muy bajo coste con módulo WIFI incorporado que están siendo ampliamente utilizados en el mundo del IoT.

El microcontrolador que se utilizará en este proyecto es el ESP32 (Fig. 40). Este microcontrolador es realmente un SoC, por sus siglas en inglés *System on Chip*, puesto que incluye la gran mayoría de los periféricos que se encuentran en un ordenador en un solo chip. Este SoC es de bajo coste y bajo consumo, está fabricado por la compañía Espressif y tiene unas características muy impresionantes para su precio. Dispone de conectividad Bluetooth, además de wifi. Además, este tiene una potencia sorprendente alta comparado con, por ejemplo, un dispositivo Arduino UNO.

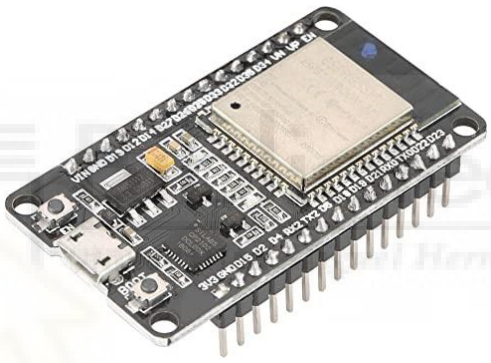


Figura 40 – Placa de desarrollo ESP32 [42]

MÓDULO MICROSD

Será necesario un módulo que permita conectar una memoria microSD al ESP32 (Fig. 41). Esta memoria se utilizará mediante el protocolo SPI. Esta memoria será muy importante para realizar todo el guardado de los datos leídos por los sensores. Se puede conectar tanto a 5V como a 3,3V.

Figura 41 – Módulo microSD [43]

MÓDULO RTC – DS3231

Se utilizará un módulo RTC (Fig. 42) con el integrado DS3231 y una pila que le permita mantener la hora actual ante pérdidas de electricidad o de la conexión a internet. En el mercado existen muchas alternativas que realizarían la misma función. Los puntos buenos de este modelo son que es muy preciso, no le afectan prácticamente los cambios de temperatura y, además, utiliza el protocolo I2C. Se puede conectar tanto a 5 V como a 3,3 V. Aunque como ya se ha comentado anteriormente será necesaria una pila, o una fuente externa, para mantener la hora.

Figura 42 – Módulo RTC DS3231 [44]

3.3.3. PROTOTIPO EN PCB

Para el prototipo en la PCB se deberá de extraer de cada uno de los módulos y placas de desarrollo los componentes que realmente vayamos a necesitar, de esta manera se podrá utilizar los integrados que realmente se necesiten y se conseguirá tener este proyecto en mucho menos espacio.

Además, se ha decidido utilizar la gran mayoría de los componentes del tipo SMD o de montaje superficial, los cuales son componentes más pequeños para tratar de reducir tamaño del dispositivo.

En los siguientes apartados se va a describir en profundidad el funcionamiento de cada uno de los elementos. Además, podremos ver el identificador que le da el fabricante en la lista de componentes del anexo 5 y su precio en el apartado de presupuesto.

PLACA DE DESARROLLO ESP32

En primer lugar, de la placa de desarrollo del ESP32 comentada anteriormente se deberá extraer los siguientes elementos:

ESP-WROOM-32

Este es el integrado o chip que contiene el microcontrolador (Fig. 43) con sus periféricos internos y antena para wifi y bluetooth.

Figura 43 – Módulo ESP32-WROOM-32 [45]

ALIMENTACIÓN

CONECTOR

Se utilizó un conector micro USB (Fig.44) para alimentar la placa que diseñaremos. Se ha elegido el de la figura 44 debido a que tiene unas patillas de inserción que le dan más rigidez, esto es importante puesto que este conector deberá aguantar la conexión y desconexión del cable que lo alimente.

Figura 44 – Conector Micro-USB [46]

REGULADOR DE TENSIÓN

Puesto que la PCB que vamos a diseñar se alimentará a 5V y tenemos elementos como necesitan alimentarse a 3,3 V, será necesario utilizar un regulador de tensión que baje la tensión.

Para este tipo de propósitos se suele utilizar un regulador de baja caída (Fig. 45), LDO, que viene de las siglas en inglés de *Low Drop Out*. Esta forma de regular es muy útil cuando el voltaje de alimentación está muy cerca del voltaje de salida. La principal característica de estos reguladores es su bajo precio, la sencillez y alta eficiencia.

Figura 45 – LDO [47]

ELEMENTOS DE PROTECCIÓN

Es importante incluir elementos de protección a la PCB para en casos de problemas o errores a la hora de alimentar no rompamos el resto de los componentes. Los integrados que se han incluido para proporcionar protección son:

- **TVS**

El diodo TVS (Fig. 46), que viene de las siglas en inglés de *transient-voltage-suppression*, es uno de los componentes de protección más utilizado debido a que permite proteger a la electrónica de sobrevoltajes en la alimentación, normalmente debidos a los transitorios.

Figura 46 – TVS [46]

- **ESD**

El diodo ESD (Fig. 47), de las siglas *Electrostatic Discharge*, es un tipo de diodo TVS que está especializado en suprimir las tensiones debidas a descargas electrostáticas.

Figura 47 – ESD [47]

- **SCHOTTKY**

El diodo Schottky (Fig. 48), llamado así en honor del físico Walter H. Schottky, es una variación al diodo normal que tiene una tecnología que tiene, con respecto al diodo estándar, las siguientes características: permite conmutaciones muy rápidas entre estados de conducción y tiene una tensión umbral muy baja, lo que provoca que la tensión de caída sea pequeña. Este componente permitirá proteger tanto la fuente de alimentación como el dispositivo de corrientes inversas.

Figura 48 – Diodo Schottky [48]

ILUMINACIÓN

Se ha incluido un led (Fig. 49) que se iluminará cuando exista una corriente que circula por él. Esto permitirá saber cuándo se está alimentando el dispositivo.

Figura 49 – Diodo Led [49]

PROGRAMADOR EXTERNO

Se utilizará un módulo externo que permita comunicarnos con el microcontrolador, estos se suelen llamar programadores FTDI (Fig. 50), de la empresa *Future Technology Devices International Limited* que está especializada en tecnologías USB. Este módulo convertirá la comunicación USB del ordenador en RS232 o TTL serial que permitirá comunicarnos con el microcontrolador.

Manteniendo este módulo externo permite reducir la cantidad de componentes por mm^2 en la PCB. Además, puesto que este componente se va a utilizar solamente una vez para cargar el programa, no tendría sentido incorporarlo en la PCB definitiva.

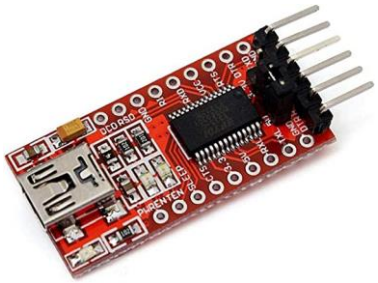


Figura 50 – Programador FTDI [50]

MÓDULO RTC

Del módulo RTC anteriormente nombrado se van a extraer los siguientes componentes:

DS3231SN

Se incorporará el chip DS3231SN (Fig. 51), este el componente principal del módulo y es el que actúa como RTC.

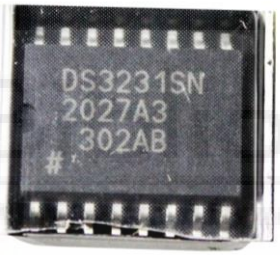


Figura 51 – DS3231SN [51]

PORTAPILA Y PILA

Será necesario tener una pila y su correspondiente portapilas (Fig. 52), que mantenga alimentado el integrado del RTC para que sea capaz de mantener la fecha y hora ante perdidas en la alimentación.

Figura 52 – Portapilas [52]

MÓDULO microSD

Del módulo microSD antes mostrado, solamente se necesitará extraer el conector (Fig. 53) que nos permita introducir y extraer fácilmente la memoria microSD.

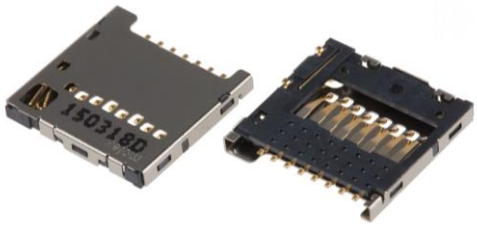


Figura 53 – Conector para MicroSD [53]

COMPONENTES BÁSICOS

Por último, además, se han utilizado componentes básicos como resistencias y condensadores. Estos componentes se han elegido de montaje superficial, o SMD, para que ocupen poco espacio en la placa.

RESISTENCIAS

Se utilizarán las resistencias (Fig. 54) para múltiples propósitos como, por ejemplo, limitar la corriente que pase por el led y poner resistencias de *pull-up*, necesarias para protocolos de comunicación como I2C y SPI.

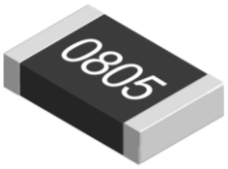


Figura 54 – Resistencias [54]

CONDENSADORES

Los condensadores (Fig. 55), por otro lado, se pondrán según indique el fabricante y como condensadores de desacoplo, que permiten desacoplar la alterna de la alimentación y de esta manera alimentar los componentes con una tensión limpia.

Figura 55 – Condensadores [55]

3.4. HERRAMIENTAS INFORMÁTICAS

Para el caso de las herramientas informáticas o *software*, se han utilizado las siguientes para llevar a cabo el proyecto:

3.4.1. MATLAB

Como se puede ver en el anexo 6 sobre la regresión multilineal se ha utilizado Matlab para obtener la ecuación de dependencia de la medida leída por el sensor de PM por factores meteorológicos.

Como se ha comentado en este apartado se podría haber utilizado cualquier otro programa con el que nos sintamos familiarizados como Python, Octave, R, etc.

3.4.2. EXCEL

En el estudio realizado para la comparativa de los sensores PM y la representación de los datos de la regresión multilineal se ha utilizado Excel. Este programa es muy intuitivo y fácil de utilizar, para realizar las representaciones y calcular los valores de R^2, de la pendiente e incluso los errores RMSE y MAE.

3.4.3. DIAGRAMAS

3.4.3.1. DIAGRAMAS.NET

La herramienta informática que se ha utilizado durante todo el proyecto para realizar diagramas ha sido diagramas.net que realmente es una web a la que se puede acceder desde cualquier navegador.

Entre los diagramas realizados se pueden destacar los realizados para el esquema base y final, las figuras 23 y 24. Además, del diagrama que se verá en el apartado del funcionamiento en bucle del microcontrolador.

3.4.3.2. FRITZING

Para realizar la representación del conexionado de los componentes en la *protoboard* se ha utilizado la aplicación llamada Fritzing. Gracias a este programa se ha podido hacer el diagrama que aparece en la sección montaje en *protoboard* en la figura 63.

3.4.4. PROGRAMACIÓN

En este caso, el microcontrolador de bajo coste elegido para conectar todos los sensores es el ESP32, el cual puede ser programado en C/C++ a través del entorno de desarrollo del fabricante Espressif IDE y del entorno de desarrollo de Arduino. Además de otras alternativas como MicroPython donde se programaría en un lenguaje similar a Python.

En este caso, se ha considerado que la mejor alternativa es el entorno de Arduino IDE. Este entorno es muy útil puesto que existe mucha documentación online, muchos y muchos ejemplos que permitirán realizar este proyecto en un tiempo más reducido.

La aplicación, que se ha utilizado para la programación del dispositivo, es Visual Studio Code (Fig. 56), con la extensión de Platformio, que permite programar microcontroladores con el entorno de desarrollo Arduino. Este programa es muy utilizado actualmente en muchos los ámbitos profesionales, debido a la facilidad que ofrece el programa a la hora de instalar herramientas, personalizar tu entorno de desarrollo y vincular la aplicación con plataformas externas.

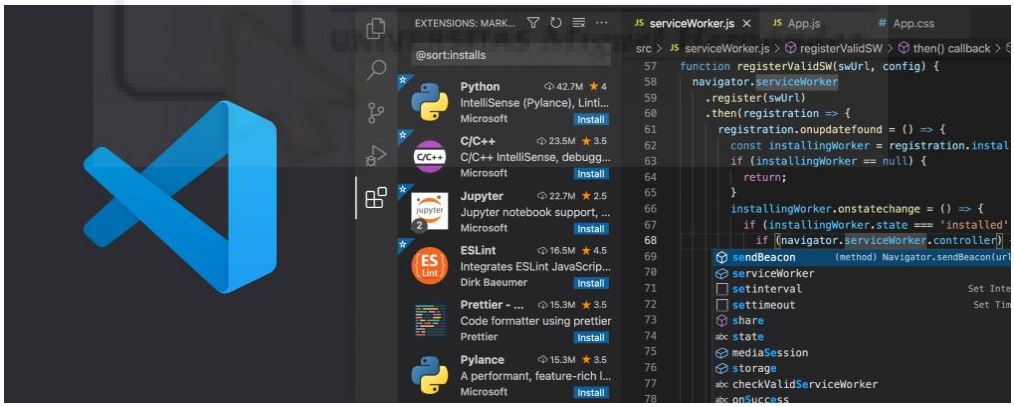


Figura 56 – Aplicación Visual Studio Code [56]

3.4.5. PLATAFORMAS IoT

Como se ha comentado en el apartado del estado del arte, internet de las cosas (IoT), las plataformas IoT permiten dotar a dispositivos electrónicos normales llevar a cabo funciones muy útiles. Para este proyecto se ha decidido utilizar a dos servicios en línea IoT. Estos permitirán cumplir el objetivo de que los datos del dispositivo se puedan consultar de forma remota y poder analizarlos sin la necesidad. Ambos servicios permitirán subir los datos a internet y poder acceder a este servicio desde cualquier dispositivo conectado a la red. Cada uno de ellos ofrece alguna ventaja en particular como se verá a continuación:

3.4.5.1. THINGSPEAK

ThingSpeak (https://thingspeak.com/, Fig. 57), es una plataforma que permite subscribir y publicar información mediante MQTT. La principal razón para utilizar esta plataforma es que almacena en la nube, visualiza y permite exportar los datos que recibe. Esta plataforma puede almacenar hasta 80 millones de datos, con una licencia gratuita. Además, permite generar archivos *.csv*, con la información recibida para poder analizarlos, a posteriori, desde cualquier hoja de cálculo como Excel. Esta plataforma pone también a disposición del usuario herramientas implementadas de Matlab, *Matlab Toolboxes*, que permiten hacer un procesado y análisis de los datos, en el momento que se reciben. El dispositivo *PurpleAir*, comentado en la introducción, almacena y procesa los datos a partir de esta plataforma.

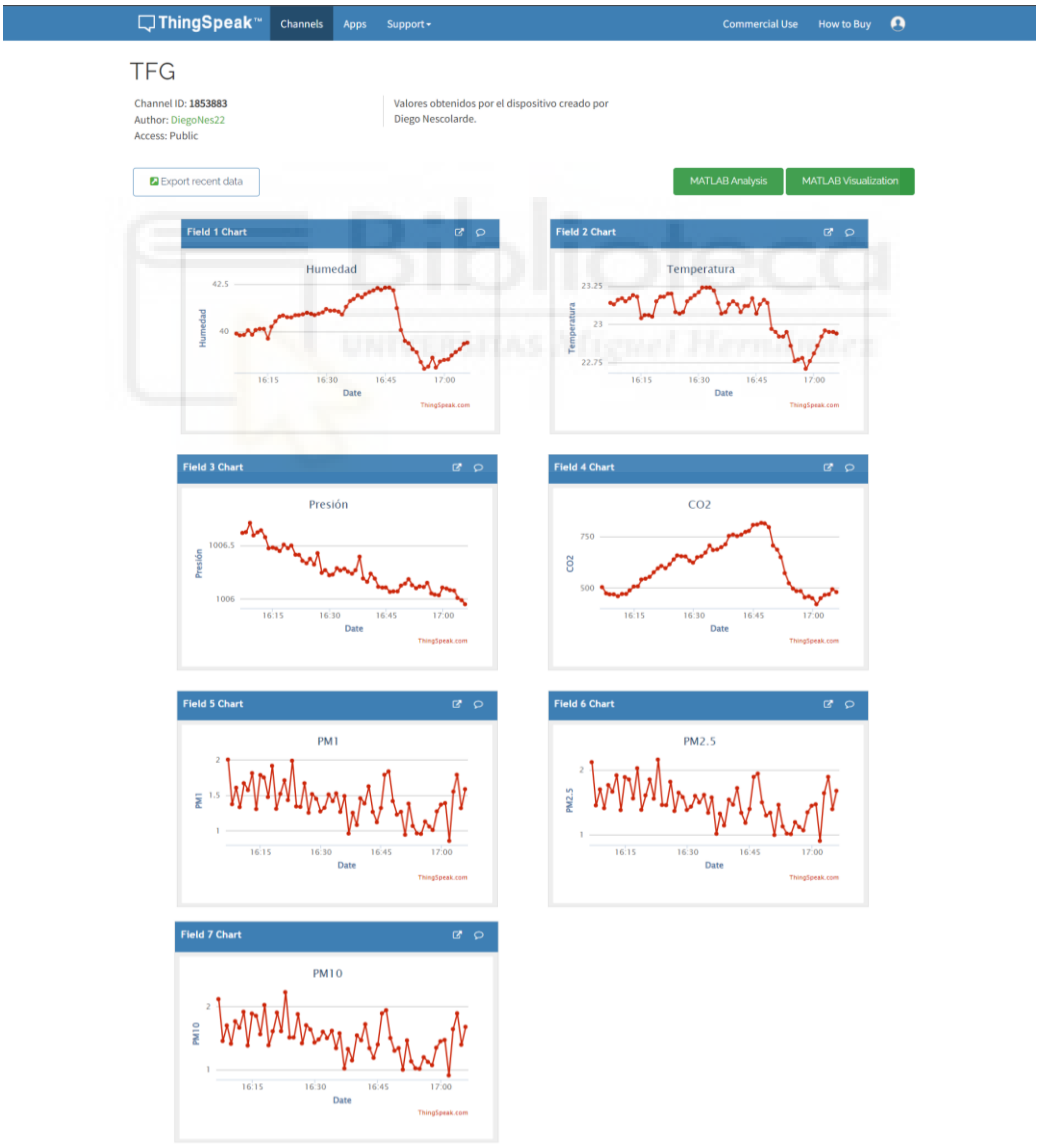


Figura 57 – ThingSpeak en un navegador web. En cada una de las gráficas se mostraría la concentración obtenida por cada sensor.

3.4.5.2. BLYNK

Blynk (https://blynk.io/ , Fig. 58), es otra plataforma y aplicación móvil que permite subscribir y publicar información mediante MQTT y por lo tanto podemos visualizar los datos en tiempo real sin necesidad de estar cerca del dispositivo. La plataforma de *Blynk*, al contrario que ocurre para *ThingSpeak*, está muy bien integrada en dispositivos móviles y permite visualizar los datos desde los mismos. Permite elegir entre muchas opciones a la hora de representar los datos.

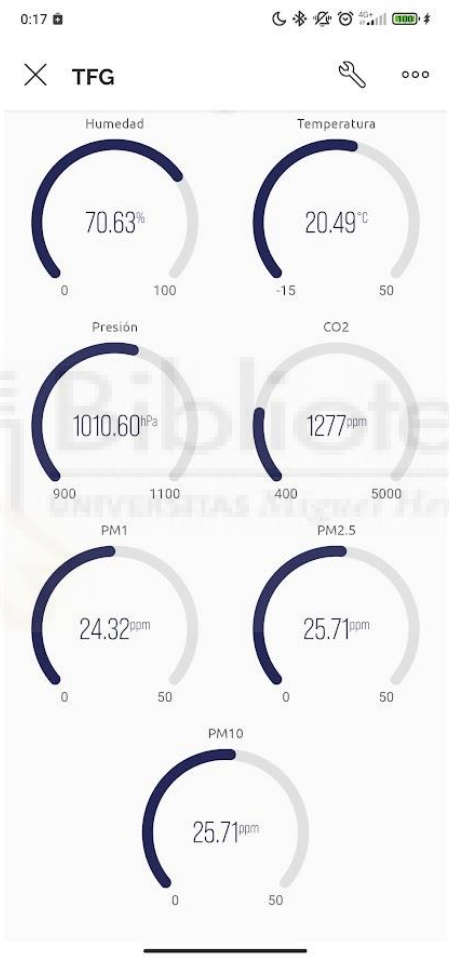


Figura 58 – Blynk en un dispositivo móvil

3.4.6. DISEÑO DE PCB

El programa elegido a la hora de realizar el diseño de la placa impresa, o *PCB (Printed Circuit Board)*, ha sido la aplicación informática llamada Eagle. Este programa ofrece muchas herramientas para facilitar el proceso de realizar cualquier diseño de una PCB.

Sin duda, existen otras muchas otras alternativas a Eagle, que permitirían obtener un resultado muy similar o igual, como Altium o Kicad.

3.4.7. DISEÑO 3D

Para diseñar la carcasa del dispositivo y poder darle un aspecto más profesional al dispositivo se va a utilizar una aplicación llamada Fusión 360.

Esta aplicación permite importar el diseño de la placa realizada con Eagle y facilitar el diseño de la carcasa para la misma (Fig. 59).

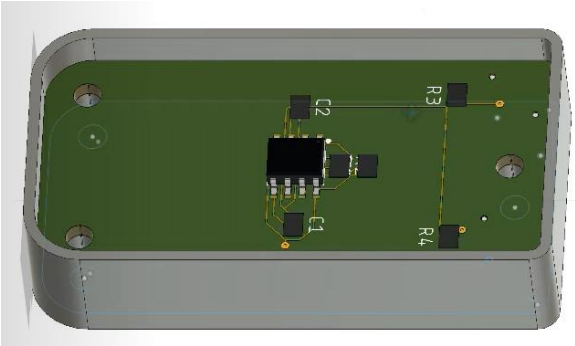


Figura 59 - PCB en Fusion 360 [57]

4. DISEÑO E IMPLEMENTACIÓN DEL DISPOSITIVO

El diseño del dispositivo deberá comenzar con el montaje físico del sistema para, posteriormente, realizar la programación del microcontrolador. Una vez se haya conseguido la comunicación entre todos los elementos y realizar que cumpla el objetivo del esquema final se procederá a diseñar una PCB y una carcasa 3D para terminar el dispositivo.

4.1. MONTAJE EN PROTOBOARD

Se comenzará haciendo el conexionado físico de todos los componentes del dispositivo en una *protoboard o breadboard* (placa de prototipado, figura 38), esta placa permitirá insertar los pines de todos los componentes e insertar los cables para hacer las conexiones necesarias para la comunicación entre el microcontrolador y cada uno de los sensores.

Para realizar el conexionado deberemos estudiar cada uno de los componentes, comprobar la información disponible en las hojas de características y, además, conocer cuál es el protocolo de comunicación que necesitan para comunicarse con el microcontrolador, posteriormente elegiremos uno de los pines disponibles del microcontrolador de este tipo de comunicación y realizaremos las conexiones en función de la información ofrecida por el fabricante de cada dispositivo.

En primer lugar, se deben buscar cuales son los pines de salida (Fig. 60) de la placa de desarrollo del microcontrolador, en este caso, el ESP32 DEVKIT V1 - DOIT:

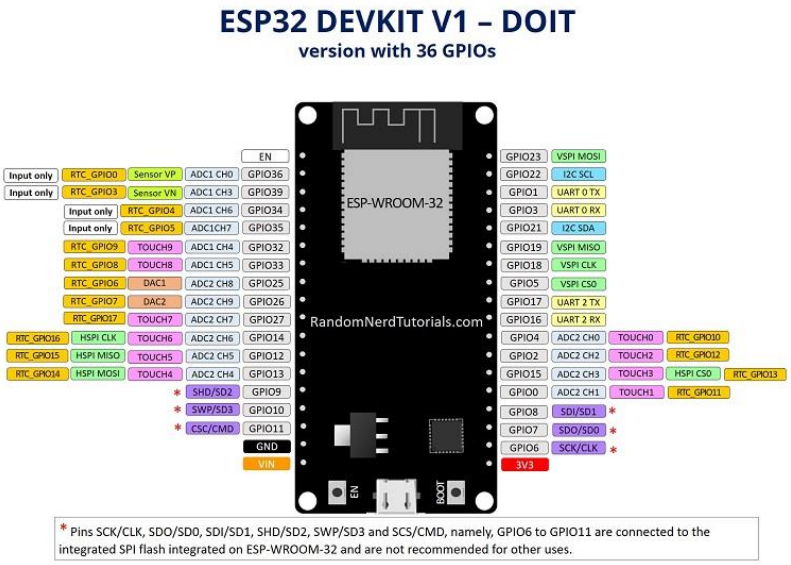


Figura 60 – Pines de salida ESP32 [58]

Por otro lado, de los componentes ya se conocen cuáles son los protocolos de comunicación que utilizan para comunicarse cada uno de ellos y la tensión a la que tienen que estar alimentados. Está información se encuentra disponible en el apartado de componentes.

En el caso del BME280, el RTC DS3231 y la pantalla oled SSD1106, se ha escogido los únicos puertos I2C disponibles en el microcontrolador. Estos pines son el 21 y el 22, SDA y SCL respectivamente. Como se puede observar, el protocolo I2C permite conectar múltiples dispositivos, en este caso tres, mediante solamente dos pines de comunicación.

Por otro lado, el sensor de CO_2, el MH-Z19B, utiliza el protocolo UART de comunicación, al cual se le puede asignar la gran mayoría de pines disponibles en el microcontrolador y, posteriormente, se configurará, desde el código del programa que gobierne el microcontrolador, cual es el pin de transmisión (TX) y cuál es el de recepción (RX). En este caso, se ha elegido los pines 33 y 25, RX y TX respectivamente.

El sensor de calidad del aire, SPS30 puede configurarse para que su comunicación sea a través de UART o I2C, en este caso se ha preferido configurarlo como UART porque permite aislar este sensor a unos pines dedicados y evitar posibles conflictos en la comunicación que se pueden producir en un protocolo de comunicación I2C, donde se tienen muchos elementos conectados. Se ha elegido los pines 16 y 17, RX y TX respectivamente.

La memoria microSD, donde se van a guardar los datos, se va a comunicar con el microcontrolador mediante el protocolo SPI. Por lo tanto, escogemos los pines disponibles en el microcontrolador para utilizar SPI, los pines 5, 30, 31, 37, que son CS, SCLK, MISO y MOSI respectivamente.

Por último, los dos botones que están disponibles para controlar el dispositivo estarán conectados a cualquier pin digital que permita comprobar si el botón esta pulsado o no, comprobando si la señal es alta o baja. Para los botones se suele utilizar una resistencia de *pull-up* o *pull-down*, como se explicaba anteriormente. Afortunadamente, se puede prescindir de esta resistencia debido a que se puede configurar internamente el microcontrolador para utilizar un *pull-up* interno.

Finalmente, el diagrama de la placa de prototipado con los periféricos conectados al microcontrolador es el de la figura 61.

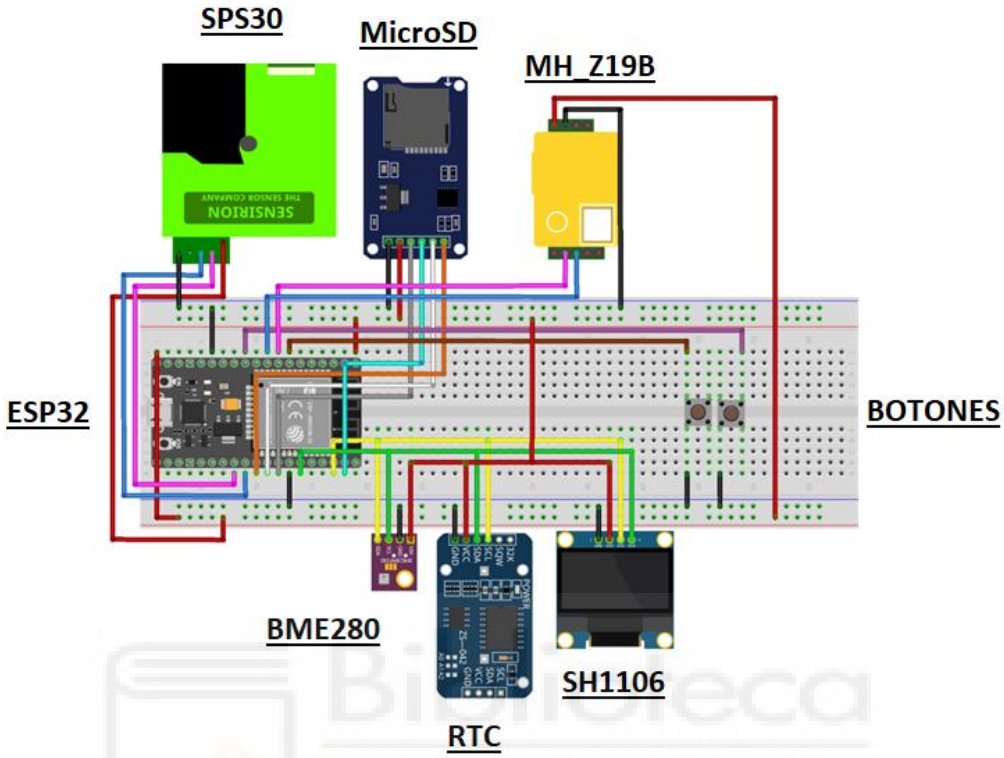


Figura 61 – Diagrama de conexión en protoboard

Para realizar este conexionado deberemos tener en cuenta la tensión de alimentación de los periféricos, el tipo de comunicación y los pines disponibles del microcontrolador, como hemos comentado anteriormente.

4.2. PROGRAMACIÓN

En primer lugar, en el anexo 1 se puede encontrar tanto una explicación en profundidad como el código completo del programa. Aunque en este apartado se va a explicar de forma general el funcionamiento del dispositivo, AirKnowledge.

Como se ha comentado en el apartado de fundamentos teóricos tenemos dos partes esenciales en el código: el inicio del programa y el bucle del programa. Estas partes se van a comentar a continuación:

INICIO

Al iniciar el dispositivo, el microcontrolador establecerá la comunicación con los sensores, el RTC, la microSD, la pantalla e intentará conectarse a internet.

Mientras el microcontrolador se encuentre estableciendo la conexión con todos los periféricos, la pantalla mostrará una transparencia donde aparece el nombre del autor del dispositivo y el logo del grupo de investigación con el que se ha hecho el proyecto (Fig. 62).

Figura 62 – Logo LCA (Laboratorio de Contaminación Atmosférica)

BUCLE

Para el caso del bucle, se ha realizado un diagrama (Fig. 63). En este, se representa el orden en el que se ejecutan cada una de las funciones que realizará el microcontrolador.

Por otro lado, de la frecuencia con la que se accede a la función de guardado en memoria, microSD, depende de las configuraciones elegidas por el usuario. Sin embargo, para el resto de los casos viene prefijados unos tiempos en el código del programa en función de las recomendaciones de los fabricantes de los sensores, etc.

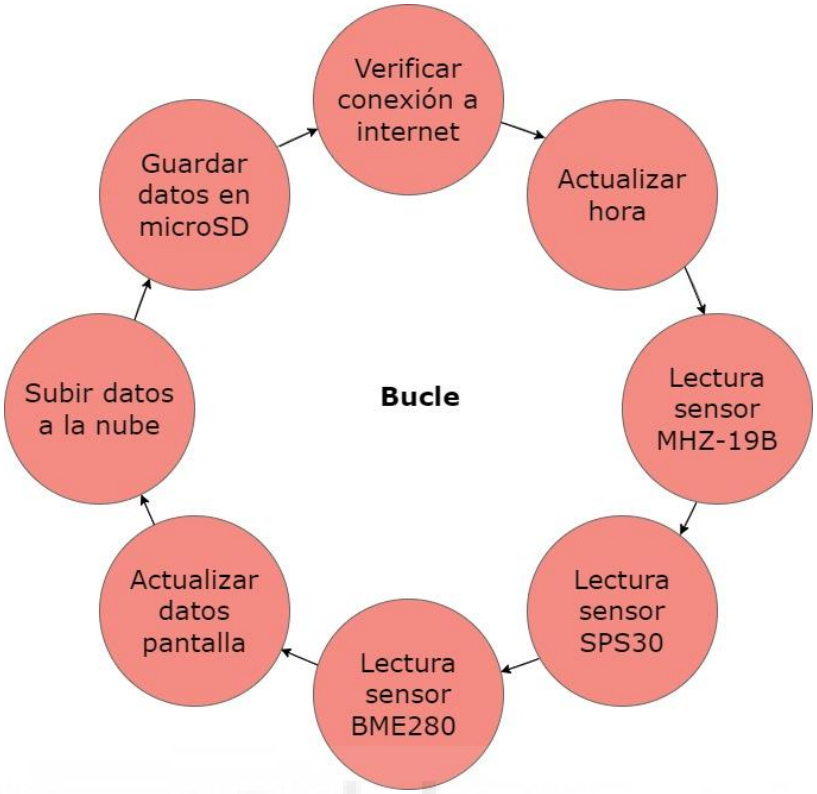


Figura 63 – Diagrama del bucle

A continuación, se va a explicar brevemente en que consiste cada una de las actividades que se pueden ver en el diagrama de la figura 63.

Verificar conexión a internet

Para poder acceder a los servicios, tanto para obtener la hora como para subir los datos a la nube, el dispositivo deberá conectarse a internet e ir verificándolo periódicamente.

Actualizar hora

El protocolo en línea llamado NTP (*Network Time Protocol*) es crucial para poder tener el dispositivo en fecha y hora para el correcto guardado de los datos, teniendo esta información temporal. Aunque, en caso de perder la conexión a internet, se dispone de un sistema RTC, muy útil para mantener la hora y fecha en caso de que el dispositivo se apagara.

Lectura sensores

Para cada sensor será necesario preparar un código que permita la lectura de este. Aunque, gracias a disponer de las librerías preparadas por los fabricantes de los sensores, será posible realizar la lectura de estos de una manera más abstracta, menos compleja y más legible. De esta manera, no se va a necesitar preparar una comunicación bit a bit para leer los sensores como haríamos si no se dispusiese de las librerías.

Actualizar datos pantalla

Puesto que la pantalla de la que se dispone no es muy grande, el mejor método que se ha pensado para mostrar los datos de los sensores es en tres transparencias distintas que van apareciendo en la pantalla rotativamente cada 5 segundos.

La primera transparencia muestra los datos meteorológicos y la hora actual (Fig. 64).

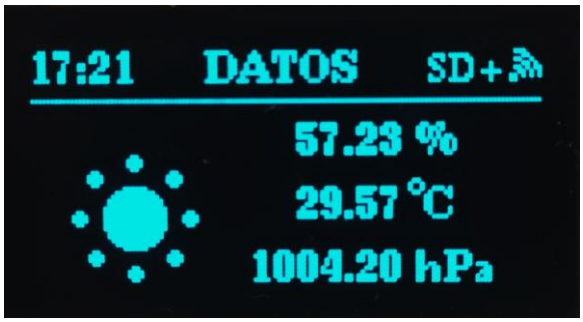


Figura 64 – Pantalla datos meteorológicos

La segunda transparencia muestra los datos de dióxido de carbono (CO_2) y la hora actual (Fig. 65).

Figura 65 – Pantalla datos CO_2

La tercera transparencia muestra los datos de la materia particulada y la hora actual (Fig. 66).

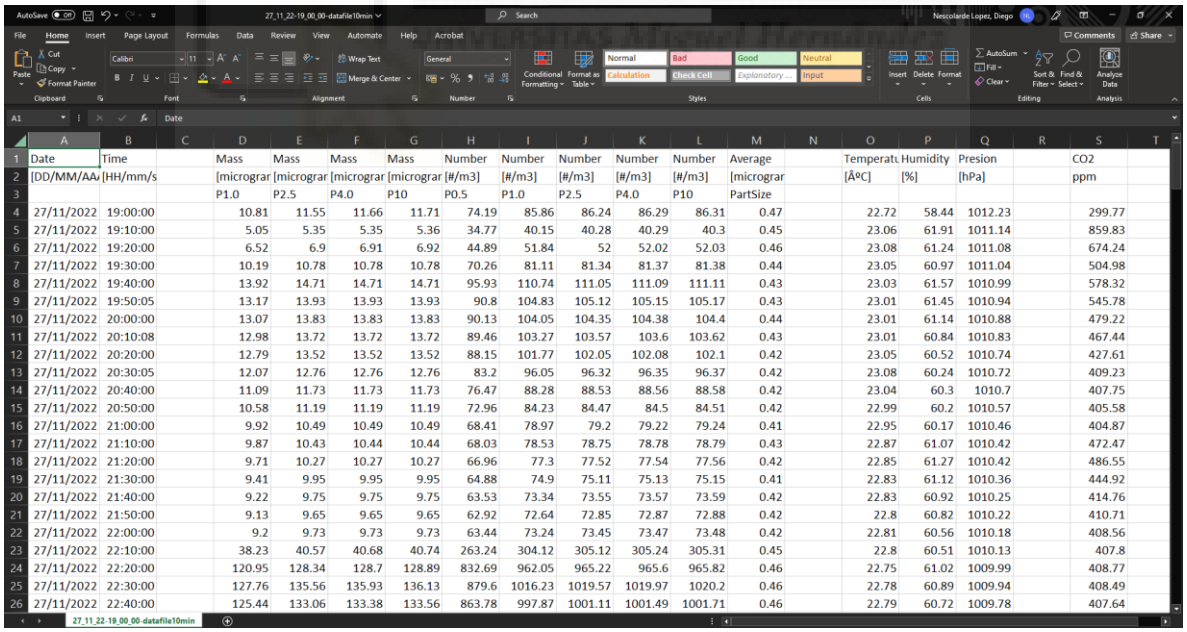
Figura 66 – Pantalla datos PM

Subir datos a la nube (IoT)

Como dispositivo IoT, una de las cosas más importantes será que este dispositivo sea capaz de acceder a servicios en línea que permitan acceder a los datos desde la nube y en tiempo real. Como se ha comentado en el apartado de herramientas informáticas, las plataformas IoT que se van a utilizar son *ThinkSpeak* y *Blynk*. Ambas funcionan mediante el protocolo MQTT, mencionado en el apartado de fundamentos teóricos.

Guardar datos en microSD

Para el guardado de los datos en la memoria microSD se va a hacer generando un archivo *.csv* (valores separados por comas) donde se almacenen los datos en filas y columnas ordenadamente para poder analizarlo posteriormente. En las columnas tendremos cada uno de los valores medidos por los sensores y las filas separan cada medida temporalmente, se guarda la fecha y hora en la que se realiza la medida. Este archivo *.csv* generado, donde se guarden todos los datos de las medidas, se puede abrir con cualquier hoja de cálculo como Excel (Fig. 67), con un básico bloc de notas o se puede importar a cualquier entorno como *R* o *Matlab*.



	Date	Time	Mass	Mass	Mass	Mass	Number	Number	Number	Number	Number	Average	Temperatu	Humidity	Presion	CO2
2	[DD/MM/AA	[HH/mm/s	[micrograr	[micrograr	[micrograr	[micrograr	[#/m3]	[#/m3]	[#/m3]	[#/m3]	[#/m3]	[micrograr	[ÂºC]	[%]	[hPa]	ppm
3			P1.0	P2.5	P4.0	P10	P0.5	P1.0	P2.5	P4.0	P10	PartSize				
4	27/11/2022	19:00:00	10.81	11.55	11.66	11.71	74.19	85.86	86.24	86.29	86.31	0.47	22.72	58.44	1012.23	299.77
5	27/11/2022	19:10:00	5.05	5.35	5.35	5.36	34.77	40.15	40.28	40.29	40.3	0.45	23.06	61.91	1011.14	859.83
6	27/11/2022	19:20:00	6.52	6.9	6.91	6.92	44.89	51.84	52	52.02	52.03	0.46	23.08	61.24	1011.08	674.24
7	27/11/2022	19:30:00	10.19	10.78	10.78	10.78	70.26	81.11	81.34	81.37	81.38	0.44	23.05	60.97	1011.04	504.98
8	27/11/2022	19:40:00	13.92	14.71	14.71	14.71	95.93	110.74	111.05	111.09	111.11	0.43	23.03	61.57	1010.99	578.32
9	27/11/2022	19:50:05	13.17	13.93	13.93	13.93	90.8	104.83	105.12	105.15	105.17	0.43	23.01	61.45	1010.94	545.78
10	27/11/2022	20:00:00	13.07	13.83	13.83	13.83	90.13	104.05	104.35	104.38	104.4	0.44	23.01	61.14	1010.88	479.22
11	27/11/2022	20:10:08	12.98	13.72	13.72	13.72	89.46	103.27	103.57	103.6	103.62	0.43	23.01	60.84	1010.83	467.44
12	27/11/2022	20:20:00	12.79	13.52	13.52	13.52	88.15	101.77	102.05	102.08	102.1	0.42	23.05	60.52	1010.74	427.61
13	27/11/2022	20:30:05	12.07	12.76	12.76	12.76	83.2	96.05	96.32	96.35	96.37	0.42	23.08	60.24	1010.72	409.23
14	27/11/2022	20:40:00	11.09	11.73	11.73	11.73	76.47	88.28	88.53	88.56	88.58	0.42	23.04	60.3	1010.7	407.75
15	27/11/2022	20:50:00	10.58	11.19	11.19	11.19	72.96	84.23	84.47	84.5	84.51	0.42	22.99	60.2	1010.57	405.58
16	27/11/2022	21:00:00	9.92	10.49	10.49	10.49	68.41	78.97	79.2	79.22	79.24	0.41	22.95	60.17	1010.46	404.87
17	27/11/2022	21:10:00	9.87	10.43	10.44	10.44	68.03	78.53	78.75	78.78	78.79	0.43	22.87	61.07	1010.42	472.47
18	27/11/2022	21:20:00	9.71	10.27	10.27	10.27	66.96	77.3	77.52	77.54	77.56	0.42	22.85	61.27	1010.42	486.55
19	27/11/2022	21:30:00	9.41	9.95	9.95	9.95	64.88	74.9	75.11	75.13	75.15	0.41	22.83	61.12	1010.36	444.92
20	27/11/2022	21:40:00	9.22	9.75	9.75	9.75	63.53	73.34	73.55	73.57	73.59	0.42	22.83	60.92	1010.25	414.76
21	27/11/2022	21:50:00	9.13	9.65	9.65	9.65	62.92	72.64	72.85	72.87	72.88	0.42	22.8	60.82	1010.22	410.71
22	27/11/2022	22:00:00	9.2	9.73	9.73	9.73	63.44	73.24	73.45	73.47	73.48	0.42	22.81	60.56	1010.18	408.56
23	27/11/2022	22:10:00	38.23	40.57	40.68	40.74	263.24	304.12	305.12	305.24	305.31	0.45	22.8	60.51	1010.13	407.8
24	27/11/2022	22:20:00	120.95	128.34	128.7	128.89	832.69	962.05	965.22	965.6	965.82	0.46	22.75	61.02	1009.99	408.77
25	27/11/2022	22:30:00	127.76	135.56	135.93	136.13	879.6	1016.23	1019.57	1019.97	1020.2	0.46	22.78	60.89	1009.94	408.49
26	27/11/2022	22:40:00	125.44	133.06	133.38	133.56	863.78	997.87	1001.11	1001.49	1001.71	0.46	22.79	60.72	1009.78	407.64

Figura 67 – Ejemplo de archivo csv generado abierto en Excel

Interfaz de usuario

El dispositivo AirKnowledge, además de estar realizando acciones en bucle, estará esperando a que cualquier usuario utilice la interfaz física mediante el uso de los botones

disponibles permitiendo al usuario interactuar con el dispositivo y modificar la configuración de guardado de los datos entre otros aspectos.

Es posible elegir entre tres modos de guardado:

- Modo wifi: guardamos los datos sólo en la nube.
- Modo SD: guardamos los datos sólo en la tarjeta microSD.
- Modo SD+wifi: guardamos los datos en ambos lados.

Además, para cada uno de los modos se va a poder elegir ciertas opciones como la wifi a la que se quiere conectar el usuario o cada cuanto se quieren guardar los datos en la tarjeta microSD.

Botones

La mejor opción para configurar los botones es mediante interrupciones, este método permite que hasta que el botón no sea pulsado el procesador del microcontrolador se centre en el resto de las tareas pendientes y solo en caso de ser pulsado accederá a la función correspondiente. En la siguiente figura ilustrativa (Fig. 68) se presenta como funciona.

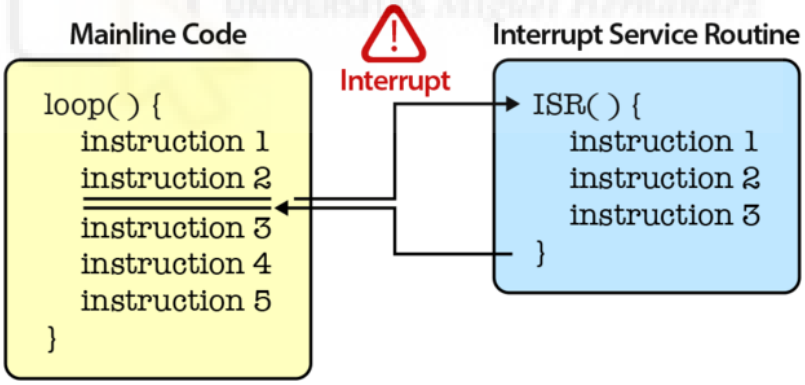


Figura 68 – Interrupciones. Una interrupción detiene la ejecución del código principal para realizar otra rutina [59].

Este método presenta muchas funcionalidades, entre las cuales destacamos:

- Tiene unas líneas de código que actúan como un sistema antirrebote hecho por *software*. Como bien sabemos los botones al ser un dispositivo mecánico su respuesta no es ideal, como se puede apreciar en la figura 69, y tenemos rebotes antes y después de pulsar el botón. Estos rebotes son debidos a las propias limitaciones mecánicas del botón y pueden provocar problemas dando falsas pulsaciones. Por esto es importante que estos rebotes sean filtrados.

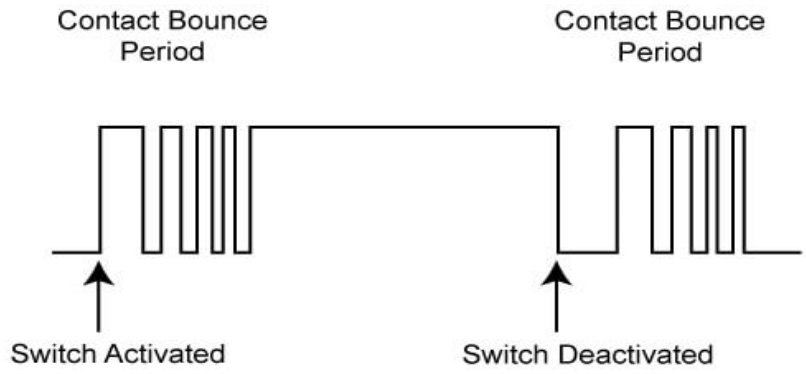


Figura 69 – Respuesta de un botón. Se aprecian los rebotes que aparecen cuando se presiona y cuando se suelta el dispositivo. [60]

- Se tiene que poder diferenciar si la pulsación del botón es larga o corta. En caso de ser una pulsación larga y encontrarse en la pantalla con los datos de los sensores, figura 64, 65 y 66, se accede a la pantalla del menú donde se puede cambiar entre los distintos modos de guardado de los datos del dispositivo (Fig. 70).

Figura 70 – Menú de modos. En este menú podremos cambiar hacia arriba o hacia abajo con los dos botones disponibles, pulsación corta, y seleccionaremos manteniendo cualquiera de ellos, pulsación larga.

Submenús de guardado

- Primer submenú para opciones de guardado en la microSD, en caso de haber elegido el modo SD o el modo WIFI+SD, en el menú de modos (Fig. 70), accederemos a un nuevo submenú (Fig. 71), que permite cambiar cada cuanto se quiere almacenar los datos en la microSD. Los datos que finalmente se guardan consisten en un promediado de todas las medidas realizadas por los sensores en el periodo de tiempo definido en el submenú de la figura 71.

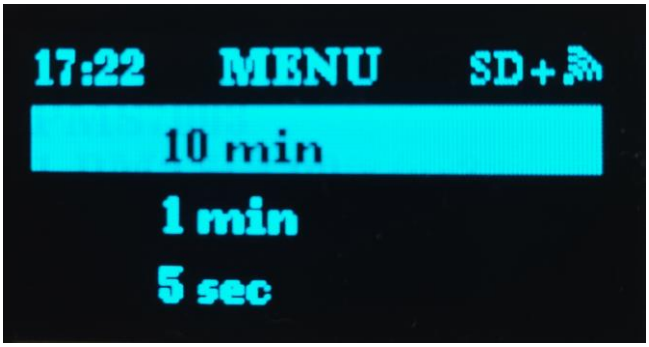


Figura 71 – Submenú microSD. En este submenú podremos cambiar hacia arriba o hacia abajo con los dos botones disponibles, pulsación corta, y seleccionaremos manteniendo cualquiera de ellos, pulsación larga.

- Segundo submenú con las opciones wifi, en caso de haber seleccionado en la pantalla de menú (Fig. 70), el modo WIFI o el modo WIFI+SD, accederemos al submenú wifi donde podremos seleccionar entre las redes wifi predeterminadas (Fig. 72). Actualmente, para poder modificar o añadir algún wifi habría que modificar el programa del dispositivo. En líneas futuras una de las mejoras a implementar sería facilitar el proceso de incorporar nuevas redes wifi al usuario desde una aplicación o página web.

Figura 72 – Submenú wifi - Podemos elegir las redes wifi disponibles. Tienen que haberse introducido a la hora de programar el microcontrolador.

- Al cambiar entre modos se verá cómo arriba a la derecha de la pantalla del dispositivo AirKnowledge, aparece el modo en el que se encuentra y en el caso del wifi, si no se pudiese conectar a la red seleccionada aparecerá el símbolo tachado podemos verlo en figuras anteriores como, por ejemplo, la figura 72.

4.3. ESQUEMÁTICO DE LA PCB

Como se comentaba en el apartado de herramientas informáticas, se va a utilizar el programa Eagle para hacer el diseño de la PCB, o placa de circuito impreso, para este prototipo. El primer paso para diseñar una PCB es hacer el esquemático de la misma.

En el anexo 3 se adjunta el esquemático completo. En los siguientes apartados se va a explicar cada una de sus partes y el porqué de ellas.

4.3.1. RESISTENCIAS DE PULL-UP

Como se ha comentado anteriormente en el apartado de fundamentos teóricos (resistencias pull-up y pull-down) y en el apartado de los componentes para la PCB, son necesarias unas resistencias *pull-up* para las comunicaciones I2C y SPI.

Por lo tanto, se necesitará una resistencia para cada uno de sus buses de comunicación (Fig. 73): SDA, SCL, CS, MISO, MOSI y SCLK. El valor de ésta no es tan importante y los valores estándar más utilizados son 10 kΩ y 4,7 kΩ. Se ha escogido finalmente el primero de los valores.

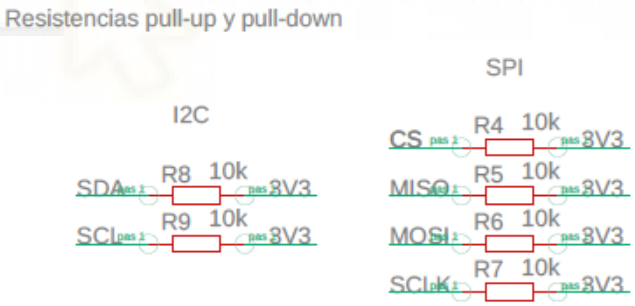


Figura 73 – Resistencias pull-up – Esquemático

4.3.2. CONEXIÓN DE SENSORES, PANTALLA, MICROSD Y RTC

En este caso, se deberá acceder, en primer lugar, al *datasheet*, hoja de características en castellano, de cada uno de los periféricos para saber cuál es la alimentación a la que deben ir conectados y a que pines corresponde cada cosa: comunicación, tierra o alimentación. En el apartado de elección de los sensores y en el apartado de componentes, se ha mencionado alguna de las características de estos como su alimentación y el protocolo de comunicación que utilizan.

Además, para garantizar una correcta alimentación se ha colocado un condensador de desacoplo de 0,1 µF para mejorar la estabilidad y desacoplar el posible ruido de la alimentación.

Para el caso del sensor meteorológico, el BME280 (Fig. 74), se debe alimentar a 3,3 V y la comunicación es I2C por lo que se conectará a los puertos SCL y SDA.

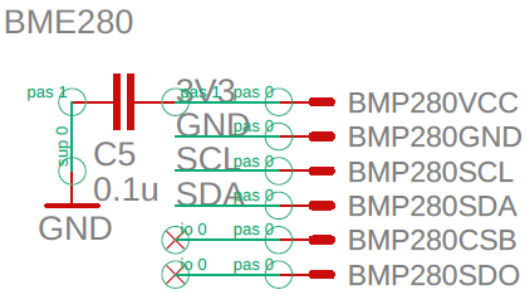


Figura 74 – BME280 – Esquemático de la conexión del sensor meteorológico

Para el sensor de CO_2, el MHZ-19B (Fig. 75), se debe alimentar a 5 V y la comunicación es mediante el protocolo UART. Se conecta mediante un conector del tipo JST con 7 pines.

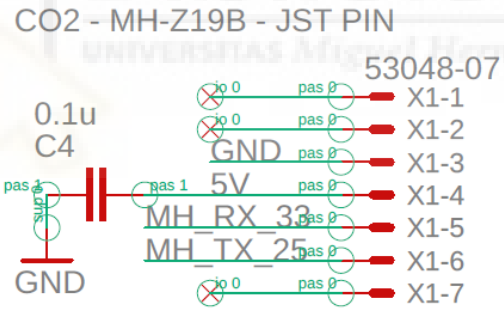


Figura 75 – MHZ-19B – Esquemático de la conexión del sensor de CO_2

El sensor de PM, SPS30 (Fig. 76), se debe alimentar a 5 V y la comunicación es mediante el protocolo UART. Se va a conectar mediante un conector del tipo ZH con 5 pines.

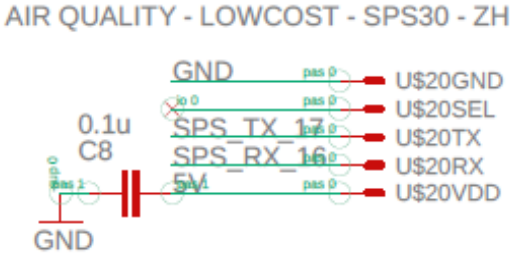


Figura 76 – SPS30 – Esquemático de la conexión del sensor de material particulado

La pantalla, modelo SH1106 (Fig. 77), necesita una alimentación de 3,3 V y la comunicación es mediante I2C.

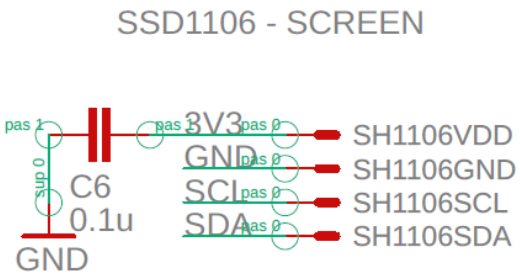


Figura 77 – SH1106 – Esquemático

Para el RTC, DS3231 (Fig. 78), se han seguido las indicaciones del fabricante y se han puesto los pines correspondientes en abierto y a tierra. Además, este integrado se debe alimentar a 3,3 V y la comunicación es mediante I2C. Por otro lado, este integrado necesita tener una pila en este caso, CR2023, para que en el caso de perdidas en la alimentación poder mantener la hora.

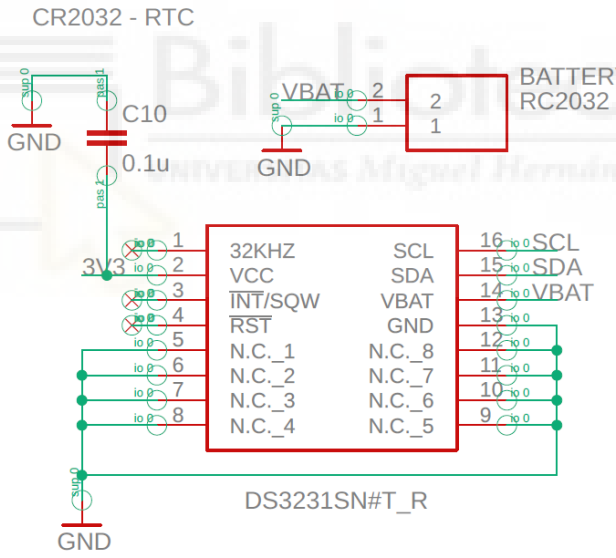


Figura 78 – DS3231 – Esquemático

Por otro lado, el soporte para la microSD tiene la conexión que se puede ver en la figura 79. Éste tiene que alimentarse a 3,3 V y se comunica mediante SPI. Se ha seguido las indicaciones del fabricante y se han puesto los pines que indica a tierra.

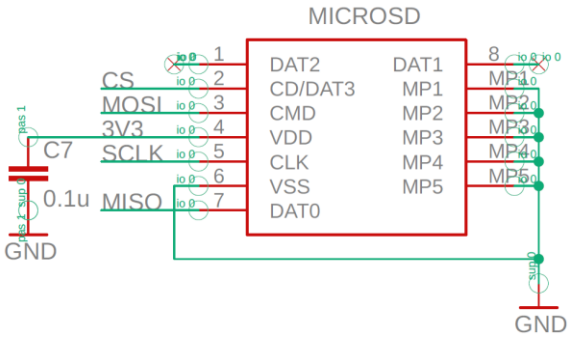


Figura 79 - microSD - Esquemático

4.3.3. PROGRAMADOR

Para programar el dispositivo se ha elegido no disponer de un componente en la placa de circuito impreso específico para ello, puesto que es un elemento que se va a utilizar solamente una vez, a la hora de cargar el programa, y por lo tanto incrementaría el coste y complejidad de la placa.

Como alternativa, se ha colocado unos pines para la programación del dispositivo los cuales están preparados para un programador estándar, FTDI, como el de la figura 50.

De esta manera, simplemente necesitaremos conectar este dispositivo a la placa y podremos programar el dispositivo.

En el esquemático es el de la figura 80, necesita una alimentación de 3,3 V y se debe conectar a los pines: reset, boot, RX0 y TX0, del microcontrolador.

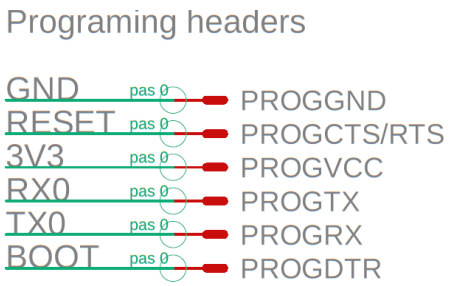


Figura 80 – Programador FTDI – Esquemático

4.3.4. ETAPA DE ALIMENTACIÓN

Este apartado sobre la etapa de alimentación se va a dividir en tres subapartados independientes. Esta división se ha realizado teniendo en cuenta el propósito de cada componente. Los tres subapartados son: conector micro USB, convertidor de tensión y componentes protección.

CONECTOR MICRO USB

En el caso del conector micro USB (Fig. 82) se va a utilizar solamente como alimentación del dispositivo y sacaremos de este conector solamente la tensión, PIN 1, y la tierra, PIN 5 y PIN6.

CONVERTIDOR DE TENSIÓN

Como se comentaba en el apartado componentes se va a usar un regulador de baja caída, a LDO (*Low Dropout*), que pase la tensión de entrada, 5V, a una tensión de salida de 3,3 V. Como vemos en la figura 81, para el regulador seleccionado el fabricante indica que se debe poner unos condensadores de 10 µF tanto a la entrada como a la salida.

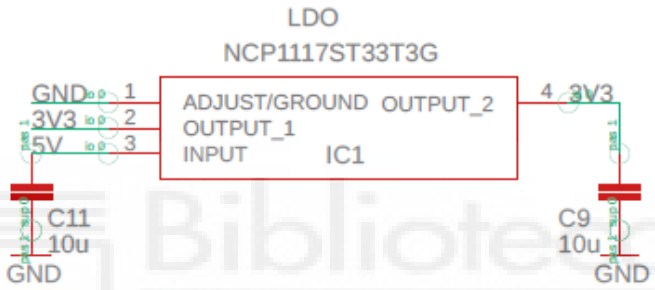


Figura 81 – Convertidor de tensión – Esquemático

PROTECCIÓN

Para este proyecto ha sido muy importante disponer de unas etapas de protección de alta calidad para evitar posibles daños en el resto de los componentes en caso de algún problema con la alimentación. En el apartado de los componentes veíamos los tres elementos que se van a utilizar en este proyecto: ESD, TVS y diodo Schottky.

Para el caso del TVS y ESD, estos sistemas en caso de detectar algún pico de tensión derivaran toda la tensión a tierra evitando el deterioro en algún componente. Estos diodos de protección deben ir conectados: con el ánodo a tierra y el cátodo a la alimentación (Fig. 82)

Mientras que, para el caso del diodo Schottky, su función es dejar el circuito en abierto en caso de detectar corrientes con dirección inversa, de esta manera evitaría que circule corriente en sentido contrario al deseado. En este caso este componente se colocaría en serie en la entrada, como se ve en la figura 82.

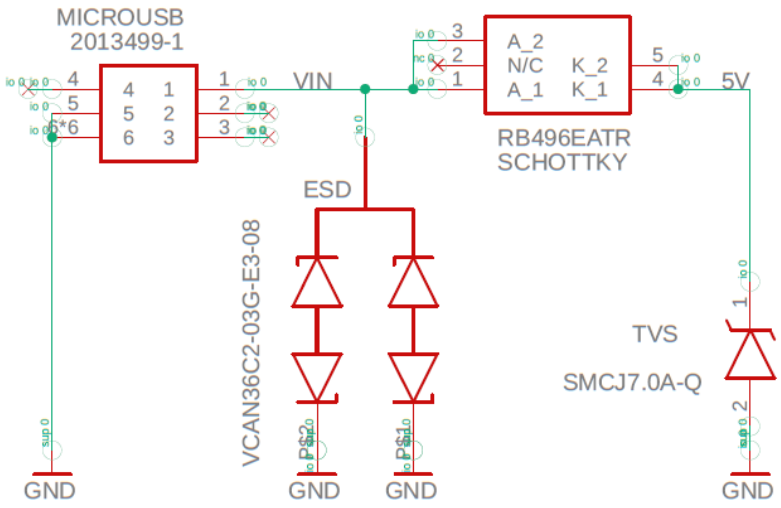


Figura 82 – Etapa de alimentación entrada – Esquemático de conexión del conector microUSB, ESD, TVS y del diodo Schottky.

4.4. DISEÑO DE LA PCB

En el anexo 4 se ha adjuntado el diseño de la PCB, o placa de circuito impreso. En los siguientes apartados se va a explicar porque el posicionamiento de cada elemento.

Se deberá tener en cuenta los conceptos básicos explicados en la introducción, fundamentos del diseño PCB.

Para la PCB de este dispositivo, las conexiones entre los componentes se han realizado mediante solamente dos capas, puesto que no se trata de un diseño muy complejo. Cuantas más capas tenga la PCB más pistas se podrán tener por mm^2, pero mayor será el coste de fabricación. Además, se ha intentado realizar el diseño manteniendo una relación entre facilidad de montaje y tamaño reducido.

Una de las cosas más importantes a la hora de hacer el diseño de una PCB es el posicionamiento de los componentes ya que esto permitirá reducir considerablemente la longitud de las pistas y facilitará realizar el conexionado entre los componentes.

Para este proyecto, se ha decidido posicionar todos los componentes SMD en la cara inferior y los componentes THT en la capa superior, para facilitar el posterior proceso de soldadura.

Por otro lado, a continuación, se comentarán los criterios que se han tomado para colocar algunos de los componentes de la placa:

POSICIONAMIENTO DE LOS COMPONENTES SMD

- El conector micro USB se ha posicionado al borde de la placa para poder conectarle el cable fácilmente.
- En cuanto, al microcontrolador puesto que dispone de una antena wifi y bluetooth ha sido importante posicionarlo al borde de la placa y sin ningún elemento, ni pista ni plano, en la zona superior para que no haya ningún tipo de interferencias y exista un buen sistema de recepción y emisión de la información.
- Para el convertidor de tensión, tipo LDO, se ha dejado un buen plano metálico donde el componente sea capaz de disipar el calor sin problemas.
- Para los condensadores de desacoplo, permiten, como su nombre indica, desacoplar la tensión alterna de la tensión continua. Se han colocado muy cerca de los pines de alimentación para que realicen su función correctamente.

POSICIONAMIENTO DE LOS COMPONENTES THT

- Para el caso de los sensores, se ha tratado de colocar los conectores para los sensores en los laterales de la placa.
- Por otro lado, para la pantalla y los botones, se han tratado de centrar para tener una buena disposición de estos y una buena estética del dispositivo.

OTRAS CONSIDERACIONES

- Se ha hecho un plano de masa que permita conectar los pines de los componentes a masa fácilmente y que, además, permitan una conexión buena y estable entre todas las masas del dispositivo.
- Se han seguido las reglas de diseño del fabricante de la PCB superando las distancias mínimas entre pads, pistas y vías. También, se han utilizado tamaños superiores a los límites de fabricación para el caso de las pistas y taladros.
- Además, se han seguido las recomendaciones mencionadas en el apartado de diseño PCB, de la introducción del proyecto.
- Se han hecho cuatro agujeros, uno en cada esquina de la placa para la posterior sujeción mecánica mediante tornillos. Esto es fácilmente visible en el anexo comentado anteriormente.

4.5. MONTAJE Y SOLDADO DE LA PCB

4.5.1. FABRICACIÓN DE LA PCB

La fabricación de placas de circuito impreso se realiza mediante el un conjunto de procesos industriales (Fig. 83). A continuación, voy a explicar el proceso simplificado de la fabricación:

1. Preparación del material: las láminas son cortadas en paneles. Los paneles, de cobre y substrato, se apilan y se adhieren uno encima del otro como vemos en la figura 83.
2. Taladro: se realizan los agujeros necesarios para componentes THT, posibles vías y agujeros de sujeción mecánica.
3. Metalización: los paneles se sumergen en una solución de cobre. Esta solución recubre las paredes de los agujeros uniendo la capa superior e inferior.
4. Generación del fotolito: se imprime el circuito diseñado en una lámina transparente llamada fotolito.
5. Fotolitografía: se transfiere el circuito diseñado mediante un fotolito. La exposición ultravioleta a una resina fotosensible hace que esta endurezca. Esta resina se deberá aplicar previamente a la superficie de cobre.
6. Ataque químico: Se inserta en una solución que elimina la resina fotosensible restante y se hace un ataque químico a la superficie no protegida.
7. Mascara de soldadura: se pone una máscara de soldadura que evita la corrosión y facilita el proceso de la soldadura.
8. Estañado: se depositará una pequeña capa fina de estaño que proteja al cobre de la posible oxidación.

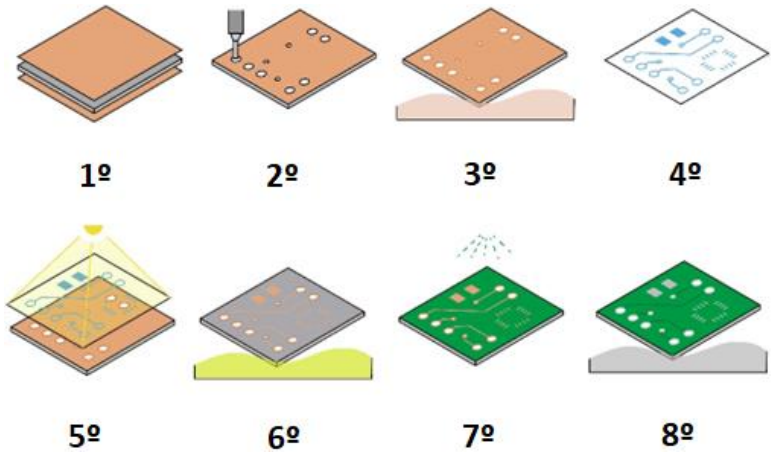


Figura 83 – Proceso de fabricación de la PCB [61]

Finalmente, tras todo el proceso de fabricación, el diseño de este proyecto ha dado como resultado la placa impresa que podemos ver en las siguientes figuras.

La capa superior, o *top* en inglés, de la PCB la podemos ver en la figura 84.

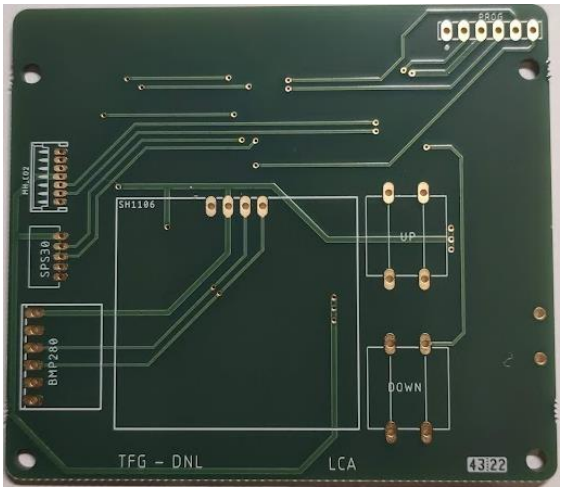


Figura 84 – Capa superior - PCB

La capa inferior, o *bottom* en inglés, de la PCB la podemos ver en la figura 85.

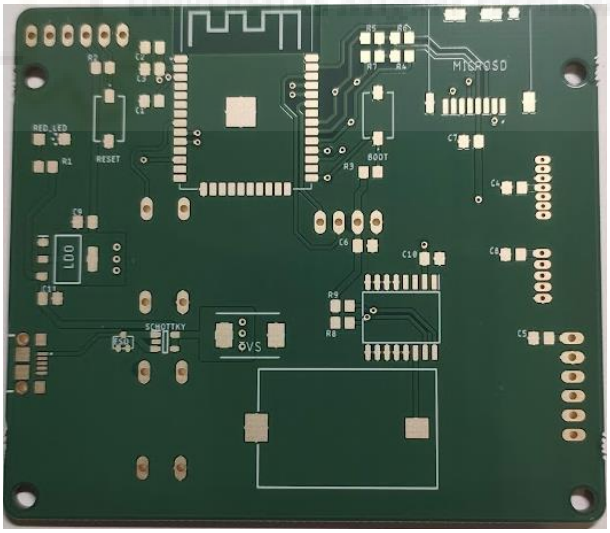


Figura 85 – Capa inferior – PCB

4.5.2. SOLDADURA

Entre todos los métodos disponibles para realizar la soldadura de los componentes SMD se ha escogido utilizar el método de soldadura por reflujo en horno. Mientras que el método de soldadura para los componentes THT ha sido mediante soldadura manual.

Se ha escogido el método de soldadura por horno de reflujo porque es un método muy utilizado, para el cual no es necesario mucho material industrial, se obtiene muy buen resultado y es muy sencillo de realizar.

HORNO DE REFLUJO

El método de soldadura por horno de reflujo consiste en:

Primero, colocar pasta de soldadura en los *pads*, contactos metálicos donde irán conectados los componentes de la PCB, esta pasta de soldadura (Fig. 86) deberá tener unas propiedades características que le permitan fundirse a altas temperaturas.

Figura 86 – Pasta de soldadura [62]

Segundo, colocar los componentes sobre la placa impresa. Se puede revisar el BOM, de las siglas en inglés *bill of materials* (lista de materiales), en el anexo 5 y ver que componente corresponde a cada huella.

Tercero, introducir la PCB en un horno de reflujo (Fig. 87) y esperar a que se complete el perfil de temperatura. Según la pasta de soldadura y los componentes utilizados se debe seguir un perfil de temperatura que permita una correcta soldadura sin dañar ninguno de los componentes.

Figura 87 – Horno de reflujo [63]

SOLDADURA MANUAL

Por otro lado, el método de soldadura manual consiste en utilizar un soldador manual o cautín, el cual calienta la punta de este para fundir estaño y con esto soldar los componentes a la placa impresa.

Un ejemplo de soldador y del estaño para soldar, es el de la figura 88.

Figura 88 – Soldador y estaño [64]

Tras la soldadura y el montaje obtenemos el siguiente resultado (Fig. 89):

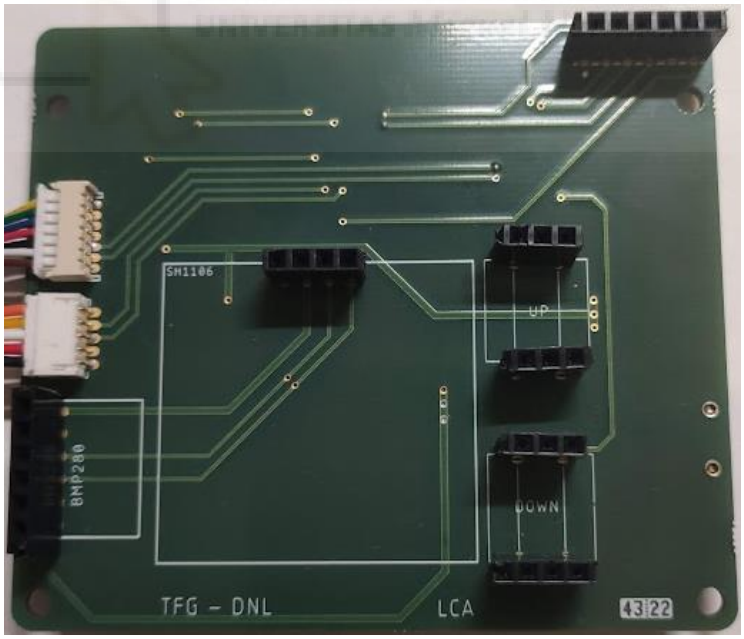


Figura 89 – Capa superior sin componentes – Soldadura

Como se puede ver se han colocado unos conectores que permiten poner y quitar fácilmente los componentes sin necesidad de soldar y desoldar los mismos.

En la figura 90 se puede observar la placa con los componentes THT ya instalados.

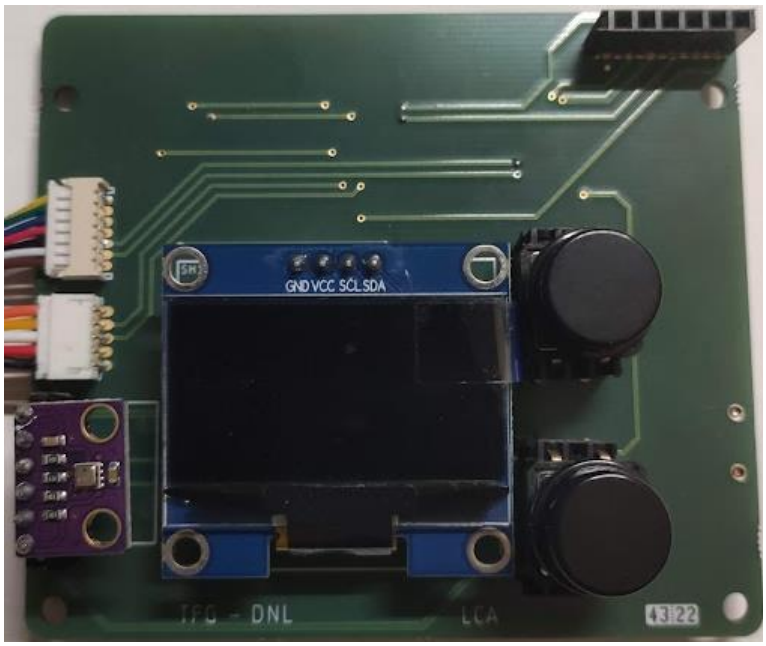


Figura 90 – Capa superior con componentes – Soldadura

Por último, la parte inferior de la placa con los componentes soldados se pueden observar en la figura 91.

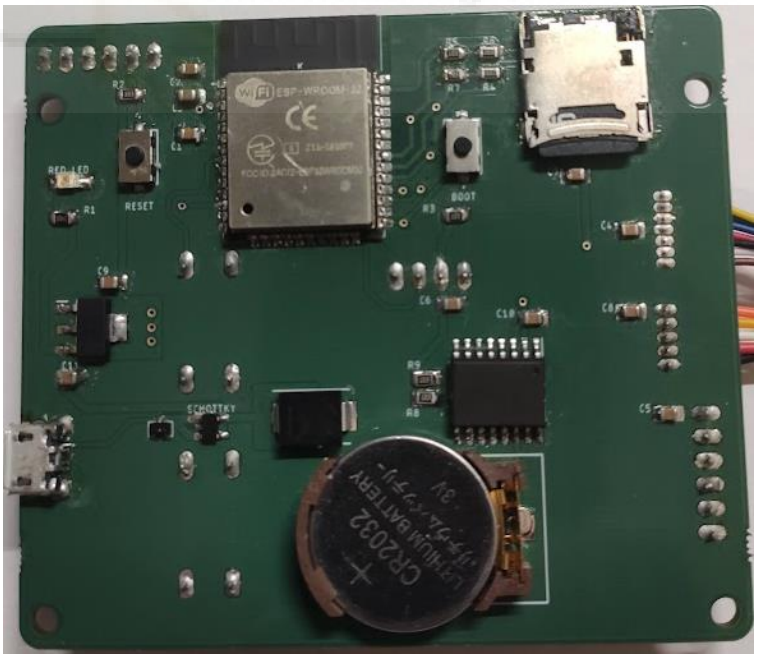


Figura 91 – Capa inferior – Soldadura

Se observa que la placa de circuito impreso, fabricada a medida para este prototipo, permite darle un aspecto y acabado más profesional, en un espacio mucho más reducido.

4.6. DISEÑO CARCASA

Para terminar el dispositivo, se va a diseñar una carcasa que permita compactar y proteger más el prototipo. Además, esta carcasa tiene el propósito o intención de dar una estética mejorada y más profesional a un producto de cara a su posible comercialización.

Como se comentaba en el apartado de herramientas informáticas, se va a utilizar el programa Fusion 360. Una de las principales ventajas de este programa es que permite importar la placa diseñada de Eagle.

Como se puede ver, en la figura 92, al importar el diseño de la placa de circuito de Eagle (Fig. 92) se obtiene como sería la placa en 3D y facilita el proceso de diseño de la carcasa para la misma.

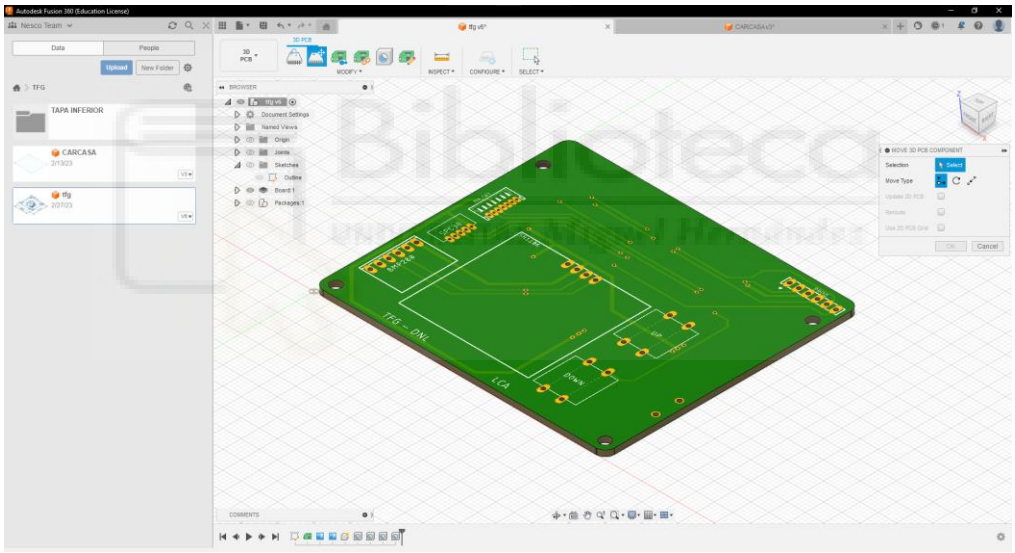


Figura 92 – PCB – Diseño 3D

Teniendo en cuenta el tamaño que ocupan los componentes se puede diseñar la placa en función de gustos. Aunque se deberá tener en cuenta que se debe dejar suficiente ventilación para que los sensores midan correctamente.

En este caso, se ha decidido dividir la carcasa en dos partes: la superior (Fig. 93) e inferior (Fig. 94).

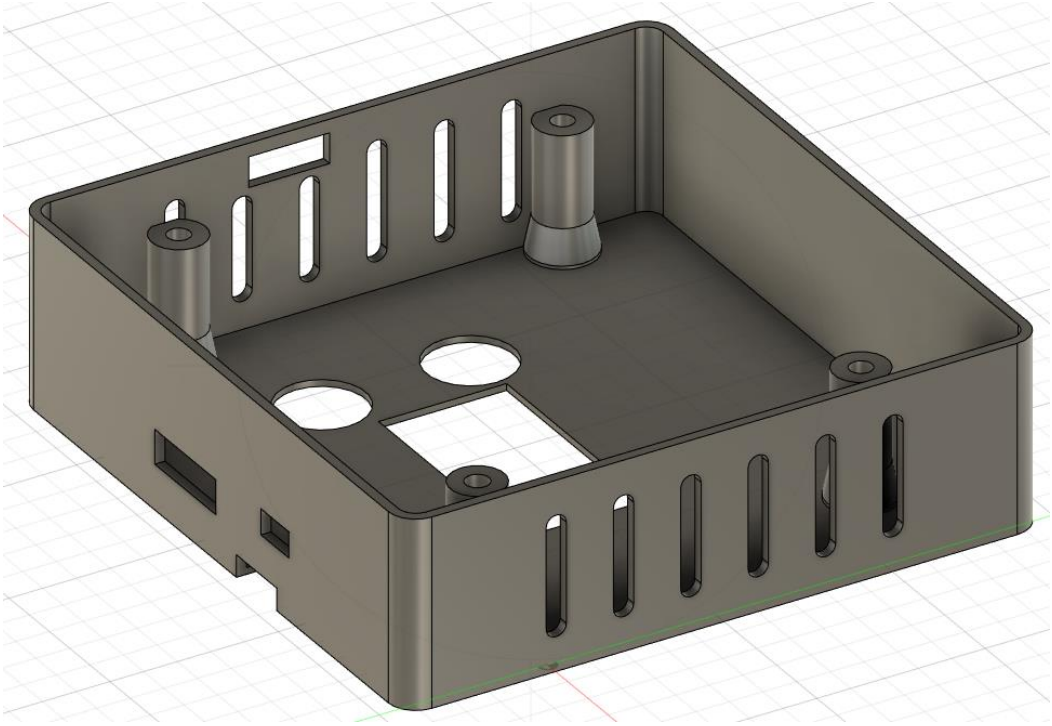


Figura 93 – Carcasa parte superior – Diseño 3D

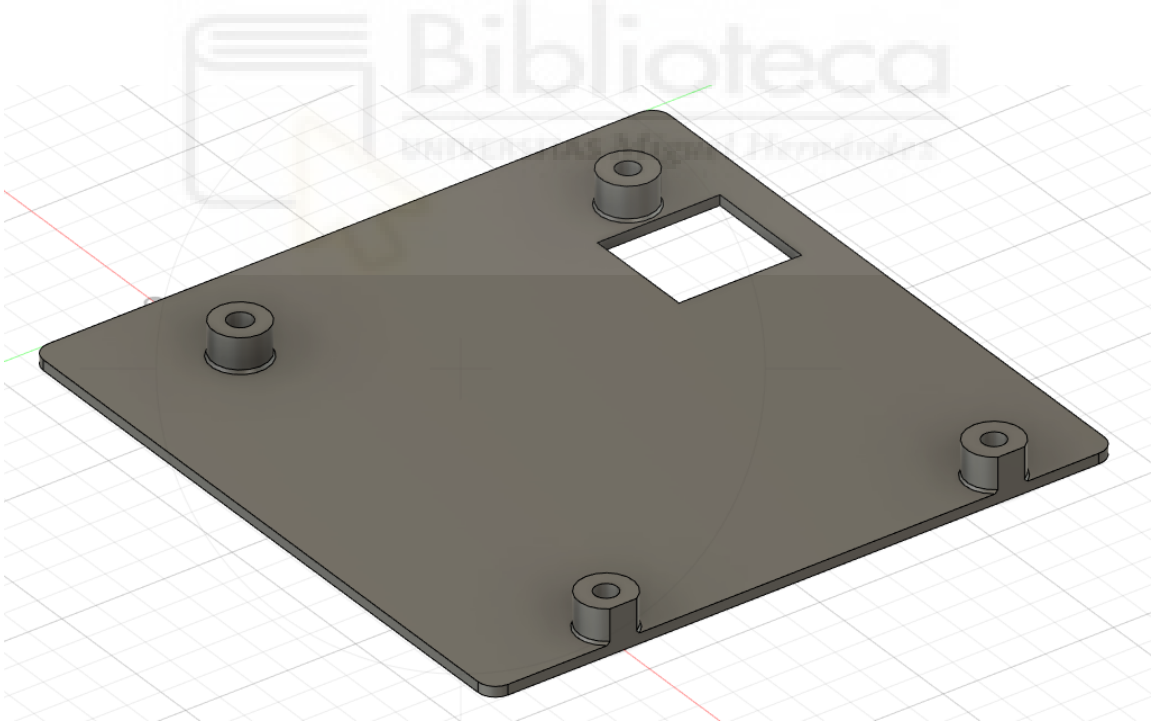


Figura 94 – Carcasa parte inferior – Diseño 3D

Como se observa en las figuras 93 y 94, se ha dejado ranuras tanto para los componentes como para ventilar correctamente. Además, se ha realizado el diseño para que los tornillos solo sean visibles desde la parte trasera del dispositivo.

4.7. FABRICACIÓN CARCASA

Una vez terminado el diseño de la carcasa, se va a fabricar la misma.

Existen muchos métodos y materiales para fabricar diseños en 3D. Aunque para este caso, se ha decidido fabricar la carcasa mediante una impresora 3D de filamento (Fig. 95) debido a las grandes ventajas que presenta: bajo coste, tiempos reducidos de fabricación y posibilidad de creación de diseños complejos.

Figura 95 – Impresora 3D [65]

Además, existen muchos tipos de materiales que permiten imprimir en 3D en función de las necesidades: rigidez, elasticidad, resistencia a altas temperaturas, etc.

En este caso, se va a utilizar el PLA, el material más utilizado y fácil de imprimir. Se ha escogido este material debido a que las necesidades no son muy exigentes. El dispositivo está diseñado para estar en espacios cerrados y, por lo tanto, no va a estar expuesto a altas temperaturas. Además, va a estar preferiblemente fijado en la pared o apoyado en una superficie, por lo tanto, no va a necesitar soportar ninguna fuerza mecánica.

Utilizando una impresora 3D, similar a la de la figura 95, y utilizando un filamento cualquiera de material PLA se puede imprimir cualquier diseño 3D como el visto anteriormente (diseño carcasa). Aunque, para que la impresora 3D pueda imprimir un diseño 3D, será necesario utilizar un laminador que convierta el archivo en 3D a un archivo que comprenda la impresora.

Una vez hecho esto se podrá imprimir la carcasa diseñada y obtendremos el siguiente resultado:

Figura 96 – Carcasa fabricada - Superior

Figura 97 – Carcasa fabricada – Inferior

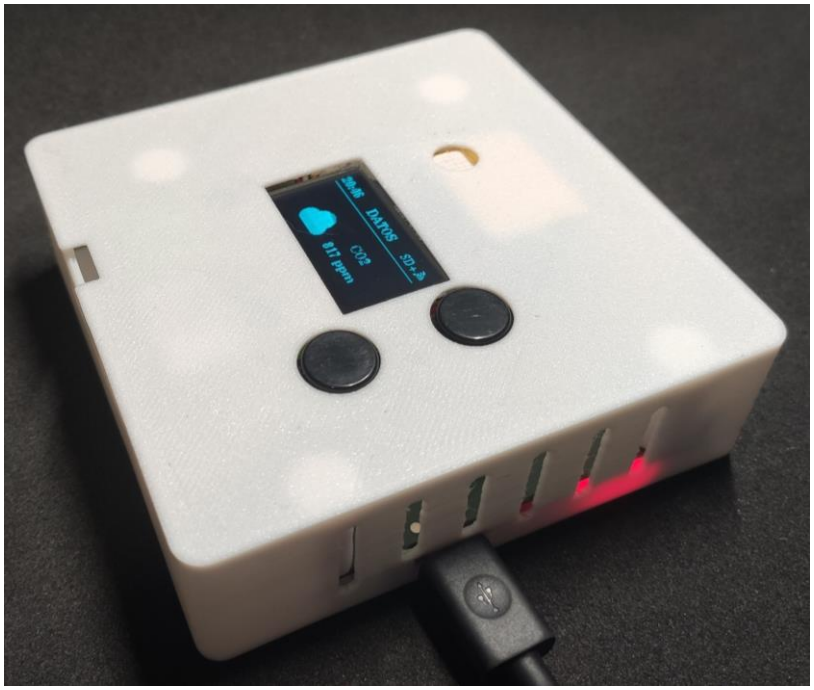


Figura 98 – Carcasa fabricada – Destacan los orificios practicados para la ventilación y para la toma de muestra de los sensores.

Se observa, en las tres figuras anteriores (Fig. 96-98), como el resultado es satisfactorio y encaja perfectamente en la placa diseñada. Además, se pueden observar los orificios con el propósito de ventilar el interior del dispositivo para la correcta medición de los sensores.

4.8. CONSUMO

El aparado de consumo está dividido en 4 apartados. En el primero se analizan los consumos de los distintos periféricos del dispositivo, en el segundo se estudia el consumo del dispositivo completo, en el tercero se optimiza el código para obtener un menor consumo y en el último, se compara la autonomía del dispositivo sin optimizar con el optimizado.

La medida de consumos se ha realizado mediante el dispositivo NRF-PPK2 o Power Profile Kit II (Fig. 99), del fabricante Nordic Semiconductor. Este dispositivo es una herramienta fácil de utilizar que permite medir la corriente y analizar consumos de corriente.

El NRF-PPK2 permite almacenar la información leída por un amperímetro interno, representar los valores de la corriente medida y estimar el consumo de corriente en un periodo de tiempo.

Figura 99 – NRF-PPK2 [66]

4.8.1. CONSUMO DE LOS PERIFÉRICOS

SENSOR BME280

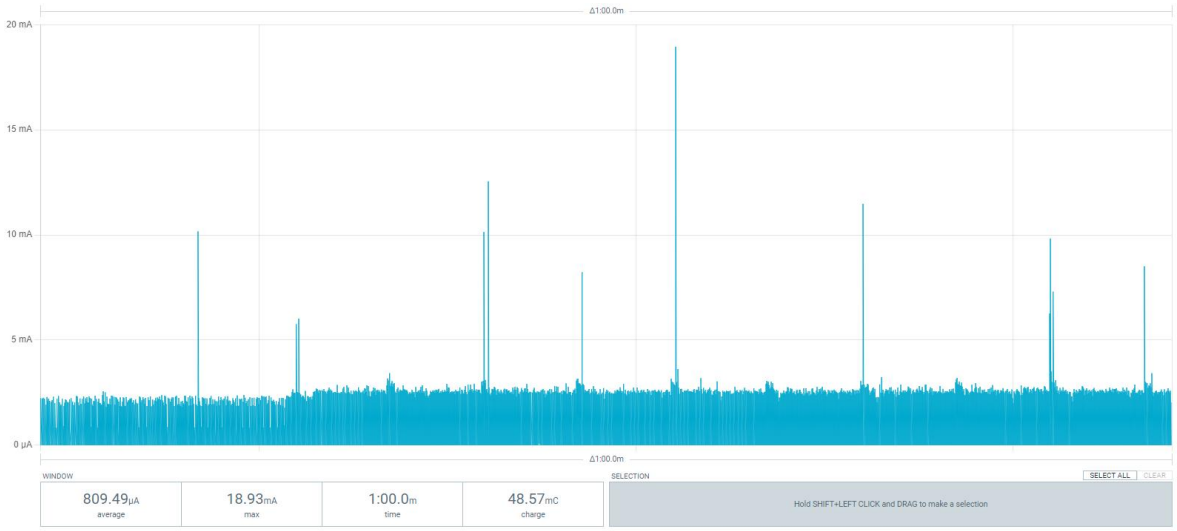


Figura 100 – Consumo BME280

En la figura 100 se observa como el consumo medio de corriente de este sensor es bajo (809,5 µA), no llega a 1 mA de media. En la misma figura se observan picos de corriente de alrededor de 10 mA, posiblemente debidos a cuando el sensor realiza una medida.

SENSOR MH-Z19B

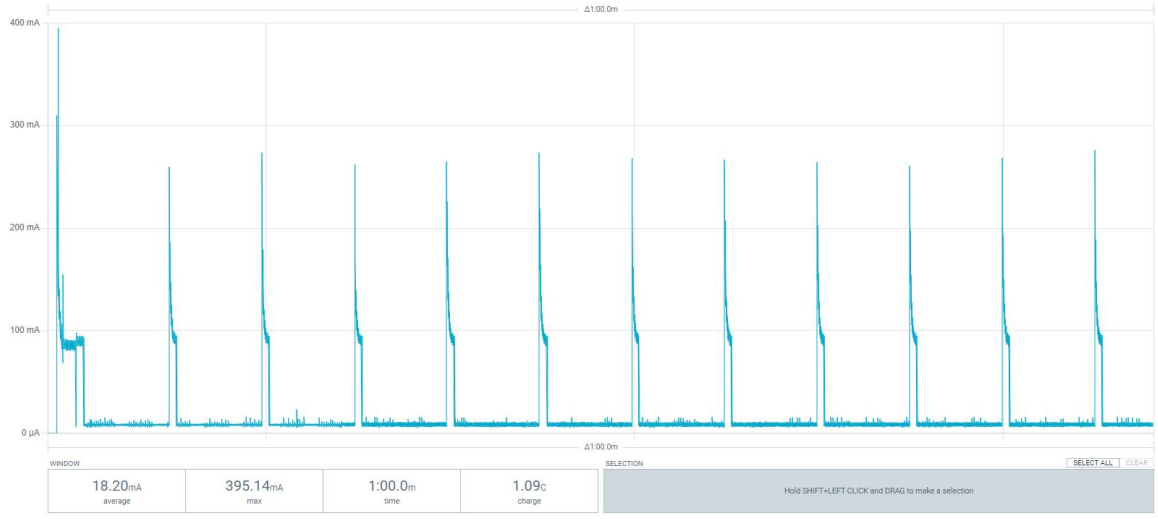


Figura 101 – Consumo MH-Z19B

El consumo de corriente por este sensor (Fig. 101) en el espacio de tiempo donde no está midiendo es muy bajo. Aunque durante la medida el consumo aumenta considerablemente, alcanzando los 395 mA. Sin embargo, la corriente media es de 18,2 mA, no muy elevada.

SENSOR SPS30

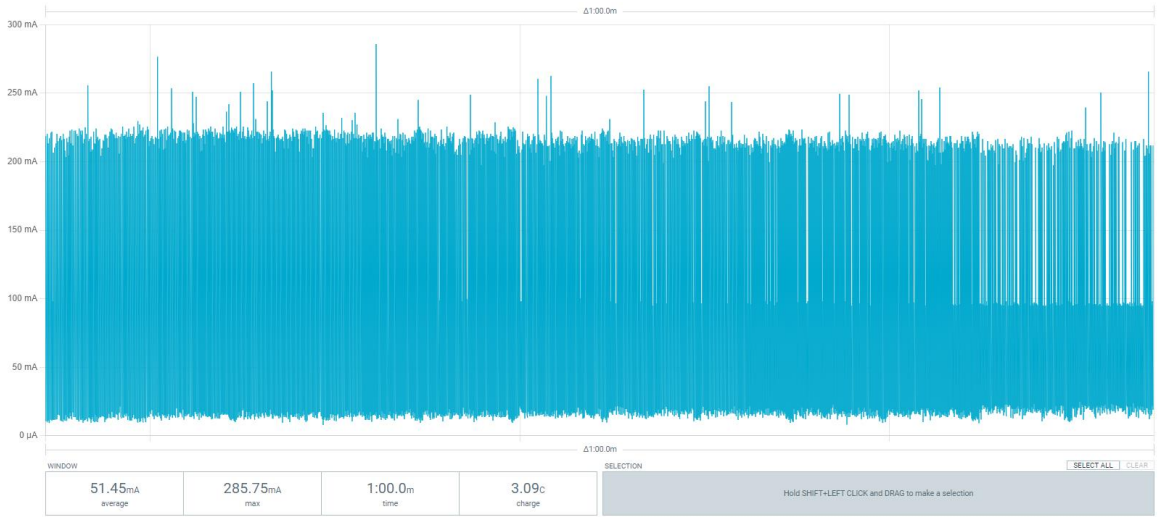


Figura 102 – Consumo SPS30

Como se puede observar el sensor SPS30 (Fig. 102), tiene un consumo de corriente bastante comedido se sitúa alrededor de los 50 mA. Aunque tenemos consumos pico bastante altos por encima de los 280 mA.

PANTALLA

Figura 103 – Consumo Pantalla

La pantalla tiene un consumo medio de corriente de 10,4 mA (Fig. 103), un consumo no muy elevado.

4.8.2. CONSUMO DISPOSITIVO

El apartado donde se va a medir el consumo del dispositivo completo se va a dividir en función de los distintos modos de funcionamiento, para ver si existe alguna diferencia de consumos ente ellos.

En primer lugar, el caso donde el dispositivo está guardando los datos tanto a través de internet como a través de la memoria microSD.

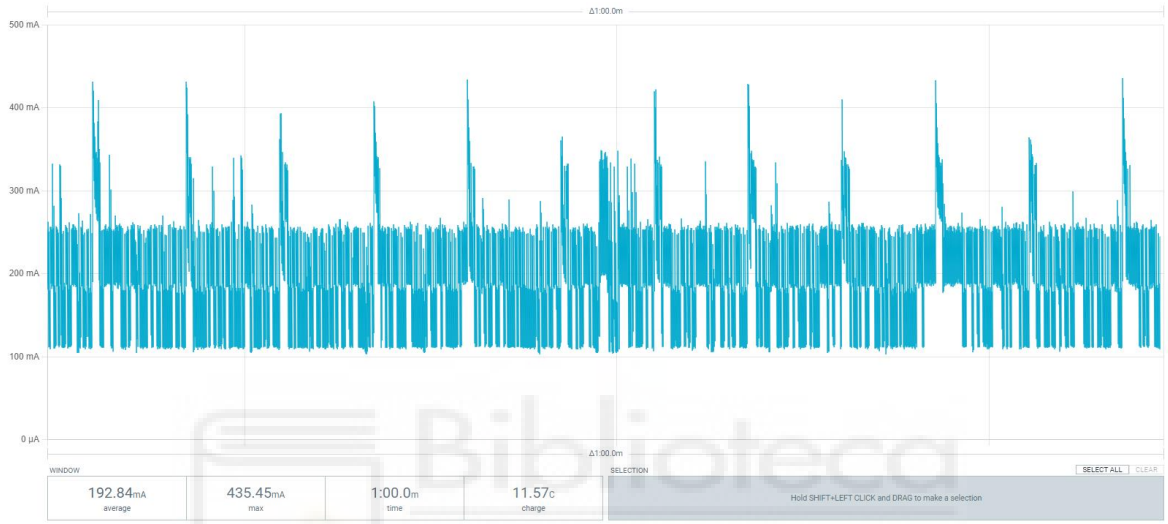


Figura 104 – Consumo prototipo – SD+WIFI

Se ha obtenido que el consumo medio de corriente del sistema completo (Fig. 104) es de 192,84 mA con unos picos de 435,45 mA.

En segundo lugar, el caso donde el dispositivo está guardando los datos solamente en la tarjeta microSD.

Figura 105 – Consumo prototipo – SD

Se ha obtenido un consumo medio de corriente de 204,42 mA con picos de consumo de hasta 465,09 mA (Fig. 105). Se observa que el consumo supera al modo de guardado anterior, aunque la diferencia es de poco más de 10 mA un aumento no muy relevante.

En tercer lugar, el caso donde el dispositivo está guardando los datos solamente a través de internet se obtiene:

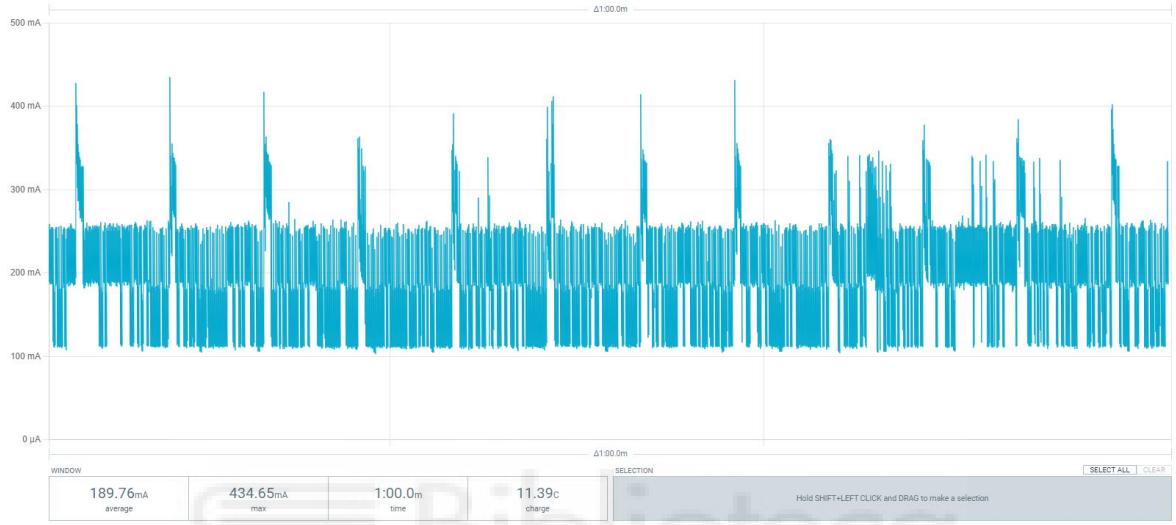


Figura 106 – Consumos prototipo - WIFI

Se ha obtenido (Fig. 106) un consumo medio de corriente de 189,76 mA con picos de consumo de 434,65 mA.

Tras hacer las tres medidas de consumo en el dispositivo para sus tres modos principales de funcionamiento, se puede concluir que no se obtiene una gran diferencia en el consumo para los distintos modos.

Se realizaron estas medidas de consumo de tal manera que cuando se encontraba en modo de guardado en microSD se realizara el guardado cada 5 segundos. No se ha visto ninguna diferencia cuando se modificaba los tiempos de guardado.

4.8.3. CONSUMO DEL DISPOSITIVO OPTIMIZADO

Observando que el dispositivo tiene una media de consumo de corriente de 200 mA, se ha procedido a optimizar el código del microcontrolador con el fin de reducir dicho consumo. Además, se ha quitado tanto la pantalla como los botones, de esta manera se tendrá dos versiones del prototipo:

- La primera dispondrá de pantalla, botones y estará orientada a situaciones donde no se tengan limitaciones en la alimentación.
- La segunda estará preparada para situaciones con limitaciones en la alimentación (medidas en lugares remotos) y en las que ésta se produzca mediante el uso de baterías o paneles solares.

Para esta segunda versión, la intención es que se realice una optimización de consumo que haga que el microcontrolador se quede en un modo de reposo "dormido" y se "despierte" una vez cada 10 minutos para hacer la medida de los sensores, subir los datos a internet y guardar los datos en la memoria. Esto implicará una pérdida de información considerable, ya que, en vez de realizar un promedio durante 10 minutos, se cogerá la medida justo cuando el dispositivo "despierte". Esto último habrá que tenerlo en cuenta y puede ser importante en función de las necesidades del usuario.

Para llevar a cabo esta optimización de consumo se ha utilizado el modo de reposo *deepsleep*. En modo *deepsleep* tanto el procesador (CPU) como el wifi y el bluetooth se encuentran apagados, esto permite reducir considerablemente el consumo del ESP32. El microcontrolador se encontrará modo dormido durante el periodo de tiempo que le indiquemos, el valor está en microsegundos, y cuando se despierte realizará todas las tareas que le indiquemos hasta que llegue a la función de empezar a dormir (Fig. 107).

```
esp_sleep_enable_timer_wakeup(tiempodormido);

(…) //Lectura sensores, subida datos a internet y guardar en microSD.

esp_deep_sleep_start(); //A dormir
```

Figura 107 - Deepsleep – Instrucción para habilitar el modo de deepsleep del dispositivo

Como se ha comentado anteriormente, el valor del *tiempodormido* escogido es de 10 minutos. Esto permitirá que en situaciones en las que no sea necesaria una resolución temporal alta se tengan consumos reducidos.

Si se quiere profundizar más en el código, éste está disponible en el anexo 2.

Tras optimizar el código del microcontrolador se procederá a medir el nuevo consumo del dispositivo.

El consumo en el periodo que está dormido (Fig. 108):

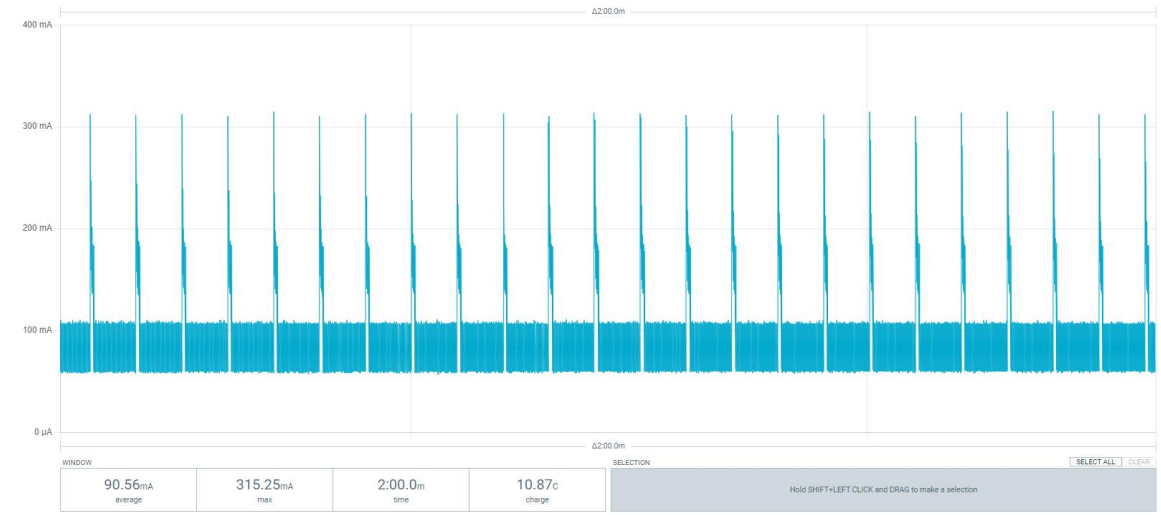


Figura 108 – Consumo prototipo optimizado - Dormido

El consumo en el periodo que está despierto (Fig. 109):

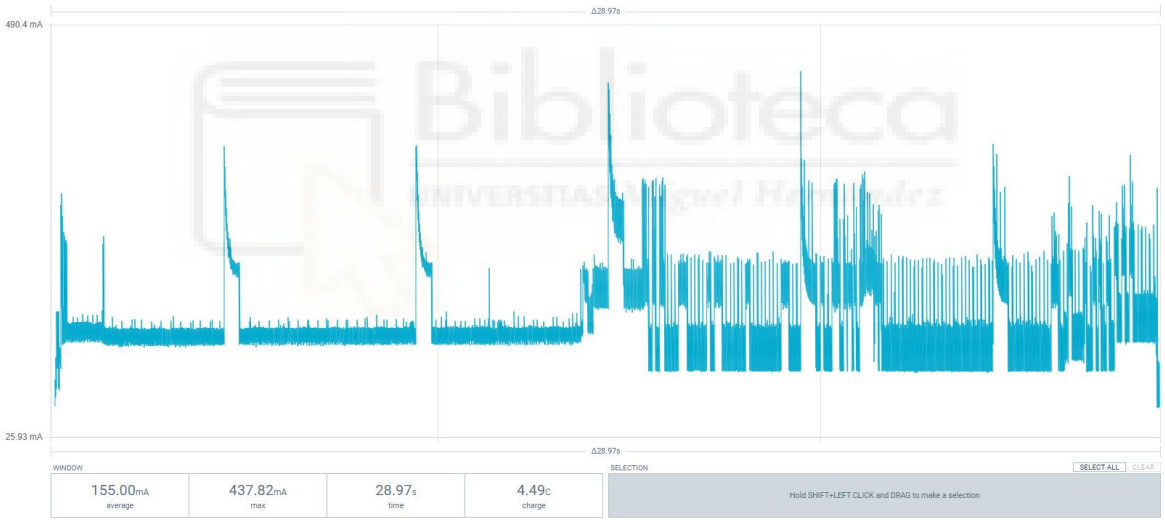


Figura 109 – Consumo prototipo optimizado - Despierto

Como se ha podido observar en la figura 109, el dispositivo se encuentra despierto durante 28,97 segundos. Este es el tiempo que el dispositivo tarda en realizar las instrucciones programadas: conectarse a internet, lectura de sensores, guardado en memoria, subida de datos, etc…

Debido que para medir el consumo medio del dispositivo necesitaríamos medir durante 10 minutos completos y el dispositivo de medida de consumos, NRF-PPK2, solo es capaz de medir 2 minutos se debe realizar un cálculo sencillo.

Para calcular el consumo medio del dispositivo utilizaremos la siguiente ecuación (Ec. 4):

$$I_{media} = \frac{T_{despierto} \cdot I_{despierto}}{T_{total}} + \frac{T_{dormido} \cdot I_{dormido}}{T_{total}}$$

Ecuación 4 – Cálculo corriente media

La ecuación 4 consiste en hacer una media del consumo de corriente. Para ello se deberá calcular la suma de consumo de cada periodo, despierto y dormido, teniendo en cuenta la proporción de tiempo que se encuentra en cada periodo.

Puesto que el tiempo en el que se encuentra despierto es de 28,97 segundos y el tiempo en el que se encuentra dormido es de 10 minutos. Además, el consumo en el periodo en el que se encuentra despierto es de 155mA y en el periodo en el que se encuentra dormido el consumo es de 90,56mA.

Podemos calcular el consumo medio de corriente:

$$I_{media} = \frac{28{,}97 \cdot 155}{(10 \cdot 60 + 28{,}97)} + \frac{10 \cdot 60 \cdot 90{,}56}{(10 \cdot 60 + 28{,}97)} = 93{,}53\ mA$$

4.8.4. COMPARATIVA DE AUTONOMÍA

Tras la medida de los consumos para las dos versiones del dispositivo, la optimizada para bajo consumo y la que no está optimizada, se puede sacar la siguiente conclusión: al tener la mitad de consumo de corriente en la versión optimizada se va a disponer del doble de autonomía, aunque se pierda algunas funcionalidades como la pantalla.

En caso de disponer de unas baterías que tengan una tensión de 5 V y una capacidad de 10.000 mAh, se va a calcular la autonomía de cada una de las versiones.

El cálculo necesario para obtener las horas de autonomía es el siguiente:

$$Horas\ de\ autonomía = \frac{Capacidad\ (mAh)}{Consumo\ de\ corriente\ (mA)}$$

Ecuación 5 – Ecuación cálculo autonomía

La ecuación 5 consiste en dividir la capacidad de la batería, unidades de mAh, entre el consumo de corriente en mA.

Para la versión sin optimizar obtenemos:

$$Horas\ de\ autonomía = \frac{10000\ mAh}{200mA} = 50h = \text{ 2 días y 2 horas}$$

Para la versión optimizada obtenemos:

$$Horas\ de\ autonomía = \frac{10000\ mAh}{200mA} = 100h = 4\ días\ y\ 4\ horas$$

Sin duda, las horas de autonomía son muy reducidas incluso en el caso en el que se optimiza el dispositivo para bajo consumo.

El sensor de partículas y el de CO_2 tienen un consumo medio bastante elevado, como hemos visto en el apartado de consumos de periféricos. Además, se ha observado (Fig. 108) que aun encontrándose en el estado de *deepsleep* del microcontrolador el sensor de CO_2 se encuentra constantemente midiendo y consumiendo corriente.

En el caso de que se quisiera optimizar aún más el consumo de corriente del dispositivo tendríamos las siguientes opciones:

- Utilizar sensores que, aparte de medir con precisión, tengan un consumo reducido.
- Desconectar la alimentación de los periféricos (memoria microSD, RTC y sensores) cuando el microcontrolador se encuentre en modo dormido. De esta forma se puede asegurar que el consumo en el periodo dormido es el mínimo necesario.

Cualquiera de estas opciones reduciría considerablemente el consumo del dispositivo.

Por otro lado, en el caso de que no se quiera perder tanta resolución en la medida, se podría reducir el tiempo que el dispositivo se encuentra dormido. Reducir, por ejemplo, el tiempo dormido de 10 minutos a 2 minutos. Aunque, se debe tener en cuenta que cuanto más se reduzca el tiempo dormido más aumentará el consumo del dispositivo.

5. PRESUPUESTO

Este apartado es muy importante puesto que estimar el coste del producto es una parte fundamental para determinar el alcance que tendrá, lo competitivo que será y los recursos económicos que son necesarios para poder fabricar cada dispositivo.

Además, dado que se trata de producir un producto económico y de bajo coste, este apartado va a ser un factor clave para saber si se ha logrado el objetivo inicial o no.

A continuación, se presenta una tabla (Tab. 14) con todos los elementos sistema, cada uno de sus precios, una pequeña descripción y el precio total del producto. En este presupuesto no se tienen en cuenta, ni la mano de obra, ni el trabajo invertido en su diseño que habría que repercutir sin duda en el presupuesto final.

Unidades	Nombre del producto	Descripción del producto	Coste/unidad (€/u)	Coste (€)	Porcentaje
1	PCB	Fabricación de la PCB	15,00	15,00	14,5%
1	ESP32	ESP32 WROOM 32	2,00	2,00	1,9%
2	Botones THT	Botones - 4 pines - THT	0,15	0,30	0,3%
1	Pantalla	Pantalla – SH1106 – 1,3”	2,00	2,00	1,9%
1	RTC	RTC - DS3231N	2,00	2,00	1,9%
1	Portapila	Portapila - CR2032 - RSPRO	0,96	0,96	0,9%
1	Pila	Pila CR2032	1,50	1,50	1,4%
1	microSD	Soporte para microSD - DM3D-SF	1,80	1,80	1,7%
1	R240 Ω	Resistencia 240 Ω - 0805 - BOURNS	0,41	0,41	0,4%
8	R10kΩ	Resistencia 10k Ω - 0805 - BOURNS	0,03	0,24	0,2%
1	Diodo	Diodo Schottky - RB496EATR	0,66	0,66	0,6%
1	ESD	ESD - VCAN36C2-03G-E3-08	0,40	0,40	0,4%
1	TVS	TVS - SMCJ7.0A-Q	0,73	0,73	0,7%
9	C0.1u	Condensador 0805 0.1u 50V - KEMET	0,13	1,17	1,1%
3	C10u	Condensador 0805 10u 50V - KEMET	0,12	0,36	0,3%
1	LDO	LDO 3.3V - NCP1117ST33T3G	0,72	0,72	0,7%
1	LED	Led rojo - 1206 - BR1101W-TR	0,45	0,45	0,4%
2	Botones SMD	Botones - 2 pines - SMD	0,10	0,20	0,2%
2	Conectores	Conectores para sensores	0,50	1,00	1,0%
1	Sensor SPS30	Sensor de partículas de Sensirion	51,00	51,00	49,1%
1	Sensor MH-Z19B	Sensor de CO2 de Winsen	12,95	12,95	12,5%
1	Sensor BME280	Sensor de meteorológico de Bosch	6,50	6,50	6,3%
1	Carcasa	Filamento PLA para carcasa	1,42	1,42	1,4%
			TOTAL	103,77	100,0%

Tabla 14 – Presupuesto de los componentes del producto

En la tabla 14, se puede observar como el precio del dispositivo se encuentra alrededor de los 100 €, inferior al objetivo deseado. Aunque, este precio se considera aún un poco elevado, habría que optimizar y cambiar algunos elementos que elevan el precio considerablemente.

Como se puede observar, el precio del sensor de partículas SPS30 representa, prácticamente, un 50% del coste total del dispositivo. Por lo tanto, un buen método para reducir los costes del dispositivo sería cambiar el sensor por uno de características similares, pero con un precio más reducido.

El coste total del dispositivo también disminuirá en el momento en el que se pase a una fabricación de grandes cantidades, alcanzando una economía de escala que permita aumentar la producción reduciendo los costes.

6. CONCLUSIONES Y LÍNEAS FUTURAS

En este apartado se comprobará si el producto realizado se ajusta a los objetivos inicialmente planteados y si cumple las expectativas iniciales. Además, también se verá si, el mismo, no supera el coste deseado inicialmente.

En este proyecto se ha conseguido fabricar un dispositivo, AirKnowledge, capaz de medir parámetros de vital importancia para el medio ambiente/salud con un tamaño reducido y con un precio asequible. El dispositivo además es capaz de medir con precisión y permite obtener medidas comparables a dispositivos de referencia de mucho más tamaño y coste.

Este producto es un dispositivo conectado a internet que permite visualizar los datos desde cualquier móvil u ordenador en cualquiera de las dos plataformas IoT utilizadas (Fig. 57-58). También se ha conseguido hacer un sistema de guardado local, gracias a la microSD y al RTC utilizados. Dispone de una pantalla y unas teclas que permiten visualizar información y controlar el funcionamiento del dispositivo. Además, se ha fabricado una PCB y una carcasa hecha a medida para este producto que permite darle una apariencia más profesional y compacta al dispositivo (Fig. 98).

En cuanto al coste, apartado de presupuesto, se ha conseguido que el dispositivo tenga un precio ajustado, alrededor de los 100 euros y que, comparándolo con otros dispositivos del mercado, se observa que ofrece iguales o superiores características, con una precisión de medida muy buena, a un precio reducido.

Como conclusión se podría decir que el dispositivo diseñado y fabricado ha conseguido cumplir con todas las expectativas y objetivos planteados al inicio del proyecto. El dispositivo es muy competitivo con respecto a otros dispositivos actuales en el mercado gracias a sus funcionalidades y su ajustado precio.

A pesar de lo dicho anteriormente el margen de mejora es todavía considerable y se proponen a continuación algunas ideas que podrían llevarse a cabo con el fin de mejorar más el dispositivo:

Habría que estudiar cómo se podría escalar el dispositivo para venderlo en masa y su viabilidad. Las tareas que habría que hacer serían, por ejemplo, optimizar las relaciones con proveedores para obtener los componentes a un precio más reducido, automatizar el proceso

de fabricación, automatizar test de verificación del correcto funcionamiento, servicio de envío al cliente, soporte técnico, etc.

En caso de estar interesados en un dispositivo portátil, el cual podamos transportar a cualquier sitio para hacer medidas en varias localizaciones, sería interesante incluir una batería con su sistema de carga y optimizar el dispositivo para el menor consumo posible. En definitiva, habría que seguir profundizando en la búsqueda de una solución que permitiera reducir el consumo del dispositivo y, por lo tanto, aumentar su autonomía lo máximo posible sin perder funcionalidades.

Otra línea futura sería incorporar paneles solares al sistema, esto permitiría disponer de una alimentación sostenible, además de brindar al dispositivo de autosuficiencia. Esto sería muy útil para obtener medidas de lugares remotos con difícil acceso a tomas de corriente.

Con el fin de mejorar la apariencia y funcionalidad del dispositivo, se podría cambiar la pantalla que se dispone actualmente (Fig. 33) y sustituirla por una pantalla táctil, con colores, más pulgadas y mayor calidad. A pesar de ser más costosa y con mayor consumo, esta pantalla permitiría visualizar más valores al mismo tiempo, permitiría añadir más funcionalidades y mejoraría aún más la interacción con el usuario.

Además, se podría incluir más sensores al dispositivo y, de esta manera, aumentar el número de medidas de contaminación. De este modo, se podría conocer más información sobre la calidad del aire.

Sería interesante incluir alarmas que avisen al usuario en el momento en el que las medidas superen los umbrales recomendados [2].

Otra línea futura podría ser la creación de redes de dispositivos interconectados entre sí donde trabajasen de forma colaborativa para recoger los datos y enviarlos a través del dispositivo con las comunicaciones óptimas.

Se podría integrar AirKnowledge en dispositivos inteligentes (como Alexa, Google Home, etc), que permitan interactuar el dispositivo en remoto.

Se debería incluir algún método, aplicación móvil o página web, que permitiese conectar el dispositivo, AirKnowledge, a cualquier red wifi de una manera sencilla para el usuario.

Por último, sería muy recomendable que existiese una aplicación propia que permitiese acceder a todos los AirKnowledge en propiedad y permitiera configurarlos individual y remotamente.

7. ANEXOS

7.1. ANEXO 1 – CÓDIGO

Como se ha comentado en el apartado de herramientas informáticas, se va a utilizar el entorno de desarrollo de Arduino en la aplicación de Visual Studio Code para programar en C/C++ el microcontrolador ESP32.

Antes de hablar de la programación de un microcontrolador se debe recordar la estructura básica de todo código orientado a microcontroladores explicada anteriormente, programación del microcontrolador.

Nota: En todas las funciones donde tenemos una lectura de un sensor he incluido una condición de una definición, #ifdef (...) #endif, que permite desde un solo punto del código modificar si queremos que se muestren, o no, por el puerto serial la información de los sensores.

7.1.1. PROGRAMACIÓN PARA LA LECTURA DE SENSORES

BME280

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 18-20).

Creamos el constructor, para inicializar el objeto (Anexo 1, línea 122-123).

Estructura donde guardaremos los datos (Anexo 1, línea 147-155).

Dentro del main

Iniciamos la comunicación con el sensor (Anexo 1, línea 258-259).

Función a la que llamaremos desde el loop para leer el sensor

La función read_bme280() (Anexo 1, línea 1779-1808) será la que incluye todos los comandos para leer el sensor, guardarlos datos en variables globales para mostrarlos por la pantalla oled y los almacenará en la estructura de datos para posteriormente guardarlos en la microSD.

SPS30

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 15-16).

Declaramos el tipo de comunicación, UART, y los pines para ella (Anexo 1, línea 75-83).

Creamos el constructor, para inicializar el objeto (Anexo 1, línea 113-114).

Estructura donde guardaremos los datos (Anexo 1, línea 129-145).

Dentro del main

Iniciamos la comunicación con el sensor y comprobamos que se consiga hacer conexión (Anexo 1, línea 307-346).

Función a la que llamaremos desde el loop para leer el sensor

La función read_sps30() (Anexo 1, línea 1645-1737) será la que incluye todos los comandos para leer el sensor, guardarlos datos en variables globales para mostrarlos por la pantalla oled y los almacenará en la estructura de datos para posteriormente guardarlos en la microSD.

Funciones complementarias

- La función GetDeviceInfo(){…} (Anexo 1, línea 1597-1643) permite obtener bastante información sobre el sensor que estamos utilizando como por ejemplo: su número de serie, el nombre del producto y distintos datos sobre la versión del producto.
- La función Errorloop(char *mess, uint8_t r){…} (Anexo 1, línea 1739-1745) es una función auxiliar que pone el programa en pausa debido a un error con el sensor sps30.
- La función ErrtoMess(char *mess, uint8_t r){…} (Anexo 1, línea 1747-1753) es una función auxiliar que muestra, por el puerto serial, información sobre el error que se ha producido con el sensor sps30.

MH-Z19B

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 6-8).

Declaramos el tipo de comunicación, UART, y los pines para ella (Anexo 1, línea 57).

Creamos el constructor, para inicializar el objeto (Anexo 1, línea 56).

Estructura donde guardaremos los datos (Anexo 1, línea 157-162).

Dentro del main

Iniciamos la comunicación con el sensor y activamos la autocalibración (Anexo 1, línea 239-243).

Función a la que llamaremos desde el loop para leer el sensor

La función read_co2() (Anexo 1, línea 1755-1777) será la que incluye todos los comandos para leer el sensor, guardarlos datos en variables globales, para mostrarlos posteriormente en pantalla oled, y los almacenará en la estructura de datos para posteriormente guardarlos en la microSD.

7.1.2. PROGRAMACIÓN PARA LA MICROSD Y EL RTC

microSD

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 22-25).

Como el tipo de comunicación es SPI, deberemos declarar el pin de selección, CS (Anexo 1, línea 87-88).

Creamos el constructor, para inicializar el objeto de la SD (Anexo 1, línea 119-120).

Declaramos los modos de guardado, los cuales según su valor el guardado se hará cada 5 segundos, 1 minuto o 10 minutos (Anexo 1, línea 187).

Dentro del main

Iniciamos la comunicación con la microSD y comprobamos que este conectada (Anexo 1, línea 261-268).

Líneas de código dentro del loop para guardar los datos

En las líneas de código (Anexo 1, línea 488-513) tenemos las condiciones que definen cada cuanto se va a realizar un guardado. Dependerá del modo de guardado elegido por el usuario que se guarden los datos cada 5 segundos, 1 minuto o 10 minutos.

La función save_sd_todos() (Anexo 1, línea 1323-1595) tendrá todo el código necesario para generar un archivo .csv donde se almacenen los datos en filas y columnas ordenadamente para poder analizarlo posteriormente. En las columnas tendremos cada uno

de los valores medidos por los sensores y las filas separan cada medida temporalmente, se guarda la fecha y hora en la que se realiza la medida.

RTC

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 27-30).

Creamos el constructor, para inicializar el objeto del RTC (Anexo 1, línea 125-126).

Dentro del main

Iniciamos la comunicación con el RTC y comprobamos que este correctamente conectado (Anexo 1, línea 245-256).

Comprobamos si se ha perdido la alimentación y en caso de estar iniciando el dispositivo por primera vez se le da un valor de fecha y hora predeterminado (Anexo 1, línea 301-305).

Líneas de código dentro del loop para guardar los datos

En la línea de código (Anexo 1, línea 353) actualizamos la hora en el RTC cada vez que se recorre el bucle.

Mientras que, en las líneas de código (Anexo 1, línea 363-400) son las condiciones que comprueban si estamos conectados a internet y, en cuyo caso, actualizan la hora con un protocolo NTP (Protocolo de tiempo de red, para sincronizar la hora de los dispositivos conectados a internet). En caso de no estar conectados a internet intentaremos conectarnos.

7.1.3. PROGRAMACIÓN PANTALLA

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 32-39).

Creamos el constructor, para inicializar el objeto de la pantalla (Anexo 1, línea 40-41).

La variable pantalla que irá de 0 a 3, en función de su valor mostramos una información u otra (Anexo 1, línea 174).

Dentro del main

Iniciamos la comunicación con la pantalla y habilitamos UTF8 que permite mostrar los datos más fácilmente (Anexo 1, línea 228-237).

En las líneas, (Anexo 1, línea 243-246), tenemos la función drawLogo(), (Anexo 1, línea 792-818), donde mostramos, cuando se enciende el dispositivo unos segundos, el logo creado con el símbolo del laboratorio con el que se ha realizado el TFG (Fig. 62).

Líneas de código dentro del loop para guardar los datos

En las líneas (Anexo 1, línea 422-437) tenemos el código necesario para ir mostrando la informando de todos los sensores. Dentro de estas líneas estamos llamando a la siguiente función mostrarendisplay() que se encuentra, (Anexo 1, línea 776-790), esta tendrá los comandos para comunicarse con la pantalla y poder mostrar cada elemento.

7.1.4. PROGRAMACIÓN INTERFAZ DE USUARIO CON BOTONES

BOTONES

Fuera de las funciones tendremos

Estructura con la información necesaria para el funcionamiento de los botones (Anexo 1, línea 164-172).

Variables que en función de su valor mostramos una información u otra y tenemos un tipo de guardado u otro (Anexo 1, línea 174, 177, 184 y 187).

Creamos dos estructuras una para cada uno de los botones, pondremos de entrada su correspondiente información (Anexo 1, línea 189-190).

Dentro del main

Ponemos los pines correspondientes a los botones de tipo pull-up (Anexo 1, línea 223-224).

En las líneas, (Anexo 1, línea 225-226), se activa la interrupción por cada uno de los botones. En este caso, se ha configurado que la interrupción para cuando el pin cambie de estado, por lo tanto, se activará la interrupción cuando pulsemos y cuando soltemos el botón. Esto servirá para poder diferenciar entre una pulsación larga y corta.

Cuando se pulsen los botones se llamará a las funciones longpressed0() y longpressed1(), (Anexo 1, línea 516-774), estas funciones son realmente parecidas con la única principal diferencia de que uno de los botones es para subir entre las opciones y otro es para bajar.

7.1.5. PROGRAMACIÓN WIFI Y PLATAFORMAS IoT

CONEXIÓN A INTERNET

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 10-12).

Datos sobre el wifi que vamos a conectarnos por defecto, tanto el nombre de la red como la contraseña (Anexo 1, línea 67-69).

Creamos el constructor, para inicializar el objeto del cliente wifi (Anexo 1, línea 110-111).

Dentro del main

Intentamos conectarnos al wifi por defecto (Anexo 1, línea 270-275). En caso de no conseguir conectarse tratará de reintentarlo más adelante cuando intente guardar los datos en la nube o trate de obtener la hora.

Líneas de código dentro del loop para guardar los datos

En las líneas de código 354-362 (Anexo 1) cambiaremos de wifi en caso de que el usuario lo haya decidido desde la interfaz con botones.

En las líneas de código 363-372 (Anexo 1) el sistema comprueba si estamos conectados a internet. Si no estamos conectados, intentará conectarse y en caso de estar conectado accederá a los servicios en línea que se han programado.

PROTOCOLO NTP

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, línea 13).

Datos sobre servidores a los que podemos acceder para solicitar la fecha y hora, e información necesaria para solicitar la hora de huso horario correspondiente (Anexo 1, línea 60-65).

Dentro del main

Configuramos el protocolo NTP con los datos nombrados anteriormente (Anexo 1, línea 296-298).

Líneas de código dentro del loop para guardar los datos

En las líneas de código 382-398 (Anexo 1) en caso de estar conectados a internet obtendremos la fecha y hora.

PLATAFORMAS IoT

Fuera de las funciones tendremos

Incluimos las librerías y archivos necesarios para la posterior programación (Anexo 1, líneas 3-4 y 50).

Datos necesarios para poder acceder al servicio Blynk (Anexo 1, línea 43-46, 52).

Datos necesarios para poder acceder al servicio ThingSpeak (Anexo 1, línea 71-73).

Dentro del main

Una vez comprobado que estamos conectados internet, accedemos a los servicios (Anexo 1, línea 277-293).

Líneas de código dentro del loop para guardar los datos

Cada minuto se comprobará si estamos conectados a internet y se volverá a acceder a los servicios en línea IoT, donde se enviará los datos medidos por los sensores (Anexo 1, línea 439-486). En ausencia de conexión a internet, tratará de reconectarse a la red wifi. Si consiguiera establecer conexión, enviaría los datos y, en caso contrario, saldría de la función.

7.1.6. FUNCIONAMIENTO DEL BUCLE

Entender el funcionamiento del bucle es crucial cuando hablamos de programación de microcontroladores.

En primer lugar, se ha declarado al inicio del programa cuanto se quiere esperar desde que se inicia el dispositivo hasta que se ejecuta por primera vez cada una de las funciones y, en segundo lugar, cada cuanto se quiere acceder a cada función: lectura de cada sensor, guardado de los datos, mostrar por pantalla, etc (Anexo 1, línea 89-108).

Por otro lado, utilizando la función millis(), que devuelve el tiempo en milisegundos que han pasado desde que el programa comenzó, podemos controlar el acceso a las distintas funciones del programa y cada cuanto queremos acceder a las mismas.

Un ejemplo del funcionamiento del bucle se presenta en la figura 110:

```
if(millis()>timeco2){
    (…)
    timeco2=millis()+lectco2;
  }
```

Figura 110 – Funcionamiento bucle

Se observa en el ejemplo de la figura 110 como se tiene una condición de tiempo que, en caso de superarse el tiempo deseado para realizar la acción, se accede a la condición y se procede a realizar las tareas correspondientes. Esto realmente funciona como un temporizador tradicional.

De esta manera, se evita el uso de *delays* (retrasos) y, por lo tanto, no se paraliza todo el funcionamiento del microcontrolador. Se podrá estar ejecutando otras funciones mientras alcanzamos el momento indicado para realizar la función en concreto.

7.1.7. CÓDIGO

Código completo del programa para el dispositivo AirKnowledge:

```
#include <Arduino.h>

//IOT - Thingspeak
#include "ThingSpeak.h"

//MH-Z19B
#include "MHZ19.h"
#include <SoftwareSerial.h>

//Wifi
#include <WiFiClientSecure.h>
#include <WiFi.h>
#include "time.h"

//SPS30 - Air quality sensor
#include "sps30.h"

//BME280
#include <Wire.h> // required by BME280 library
#include <BME280_t.h>

//SD
```

```
//#include "FS.h"
#include <SD.h>
//#include <SPI.h>

//RTClib - Pila
#include "RTClib.h"
//#include <Wire.h>
#include <SPI.h>

//SH1106
#include <U8g2lib.h>
#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
#endif
//#ifdef U8X8_HAVE_HW_I2C
//#include <Wire.h>
//#endif
U8G2_SH1106_128X64_NONAME_1_HW_I2C u8g2(U8G2_R0, /* reset=*/
U8X8_PIN_NONE);

//BLYNK
#define BLYNK_TEMPLATE_ID "************"
#define BLYNK_DEVICE_NAME "************"
#define BLYNK_AUTH_TOKEN "************"

//#include <WiFi.h>
//#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char auth[] = BLYNK_AUTH_TOKEN;

//MH-Z19B
#define FORCE_SPAN 0 // < --- set to 1 as an absoloute final resort
MHZ19 myMHZ19;
SoftwareSerial mySerial(25, 33);
//SoftwareSerial mySerial(RX_PIN,TX_PIN);

//NTP
const char *ntpServer1 = "pool.ntp.org";
const char *ntpServer2 = "hora.roa.es";
const char *ntpServer3 = "europe.pool.ntp.org";
const long gmtOffset_sec = 7200;
const int daylightOffset_sec = 0;

//wifi
char ssid[20] = "************"; // Aqui va ti identificador
char password[40] = "************"; //Aqui va tu contraseña

```

```
//thingspeak
unsigned long myChannelNumber = 1853883;
const char * myWriteAPIKey = "************";

//SPS30
#define SP30_COMMS SERIALPORT2
#define USE_I2C   // I2C using library wire (+10k code, 0k2 mem, 124 iram)
#define USE_SPS30

#define TX_PIN 17
#define RX_PIN 16

#define DEBUG 0

//#define DiTodos 0

//SD
const int chipSelect = 5;
//tiempos de lectura
unsigned long tiempo_sd = 5000;          //ms
unsigned long tiempo_sd_1min = 60000;    //ms
unsigned long tiempo_iot_1min = 60000;   //ms
unsigned long tiempo_sd_10min = 600000; //ms
unsigned long tiempo_hora = millis();    //ms
unsigned long tiempo_sps30 = millis();   //ms
unsigned long tiempo_bme = millis();     //ms
unsigned long timeco2 = millis();        //ms
unsigned long timeOLED = millis();       //ms

unsigned long lectco2 = 3000;          //ms
unsigned long lectOLEDrefresh = 5000; //ms
unsigned long lectOLEDmenu = 10000;    //ms
unsigned long lect_sd_1min = 45000;    //ms
unsigned long lect_sd_10min = 560000; //ms
unsigned long lect_sps30 = 3000;       //ms
unsigned long lect_bme = 3000;         //ms
unsigned long acthora = 604800000;     //Cada semana actualizamos la hora y
comprobamos el wifi

//IOT - thingspeak
WiFiClient  client;

// create constructor
SPS30 sps30;

//Hora
DateTime now;

```

```
//SD
File myFile;

//BME280
BME280<> BMESensor;

//PILA
RTC_DS3231 rtc;

//ESTRUCTURAS
struct sps30datos
{

  float medmassPM1;
  float medmassPM2;
  float medmassPM4;
  float medmassPM10;
  float medPartSize;

  float mednumPM0;
  float mednumPM1;
  float mednumPM2;
  float mednumPM4;
  float mednumPM10;
  float medidas;

} dato_sps30;

struct bme280datos
{
  float medpress;
  float medtemp;
  float medhum;
  float medidas;
  String horainic;

} dato_bme280;

struct mhz19b_datos
{
  float medco2;
  float medidas;

} dato_mhz19b;

struct Button{
  const uint8_t PIN;
  //BUTTON LONG PRESSED
```

```
  bool begin;
  unsigned long timebegin;
  unsigned long timeend;
};
//boton[0] abajo
//boton[1] arriba

int pantalla=0, opcion=1, mostrar=0, menu=0, menusd=0, priorizar=0;

//WIFI
int cambiarwifi=0 , menuwifi=0;

//PANTALLA
//Si pantalla=!1 datos
//Si pantalla=1 menu

//MODOS
int modowifiysd=0, modowifi=1, modosd=0;

//MODOS SD GUARDADO
int modosd10min=1, modosd1min=0, modosd5sec=0;

Button button0 = {32, true, 0, 0};
Button button1 = {27, true, 0, 0};

//datos globales pantalla
float pm1_sps,pm10_sps,pm25_sps,hum,temp,press;
int CO2=400, sps30notdetected=0;

void IRAM_ATTR longpressed0();
void IRAM_ATTR longpressed1();

void save_sd_todos();

void read_bme280();

void ErrtoMess(char *mess, uint8_t r);
void Errorloop(char *mess, uint8_t r);
void GetDeviceInfo();
bool read_sps30();

void mostrarmenu();
void mostrarendisplay();
void draw();
void drawLogo(void);

void printErrorCode();
void setRange(int range);
```

```
void read_co2();

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while (!Serial){} //Wait for serial port to connect

  //Botones
  pinMode(button0.PIN, INPUT_PULLUP);
  pinMode(button1.PIN, INPUT_PULLUP);
  attachInterrupt(button0.PIN, longpressed0, CHANGE);
  attachInterrupt(button1.PIN, longpressed1, CHANGE);

  //Pantalla
  u8g2.begin();
  u8g2.enableUTF8Print(); // enable UTF8 support for the Arduino print()
function
  delay(200);
  //logo
  u8g2.firstPage();
  do {
    drawLogo();
  } while ( u8g2.nextPage() );

  //MHZ19B
  mySerial.begin(9600); // Uno example: Begin Stream with MHZ19 baudrate
  myMHZ19.begin(mySerial); // *Imporant, Pass your Stream reference
  myMHZ19.autoCalibration(); // Turn auto calibration ON (OFF
autoCalibration(false))

  //RTC
  if (!rtc.begin())
  {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    abort();
  }
  else
  {
    Serial.println("El RTC funciona correctamente");
  }
  //FIN RTC

  //BME280
  BMESensor.begin();

  //SD
  Serial.println("Initializing SD card...");
```

```
263   if (!SD.begin(chipSelect))
264   {
265     Serial.println("Initialization failed!");
266     while (1);
267   }
268   Serial.println("initialization done.");
269 
270   //WIFI
271   Serial.println("Conectando... ");
272   Serial.println(ssid);
273 
274   WiFi.begin(ssid, password); // Intentamos la conexion a la WIFI
275   delay(5000);
276 
277   if (WiFi.status() != WL_CONNECTED)
278   {
279     Serial.println("Aun no se ha conectado al wifi");
280       delay(5000);
281   }
282   else
283   {
284     Serial.println("WiFi connected..!");
285     Serial.println("");
286 
287     // Initialize ThingSpeak
288     ThingSpeak.begin(client);
289     // Initialize Blynk
290     Blynk.connectWiFi(ssid, password);
291     Blynk.config(auth, BLYNK_DEFAULT_DOMAIN, BLYNK_DEFAULT_PORT);
292     if(Blynk.connect() != true){delay(1000);}
293   } //FIN WIFI
294 
295   ////////////////////////////////////////////////////
296   //init and get the time
297    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer1, ntpServer2,
298 ntpServer3);
299   ////////////////////////////////////////////////////
300 
301   //RTC
302   if (rtc.lostPower()) {
303     //Si se ha ido la luz no entrA
304     rtc.adjust(DateTime(2022, 1, 1, 0, 0, 0));
305   }
306 
307   //SPS30
308   Serial.println(F("Trying to connect to sps30"));
309   // set driver debug level
310   sps30.EnableDebugging(DEBUG);
```

```
311
312   // set pins to use for softserial and Serial1 on ESP32
313   if (TX_PIN != 0 && RX_PIN != 0) sps30.SetSerialPin(RX_PIN,TX_PIN);
314
315   // Begin communication channel;
316   if (! sps30.begin(SP30_COMMS))
317     Errorloop((char *) "could not initialize communication channel.", 0);
318
319   // check for SPS30 connection
320   if (! sps30.probe()) {
321       //Errorloop((char *) "could not probe / connect with SPS30.", 0);
322       Serial.print("could not probe / connect with SPS30.");
323       sps30notdetected=1;
324   }
325   else  Serial.println(F("Detected SPS30."));
326
327   // reset SPS30 connection
328   if (! sps30.reset()) Errorloop((char *) "could not reset.", 0);
329
330   // read device info
331   if(!sps30notdetected) {
332
333     GetDeviceInfo();
334
335     // start measurement
336     if (sps30.start()) Serial.println(F("Measurement started"));
337     else Errorloop((char *) "Could NOT start measurement", 0);
338
339     delay(300);
340
341     if (SP30_COMMS == I2C_COMMS) {
342       if (sps30.I2C_expect() == 4)
343         Serial.println(F(" !!! Due to I2C buffersize only the SPS30 MASS
344 concentration is available !!! \n"));
345     }
346   }
347
348 }
349
350 void loop() {
351   // put your main code here, to run repeatedly:
352   //Hora
353   now = rtc.now();
354   if (cambiarwifi)
355   {
356       WiFi.disconnect();
357       delay(500);
358       WiFi.begin(ssid, password); // Cambiamos la conexion WIFI
```

```
      tiempo_hora=millis();
      delay(200);
      cambiarwifi=0;
  }
  if ((millis() >= tiempo_hora) && (priorizar!=1) && (modowifi==1 ||
modowifiysd==1))
  {
    if ((WiFi.status() != WL_CONNECTED))
    {
      delay(200);
      WiFi.begin(ssid, password); // Intentamos la conexion a la WIFI
      tiempo_hora = millis() + 30000; //Si falla al obtener la hora
volvemos a intentarlo 30 seg despues
    }
    else
    {
      // Initialize ThingSpeak
      ThingSpeak.begin(client);
      // Initialize Blynk
      Blynk.connectWiFi(ssid, password);
      Blynk.config(auth, BLYNK_DEFAULT_DOMAIN, BLYNK_DEFAULT_PORT);
      if(Blynk.connect() != true){delay(1000);}

      time_t timeepoch;
      struct tm timeinfo;

      Serial.println("Estoy conectado al wifi");
      if (!getLocalTime(&timeinfo))
      {
        Serial.println("Failed to obtain time");
      }
      else
      {
        time(&timeepoch);
        timeepoch = timeepoch + 7200; // GTM + 1
        rtc.adjust(timeepoch);
        delay(100);
        now = rtc.now();
        tiempo_hora = millis() + acthora;
      }
    }
  }

  if((millis()>timeco2) && (priorizar!=1))
  {
    read_co2();
    timeco2=millis()+lectco2;
  }
```

```
  //SPS30
  if ((millis() >= tiempo_sps30) && (priorizar!=1) && (!sps30notdetected))
  {
    read_sps30();
    tiempo_sps30 = millis() + lect_sps30;
  }

  //BME280
  if ((millis() >= tiempo_bme) && (priorizar!=1))
  {
    read_bme280();
    tiempo_bme = millis() + lect_bme;
  }

  if(millis()>timeOLED)
  {
    priorizar=0;
    mostrarendisplay();
    switch(pantalla){
      case 0:
        pantalla=2;
      break;
      case 2:
        pantalla=3;
      break;
      case 3:
        pantalla=0;
      break;
    }
  }

  //WIFI - IOT
  if((modowifiysd || modowifi) && (millis() >= tiempo_iot_1min) &&
(now.second() <= 5 || now.second() >= 55) && (dato_bme280.medidas > 0.0) &&
(dato_mhz19b.medidas > 0.0) && (dato_sps30.medidas > 0.0) &&
(priorizar!=1))
  {
    if (WiFi.status() != WL_CONNECTED)
    {
      delay(200);
      WiFi.begin(ssid, password); // Intentamos la conexion a la WIFI
      tiempo_hora = millis() + 30000;//Si falla al obtener la hora volvemos
a intentarlo 30 seg despues
      delay(500);
      if (WiFi.status() == WL_CONNECTED){
        // Initialize ThingSpeak
        ThingSpeak.begin(client);
```

```
        // Initialize Blynk
        Blynk.connectWiFi(ssid, password);
        Blynk.config(auth, BLYNK_DEFAULT_DOMAIN, BLYNK_DEFAULT_PORT);
        if(Blynk.connect() != true){delay(1000);}
      }
    }
    else
    {
          Blynk.run();
          // set the fields with the values
          ThingSpeak.setField(1, hum);
          ThingSpeak.setField(2, temp);
          ThingSpeak.setField(3, press);
          ThingSpeak.setField(4, CO2);
          ThingSpeak.setField(5, pm1_sps);
          ThingSpeak.setField(6, pm25_sps);
          ThingSpeak.setField(7, pm10_sps);

          Blynk.virtualWrite(V1, hum);
          Blynk.virtualWrite(V2, temp);
          Blynk.virtualWrite(V3, press);
          Blynk.virtualWrite(V4, CO2);
          Blynk.virtualWrite(V5, pm1_sps);
          Blynk.virtualWrite(V6, pm25_sps);
          Blynk.virtualWrite(V7, pm10_sps);

          ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
          Serial.println("Datos subidos a thingiverse y blynk");

          tiempo_iot_1min = millis() + lect_sd_1min;
    }
  }

  //SD
  if((modowifiysd || modosd) && (priorizar!=1))
  {
    if ((modosd5sec) && (dato_bme280.medidas > 0.0) && (dato_sps30.medidas
> 0.0)  && (dato_mhz19b.medidas > 0.0) && (millis() >= tiempo_sd))
    {
      save_sd_todos();
    }

    if ((modosd1min) && (millis() >= tiempo_sd_1min) && (now.second() <= 5
|| now.second() >= 55)  && (dato_mhz19b.medidas > 0.0) &&
(dato_bme280.medidas > 0.0) && (dato_sps30.medidas > 0.0) )
    {
      save_sd_todos();
      tiempo_sd_1min = millis() + lect_sd_1min;
```

```
    }

    if ((modosd10min) && (millis() >= tiempo_sd_10min) && (now.minute() %
10 == 0) && (now.second() <= 10 || now.second() >= 50)  &&
(dato_mhz19b.medidas > 0.0) && (dato_bme280.medidas > 0.0) &&
(dato_sps30.medidas > 0.0) )
    {
      save_sd_todos();
      tiempo_sd_10min = millis() + lect_sd_10min;
    }
  }
}

//botones
void IRAM_ATTR longpressed0()
{
  if(millis()>button0.timeend+100 && millis()>button0.timebegin+50)
  {
    if (button0.begin==true)
    {
      button0.timebegin=millis();
      button0.timeend=0;
      button0.begin=false;
    }
    else
    {
      button0.timeend=millis();
      button0.begin=true;
    }

    if (button0.begin==true)
    {
        timeOLED = millis();
        priorizar=1;

        if (button0.timeend-button0.timebegin > 800)
        {
          //Serial.println("Pulsacion larga - Boton 0");
          if(menu!=1){pantalla=1; menu=1;}
          else if(menusd){
          switch(opcion)
            {
              case 1:
                modosd10min=1;
                modosd1min=0;
                modosd5sec=0;
                opcion=1;
                //Serial.println("Modo 10 min");
```

```
            break;
            case 2:
              modosd10min=0;
              modosd1min=1;
              modosd5sec=0;
              opcion=1;
              //Serial.println("Modo 1 min");
            break;
            case 3:
              modosd10min=0;
              modosd1min=0;
              modosd5sec=1;
              opcion=1;
              //Serial.println("Modo 5 sec");
            break;
          }if(modowifi || modowifiysd)
          {
            pantalla=5;
            menu=1;
            menusd=0;
            menuwifi=1;
          }
          else
          {
            pantalla=0;
            menu=0;
            menusd=0;
          }
        }
        else if(menuwifi)
        {
          switch(opcion)
          {
            case 1:
              strcpy(ssid,"BARACOA"); // Aqui va ti identificador
              strcpy(password, "************"); //Aqui va tu contraseña
            break;
            case 2:
              strcpy(ssid,"DN");// Aqui va ti identificador
              strcpy(password, "************"); //Aqui va tu contraseña
            break;
          }
          tiempo_hora=millis();
          pantalla=0;
          menu=0;
          menuwifi=0;
          opcion=1;
          cambiarwifi=1;
```

```
          }
          else
          {
            switch(opcion)
            {
              case 1:
                modowifi=1;
                modosd=0;
                modowifiysd=0;
                opcion=1;
                pantalla=5;
                menu=1;
                menusd=0;
                menuwifi=1;
              break;
              case 2:
                modowifi=0;
                modosd=1;
                modowifiysd=0;
                opcion=1;
                menusd=1;
                pantalla=4;
              break;
              case 3:
                modowifi=0;
                modosd=0;
                modowifiysd=1;
                opcion=1;
                menusd=1;
                pantalla=4;
              break;
            }
          }
        }
        else
        {
          //Serial.println("Pulsacion corta - Boton 0");
          if(menu==1){
            if(opcion>1)
              opcion--;
          }
        }
    }

  }//ifglobal
}//longpressed1

void IRAM_ATTR longpressed1()
```

```
{
  if(millis()>button1.timeend+100 && millis()>button1.timebegin+50)
  {

    if (button1.begin==true)
    {
      button1.timebegin=millis();
      button1.timeend=0;
      button1.begin=false;
    }else
    {
      button1.timeend=millis();
      button1.begin=true;
    }

    if (button1.begin==true)
    {
        timeOLED = millis();
        priorizar=1;

        if (button1.timeend-button1.timebegin > 800)
        {
          //Serial.println("Pulsacion larga - Boton 1");
          if(menu!=1){pantalla=1; menu=1;}
          else if(menusd)
          {
            switch(opcion)
            {
              case 1:
                modosd10min=1;
                modosd1min=0;
                modosd5sec=0;
                opcion=1;
                //Serial.println("Modo 10 min");
              break;
              case 2:
                modosd10min=0;
                modosd1min=1;
                modosd5sec=0;
                opcion=1;
                //Serial.println("Modo 1 min");
              break;
              case 3:
                modosd10min=0;
                modosd1min=0;
                modosd5sec=1;
                opcion=1;
                //Serial.println("Modo 5 sec");
```

```
695             break;
696           }
697           if(modowifi || modowifiysd)
698           {
699             pantalla=5;
700             menu=1;
701             menusd=0;
702             menuwifi=1;
703           }
704           else{
705             pantalla=0;
706             menu=0;
707             menusd=0;
708           }
709         }
710         else if(menuwifi)
711         {
712           switch(opcion)
713           {
714             case 1:
715               strcpy(ssid,"BARACOA"); // Aqui va ti identificador
716               strcpy(password, "************"); //Aqui va tu contraseña
717             break;
718             case 2:
719               strcpy(ssid,"DN"); // Aqui va ti identificador
720               strcpy(password, "************"); //Aqui va tu contraseña
721             break;
722           }
723           tiempo_hora=millis();
724           pantalla=0;
725           menu=0;
726           menuwifi=0;
727           opcion=1;
728           cambiarwifi=1;
729         }
730         else
731         {
732           switch(opcion)
733           {
734             case 1:
735               modowifi=1;
736               modosd=0;
737               modowifiysd=0;
738               opcion=1;
739               pantalla=5;
740               menu=1;
741               menusd=0;
742               menuwifi=1;
```

```
              break;
              case 2:
                modowifi=0;
                modosd=1;
                modowifiysd=0;
                opcion=1;
                menusd=1;
                pantalla=4;
              break;
              case 3:
                modowifi=0;
                modosd=0;
                modowifiysd=1;
                opcion=1;
                menusd=1;
                pantalla=4;
              break;
            }
          }
        }
        else
        {
          //Serial.println("Pulsacion corta - Boton 1");
          if(menu==1){
            if(opcion<3)
              opcion++;
          }
        }
    }

  }//ifglobal
}//longpressed1

void mostrarendisplay()
{
  u8g2.firstPage();
  do {
    draw();
  } while (u8g2.nextPage());

    if(pantalla==0)
    {
      timeOLED = millis() + lectOLEDrefresh;
    }else
    {
      timeOLED = millis() + lectOLEDmenu;
    }
}
```

```
791
792 void drawLogo(void)
793 {
794   int offset = 40;
795
796     u8g2.setFontMode(1);  // Transparent
797     u8g2.setDrawColor(1);
798
799     u8g2.drawDisc(20, 20, 12, U8G2_DRAW_ALL);
800     u8g2.drawDisc(20, 30, 12, U8G2_DRAW_ALL);
801
802     u8g2.setFontDirection(0);
803     u8g2.setFont(u8g2_font_inb30_mf);
804     u8g2.drawStr(5+offset, 47, "L");
805
806     u8g2.setFontDirection(0);
807     u8g2.setFont(u8g2_font_inb30_mf);
808     u8g2.drawStr(27+offset, 47,"C");
809
810     u8g2.setFontDirection(0);
811     u8g2.setFont(u8g2_font_inb30_mf);
812     u8g2.drawStr(51+offset, 47,"A");
813
814     u8g2.setFontDirection(0);
815     u8g2.setFont( u8g2_font_6x13_tr);
816     u8g2.drawStr(10, 64,"TFG - D. Nescolarde");
817
818 }
819
820 void draw()
821 {
822   char date_string[32] = "DD/MM/YY - hh:mm";
823   char hour_string[32] = "hh:mm";
824   String fecha = (String)now.toString(date_string);
825   String hora = (String)now.toString(hour_string);
826   int offset=25, offsetdecimal=0;
827
828   switch(pantalla)
829   {
830
831     case 0:
832       //PANTALLA
833
834       //PARTE SUPERIOR
835           u8g2.setFontMode(1); /* activate transparent font mode */
836           u8g2.setDrawColor(2); /* color 2 for the text */
837           u8g2.setFont(u8g2_font_ncenB08_tr);
838           u8g2.setCursor(0,12);
```

```
839           u8g2.print(hora);
840
841           if(modowifiysd){
842
843           u8g2.setCursor(44,12);
844           u8g2.print(F("DATOS"));
845
846             //ICONO SD
847             u8g2.setFont(u8g2_font_timR08_tr );
848             u8g2.setCursor(100,12);
849             u8g2.print(F("SD"));
850
851
852             //ICONO +
853             u8g2.setFont(u8g2_font_timR08_tr );
854             u8g2.setCursor(113,12);
855             u8g2.print(F("+"));
856
857             //ICONO WIFI
858             if(WiFi.status() != WL_CONNECTED){
859               u8g2.drawLine(128, 4, 118, 13);
860             }
861             u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
862             u8g2.drawGlyph(120,12, 81);
863           }
864           else if(modosd){
865
866           u8g2.setCursor(49,12);
867           u8g2.print(F("DATOS"));
868
869             //ICONO SD
870             u8g2.setFont(u8g2_font_timR08_tr );
871             u8g2.setCursor(110,12);
872             u8g2.print(F("SD"));
873           }else{
874
875           u8g2.setCursor(49,12);
876           u8g2.print(F("DATOS"));
877
878             //ICONO WIFI
879             if(WiFi.status() != WL_CONNECTED){
880               u8g2.drawLine(118, 4, 108, 13);
881             }
882             u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
883             u8g2.drawGlyph(110,12, 81);
884           }
885
886           //LINEA
```

```
        u8g2.setDrawColor(2); /* color 2 for the text */
        u8g2.drawLine(0, 16, 128, 16);
        ////////////////////
        
        u8g2.setFont(u8g2_font_open_iconic_weather_4x_t);
        u8g2.drawGlyph(10,62, 69);
        
        u8g2.setFont(u8g2_font_ncenB08_tr);
        u8g2.setCursor(offset+42,30);
        u8g2.print(hum);
        u8g2.print(F(" %"));
        u8g2.setCursor(offset+42,46);
        u8g2.print(temp);
        u8g2.print(" ");
        u8g2.setFont(u8g2_font_unifont_t_symbols);
        u8g2.drawGlyph(offset+68,46, 176);
        u8g2.setFont(u8g2_font_ncenB08_tr);
        u8g2.setCursor(offset+74,46);
        u8g2.println(F("C"));
        u8g2.setCursor(offset+30,62);
        u8g2.print(press);
        u8g2.print(F(" hPa"));
    break;

    case 2:
      //AEROSOLES

         //PARTE SUPERIOR
        u8g2.setFontMode(1);  /* activate transparent font mode */
        u8g2.setDrawColor(2); /* color 2 for the text */
        u8g2.setFont(u8g2_font_ncenB08_tr);
        u8g2.setCursor(0,12);
        u8g2.print(hora);

        if(modowifiysd){

        u8g2.setCursor(44,12);
        u8g2.print(F("DATOS"));

          //ICONO SD
          u8g2.setFont(u8g2_font_timR08_tr );
          u8g2.setCursor(100,12);
          u8g2.print(F("SD"));


          //ICONO +
          u8g2.setFont(u8g2_font_timR08_tr );
          u8g2.setCursor(113,12);
```

```
            u8g2.print(F("+"));

            //ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(128, 4, 118, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(120,12, 81);
          }
          else if(modosd){

          u8g2.setCursor(49,12);
          u8g2.print(F("DATOS"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(110,12);
            u8g2.print(F("SD"));
          }else{

          u8g2.setCursor(49,12);
          u8g2.print(F("DATOS"));

            //ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(118, 4, 108, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(110,12, 81);
          }

          //LINEA
          u8g2.setDrawColor(2); /* color 2 for the text */
          u8g2.drawLine(0, 16, 128, 16);
          ////////////////////////

              u8g2.setFont(u8g2_font_ncenB08_tr);
              u8g2.setCursor(15,30);
              u8g2.print(F("PM1: "));
              u8g2.print(pm1_sps);
              //
                if((pm1_sps/10)>1.0){offsetdecimal=5;}
                else{offsetdecimal=0;}
                u8g2.print(" ");
                u8g2.setFont(u8g2_font_6x12_t_symbols);
                u8g2.drawGlyph(72+offsetdecimal,30, 181);
                u8g2.setCursor(75+offsetdecimal,30);
                u8g2.setFont(u8g2_font_ncenB08_tr);
```

```
              u8g2.print(F(" g/m3"));
            //
            u8g2.setCursor(15,46);
            u8g2.print(F("PM2.5: "));
            u8g2.print(pm25_sps);
            //
              if((pm25_sps/10)>1.0){offsetdecimal=5;}
              else{offsetdecimal=0;}
              u8g2.print(" ");
              u8g2.setFont(u8g2_font_6x12_t_symbols);
              u8g2.drawGlyph(80+offsetdecimal,46, 181);
              u8g2.setCursor(83+offsetdecimal,46);
              u8g2.setFont(u8g2_font_ncenB08_tr);
              u8g2.print(F(" g/m3"));
            //
            u8g2.setCursor(15,62);
            u8g2.print(F("PM10: "));
            u8g2.print(pm10_sps);
            //
              if((pm10_sps/10)>1.0){offsetdecimal=5;}
              else{offsetdecimal=0;}
              u8g2.print(" ");
              u8g2.setFont(u8g2_font_6x12_t_symbols);
              u8g2.drawGlyph(77+offsetdecimal,62, 181);
              u8g2.setCursor(80+offsetdecimal,62);
              u8g2.setFont(u8g2_font_ncenB08_tr);
              u8g2.print(F(" g/m3"));
            //
    break;

    case 3:
      //CO2

      //PARTE SUPERIOR
          u8g2.setFontMode(1);  /* activate transparent font mode */
          u8g2.setDrawColor(2); /* color 2 for the text */
          u8g2.setFont(u8g2_font_ncenB08_tr);
          u8g2.setCursor(0,12);
          u8g2.print(hora);

          if(modowifiysd){

          u8g2.setCursor(44,12);
          u8g2.print(F("DATOS"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(100,12);
```

```
            u8g2.print(F("SD"));


            //ICONO +
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(113,12);
            u8g2.print(F("+"));

            //ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(128, 4, 118, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(120,12, 81);
          }
          else if(modosd){

          u8g2.setCursor(49,12);
          u8g2.print(F("DATOS"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(110,12);
            u8g2.print(F("SD"));
          }else{

          u8g2.setCursor(49,12);
          u8g2.print(F("DATOS"));

            //ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(118, 4, 108, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(110,12, 81);
          }

          //LINEA
          u8g2.setDrawColor(2); /* color 2 for the text */
          u8g2.drawLine(0, 16, 128, 16);
          ////////////////////

          u8g2.setFont(u8g2_font_open_iconic_weather_4x_t);
          u8g2.drawGlyph(20,62, 64);

          u8g2.setFont(u8g2_font_ncenB08_tr);
          u8g2.setCursor(offset+50,40);
          u8g2.print(F("CO2"));
```

```
1079          u8g2.setCursor(offset+38,56);
1080          u8g2.print(CO2);
1081          u8g2.print(F(" ppm"));
1082      break;
1083
1084      case 1:
1085        //MENU DE SELECCIÓN
1086        u8g2.setDrawColor(1); /* color 1 for the box */
1087        switch(opcion){
1088          case 1:
1089            u8g2.drawBox(0, 16, 128, 16);
1090          break;
1091          case 2:
1092            u8g2.drawBox(0, 32, 128, 16);
1093          break;
1094          case 3:
1095            u8g2.drawBox(0, 48, 128, 16);
1096          break;
1097        }
1098            //PARTE SUPERIOR
1099            u8g2.setFontMode(1);  /* activate transparent font mode */
1100            u8g2.setDrawColor(2); /* color 2 for the text */
1101            u8g2.setFont(u8g2_font_ncenB08_tr);
1102            u8g2.setCursor(0,12);
1103            u8g2.print(hora);
1104
1105            if(modowifiysd){
1106
1107            u8g2.setCursor(44,12);
1108            u8g2.print(F("MENU"));
1109
1110              //ICONO SD
1111              u8g2.setFont(u8g2_font_timR08_tr );
1112              u8g2.setCursor(100,12);
1113              u8g2.print(F("SD"));
1114
1115
1116              //ICONO +
1117              u8g2.setFont(u8g2_font_timR08_tr );
1118              u8g2.setCursor(113,12);
1119              u8g2.print(F("+"));
1120
1121              //ICONO WIFI
1122              if(WiFi.status() != WL_CONNECTED){
1123                u8g2.drawLine(128, 4, 118, 13);
1124              }
1125              u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
1126              u8g2.drawGlyph(120,12, 81);
```

```
1127          }
1128          else if(modosd){
1129
1130          u8g2.setCursor(49,12);
1131          u8g2.print(F("MENU"));
1132
1133            //ICONO SD
1134            u8g2.setFont(u8g2_font_timR08_tr );
1135            u8g2.setCursor(110,12);
1136            u8g2.print(F("SD"));
1137          }else{
1138
1139          u8g2.setCursor(49,12);
1140          u8g2.print(F("MENU"));
1141
1142            ///ICONO WIFI
1143            if(WiFi.status() != WL_CONNECTED){
1144              u8g2.drawLine(118, 4, 108, 13);
1145            }
1146            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
1147            u8g2.drawGlyph(110,12, 81);
1148          }
1149
1150        //LINEA
1151        u8g2.drawLine(0, 16, 128, 16);
1152
1153        //OPCIONES
1154        u8g2.setFont(u8g2_font_ncenB08_tr);
1155        u8g2.setCursor(30,30);
1156        u8g2.print(F("MODO WIFI"));
1157        u8g2.setCursor(34,46);
1158        u8g2.print(F("MODO SD"));
1159        u8g2.setCursor(19,62);
1160        u8g2.print(F("MODO WIFI+SD"));
1161
1162      break;
1163
1164      case 4:
1165        //MENU DE SD
1166        u8g2.setDrawColor(1); /* color 1 for the box */
1167        switch(opcion){
1168          case 1:
1169            u8g2.drawBox(0, 16, 128, 16);
1170          break;
1171          case 2:
1172            u8g2.drawBox(0, 32, 128, 16);
1173          break;
1174          case 3:
```

```
          u8g2.drawBox(0, 48, 128, 16);
        break;
      }
          //PARTE SUPERIOR
          u8g2.setFontMode(1);  /* activate transparent font mode */
          u8g2.setDrawColor(2); /* color 2 for the text */
          u8g2.setFont(u8g2_font_ncenB08_tr);
          u8g2.setCursor(0,12);
          u8g2.print(hora);

          if(modowifiysd){

          u8g2.setCursor(44,12);
          u8g2.print(F("MENU"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(100,12);
            u8g2.print(F("SD"));


            //ICONO +
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(113,12);
            u8g2.print(F("+"));

            //ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(128, 4, 118, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(120,12, 81);
          }
          else if(modosd){

          u8g2.setCursor(49,12);
          u8g2.print(F("MENU"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(110,12);
            u8g2.print(F("SD"));
          }else{

          u8g2.setCursor(49,12);
          u8g2.print(F("MENU"));

            ///ICONO WIFI
```

```
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(118, 4, 108, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(110,12, 81);
          }

      //LINEA
      u8g2.drawLine(0, 16, 128, 16);

      //OPCIONES
        u8g2.setFont(u8g2_font_ncenB08_tr);
        u8g2.setCursor(30,30);
        u8g2.print(F("10 min"));
        u8g2.setCursor(30,46);
        u8g2.print(F("1 min"));
        u8g2.setCursor(30,62);
        u8g2.print(F("5 sec"));
    break;

    case 5:
      //MENU DE WIFI
      u8g2.setDrawColor(1); /* color 1 for the box */
      switch(opcion){
        case 1:
          u8g2.drawBox(0, 16, 128, 16);
        break;
        case 2:
          u8g2.drawBox(0, 32, 128, 16);
        break;
        case 3:
          u8g2.drawBox(0, 48, 128, 16);
        break;
      }
          //PARTE SUPERIOR
          u8g2.setFontMode(1);  /* activate transparent font mode */
          u8g2.setDrawColor(2); /* color 2 for the text */
          u8g2.setFont(u8g2_font_ncenB08_tr);
          u8g2.setCursor(0,12);
          u8g2.print(hora);

          if(modowifiysd){

          u8g2.setCursor(44,12);
          u8g2.print(F("MENU"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
```

```
            u8g2.setCursor(100,12);
            u8g2.print(F("SD"));


            //ICONO +
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(113,12);
            u8g2.print(F("+"));

            //ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(128, 4, 118, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(120,12, 81);
          }
          else if(modosd){

          u8g2.setCursor(49,12);
          u8g2.print(F("MENU"));

            //ICONO SD
            u8g2.setFont(u8g2_font_timR08_tr );
            u8g2.setCursor(110,12);
            u8g2.print(F("SD"));
          }else{

          u8g2.setCursor(49,12);
          u8g2.print(F("MENU"));

            ///ICONO WIFI
            if(WiFi.status() != WL_CONNECTED){
              u8g2.drawLine(118, 4, 108, 13);
            }
            u8g2.setFont(u8g2_font_open_iconic_www_1x_t);
            u8g2.drawGlyph(110,12, 81);
          }

      //LINEA
      u8g2.drawLine(0, 16, 128, 16);

      //OPCIONES
        u8g2.setFont(u8g2_font_ncenB08_tr);
        u8g2.setCursor(30,30);
        u8g2.print(F("BARACOA"));
        u8g2.setCursor(30,46);
        u8g2.print(F("MOVIL"));
    break;
```

```
  }

}

void save_sd_todos()
{

  static bool header = true;

  dato_sps30.medmassPM1 = dato_sps30.medmassPM1 / dato_sps30.medidas;
  dato_sps30.medmassPM2 = dato_sps30.medmassPM2 / dato_sps30.medidas;
  dato_sps30.medmassPM4 = dato_sps30.medmassPM4 / dato_sps30.medidas;
  dato_sps30.medmassPM10 = dato_sps30.medmassPM10 / dato_sps30.medidas;
  dato_sps30.medPartSize = dato_sps30.medPartSize / dato_sps30.medidas;

  dato_sps30.mednumPM0 = dato_sps30.mednumPM0 / dato_sps30.medidas;
  dato_sps30.mednumPM1 = dato_sps30.mednumPM1 / dato_sps30.medidas;
  dato_sps30.mednumPM2 = dato_sps30.mednumPM2 / dato_sps30.medidas;
  dato_sps30.mednumPM4 = dato_sps30.mednumPM4 / dato_sps30.medidas;
  dato_sps30.mednumPM10 = dato_sps30.mednumPM10 / dato_sps30.medidas;

  dato_bme280.medhum = dato_bme280.medhum / dato_bme280.medidas;
  dato_bme280.medtemp = dato_bme280.medtemp / dato_bme280.medidas;
  dato_bme280.medpress = dato_bme280.medpress / dato_bme280.medidas;

  dato_mhz19b.medco2 = dato_mhz19b.medco2 / dato_mhz19b.medidas;

  if (header)
  {
    if(modosd5sec){
      char date_string[32] = "DD_MM_YY-hh_mm_ss-";
      dato_bme280.horainic = "/" + (String)now.toString(date_string);
      dato_bme280.horainic = dato_bme280.horainic + "datafile.csv";
      myFile = SD.open(dato_bme280.horainic, FILE_WRITE);
    }else if(modosd10min){
      char date_string[32] = "DD_MM_YY-hh_mm_ss-";
      dato_bme280.horainic = "/" + (String)now.toString(date_string);
      dato_bme280.horainic = dato_bme280.horainic + "datafile10min.csv";
      myFile = SD.open(dato_bme280.horainic, FILE_WRITE);
    }
    else if(modosd1min){
      char date_string[32] = "DD_MM_YY-hh_mm_ss-";
      dato_bme280.horainic = "/" + (String)now.toString(date_string);
      dato_bme280.horainic = dato_bme280.horainic + "datafile1min.csv";
      myFile = SD.open(dato_bme280.horainic, FILE_WRITE);
    }
  }
  else
```

```
  {
    //Para no sobre escribir
    myFile = SD.open(dato_bme280.horainic, FILE_APPEND);
  }

  if (myFile)
  {
    Serial.print("opened datafile.csv...");
    if (header)
    {
      ///////////HORA///////////////
      myFile.print("Date");
      myFile.print(",");
      myFile.print("Time");
      myFile.print(",");
      ///////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////SPS30_1///////////////
      myFile.print("Mass");
      myFile.print(",");
      myFile.print("Mass");
      myFile.print(",");
      myFile.print("Mass");
      myFile.print(",");
      myFile.print("Mass");
      myFile.print(",");
      myFile.print("Number");
      myFile.print(",");
      myFile.print("Number");
      myFile.print(",");
      myFile.print("Number");
      myFile.print(",");
      myFile.print("Number");
      myFile.print(",");
      myFile.print("Number");
      myFile.print(",");
      myFile.print("Average");
      myFile.print(",");
      ///////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////DHT11///////////////
      myFile.print("Temperature");
      myFile.print(",");
      myFile.print("Humidity");
      myFile.print(",");
      myFile.print("Presion");
```

```
      myFile.print(",");
      ///////////////SALTO////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////////MHZ19B////////////////
      myFile.println("CO2");
      ////////////FILA 2///////////////
      myFile.print("[DD/MM/AAAA]");
      myFile.print(",");
      myFile.print("[HH/mm/ss]");
      myFile.print(",");
      ///////////////SALTO////////////////
      myFile.print("      ");
      myFile.print(",");
      ////////////SPS30_2///////////////
      myFile.print("[microgramo/m3]");
      myFile.print(",");
      myFile.print("[microgramo/m3]");
      myFile.print(",");
      myFile.print("[microgramo/m3]");
      myFile.print(",");
      myFile.print("[microgramo/m3]");
      myFile.print(",");
      myFile.print("[#/m3]");
      myFile.print(",");
      myFile.print("[#/m3]");
      myFile.print(",");
      myFile.print("[#/m3]");
      myFile.print(",");
      myFile.print("[#/m3]");
      myFile.print(",");
      myFile.print("[#/m3]");
      myFile.print(",");
      myFile.print("[microgramo]");
      myFile.print(",");
      ///////////////SALTO////////////////
      myFile.print("      ");
      myFile.print(",");
      ////////////BME280///////////////
      myFile.print(F("[ºC]"));
      myFile.print(",");
      myFile.print("[%]");
      myFile.print(",");
      myFile.print("[hPa]");
      myFile.print(",");
      ///////////////SALTO////////////////
      myFile.print("      ");
      myFile.print(",");
```

```
      ///////////////MHZ19B/////////////////
      myFile.println("ppm");
      ///////////FILA 3////////////
      myFile.print("");
      myFile.print(",");
      myFile.print("");
      myFile.print(",");
      ///////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////SPS30_2///////////////
      myFile.print("P1.0");
      myFile.print(",");
      myFile.print("P2.5");
      myFile.print(",");
      myFile.print("P4.0");
      myFile.print(",");
      myFile.print("P10");
      myFile.print(",");
      myFile.print("P0.5");
      myFile.print(",");
      myFile.print("P1.0");
      myFile.print(",");
      myFile.print("P2.5");
      myFile.print(",");
      myFile.print("P4.0");
      myFile.print(",");
      myFile.print("P10");
      myFile.print(",");
      myFile.print("PartSize");
      myFile.print(",");
      ///////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////BME280///////////////
      myFile.print(" ");
      myFile.print(",");
      myFile.print(" ");
      myFile.print(",");
      myFile.print(" ");
      myFile.print(",");
      ///////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ////////////////////////////////
      myFile.println(" ");

      header = false;
```

```
    }

    char date_string[32] = "DD/MM/YY";
    String fecha = (String)now.toString(date_string);

    char hour_string[32] = "hh:mm:ss";
    String hora = (String)now.toString(hour_string);

    myFile.print(fecha);
    myFile.print(",");
    myFile.print(hora);
    myFile.print(",");
    ////////////////SALTO/////////////////
    myFile.print("      ");
    myFile.print(",");
    ///////////SPS30_2///////////////
    myFile.print(dato_sps30.medmassPM1);
    myFile.print(",");
    myFile.print(dato_sps30.medmassPM2);
    myFile.print(",");
    myFile.print(dato_sps30.medmassPM4);
    myFile.print(",");
    myFile.print(dato_sps30.medmassPM10);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM0);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM1);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM2);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM4);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM10);
    myFile.print(",");
    myFile.print(dato_sps30.medPartSize);
    myFile.print(",");
    ////////////////SALTO/////////////////
    myFile.print("      ");
    myFile.print(",");
    ///////////BME280///////////////
    myFile.print(dato_bme280.medtemp);
    myFile.print(",");
    myFile.print(dato_bme280.medhum);
    myFile.print(",");
    myFile.print(dato_bme280.medpress);
    myFile.print(",");
    ////////////////SALTO/////////////////
    myFile.print("      ");
```

```
    myFile.print(",");
    ////////////////MHZ19B////////////////
    myFile.println(dato_mhz19b.medco2);

    // close the file:
    myFile.close();
    Serial.println("closed datafile.csv.");
  }
  else
  {
    // if the file didn't open, print an error:
    Serial.println("error opening datafile.csv");
  }

  //Ponemos los valores de las medias a 0
  dato_sps30.medmassPM1 = 0;
  dato_sps30.medmassPM2 = 0;
  dato_sps30.medmassPM4 = 0;
  dato_sps30.medmassPM10 = 0;
  dato_sps30.medPartSize = 0;

  dato_sps30.mednumPM0 = 0;
  dato_sps30.mednumPM1 = 0;
  dato_sps30.mednumPM2 = 0;
  dato_sps30.mednumPM4 = 0;
  dato_sps30.mednumPM10 = 0;
  dato_sps30.medidas = 0;

  dato_bme280.medhum = 0;
  dato_bme280.medidas = 0;
  dato_bme280.medtemp = 0;
  dato_bme280.medpress = 0;

  dato_mhz19b.medco2 = 0;
  dato_mhz19b.medidas = 0;

}

void GetDeviceInfo()
{
  char buf[32];
  uint8_t ret;
  SPS30_version v;

  //try to read serial number
  ret = sps30.GetSerialNumber(buf, 32);
  if (ret == SPS30_ERR_OK) {
    Serial.print(F("Serial number : "));
```

```
    if(strlen(buf) > 0)  Serial.println(buf);
    else Serial.println(F("not available"));
  }
  else
    ErrtoMess((char *) "could not get serial number", ret);

  // try to get product name
  ret = sps30.GetProductName(buf, 32);
  if (ret == SPS30_ERR_OK)  {
    Serial.print(F("Product name  : "));

    if(strlen(buf) > 0)  Serial.println(buf);
    else Serial.println(F("not available"));
  }
  else
    ErrtoMess((char *) "could not get product name.", ret);

  // try to get version info
  ret = sps30.GetVersion(&v);
  if (ret != SPS30_ERR_OK) {
    Serial.println(F("Can not read version info"));
    return;
  }

  Serial.print(F("Firmware level: "));  Serial.print(v.major);
  Serial.print("."); Serial.println(v.minor);

  if (SP30_COMMS != I2C_COMMS) {
    Serial.print(F("Hardware level: ")); Serial.println(v.HW_version);

    Serial.print(F("SHDLC protocol: ")); Serial.print(v.SHDLC_major);
    Serial.print("."); Serial.println(v.SHDLC_minor);
  }

  Serial.print(F("Library level : "));  Serial.print(v.DRV_major);
  Serial.print(".");  Serial.println(v.DRV_minor);
}

bool read_sps30()
{
  static bool header = true;
  uint8_t ret, error_cnt = 0;
  struct sps_values val;

  // loop to get data
  do {

    ret = sps30.GetValues(&val);
```

```
    // data might not have been ready
    if (ret == SPS30_ERR_DATALENGTH){

        if (error_cnt++ > 3) {
          ErrtoMess((char *) "Error during reading values: ",ret);
          return(false);
        }
        delay(1000);
    }

    // if other error
    else if(ret != SPS30_ERR_OK) {
      ErrtoMess((char *) "Error during reading values: ",ret);
      return(false);
    }

  } while (ret != SPS30_ERR_OK);

#ifdef DiTodos
  // only print header first time
  if (header) {
    Serial.println(F("-------------Mass -----------   ------------- Number
--------------   -Average-"));
    Serial.println(F("     Concentration [µg/m3]           Concentration
[#/cm3]             [µm]"));
    Serial.println(F("P1.0\tP2.5\tP4.0\tP10\tP0.5\tP1.0\tP2.5\tP4.0\tP10\tP
artSize\n"));
    header = false;
  }

  Serial.print(val.MassPM1);
  Serial.print(F("\t"));
  Serial.print(val.MassPM2);
  Serial.print(F("\t"));
  Serial.print(val.MassPM4);
  Serial.print(F("\t"));
  Serial.print(val.MassPM10);
  Serial.print(F("\t"));
  Serial.print(val.NumPM0);
  Serial.print(F("\t"));
  Serial.print(val.NumPM1);
  Serial.print(F("\t"));
  Serial.print(val.NumPM2);
  Serial.print(F("\t"));
  Serial.print(val.NumPM4);
  Serial.print(F("\t"));
  Serial.print(val.NumPM10);
```

```
  Serial.print(F("\t"));
  Serial.print(val.PartSize);
  Serial.print(F("\n"));
#endif

  //Multilinear regression
  val.MassPM1 = 1.0579*val.MassPM1 + 0.02534*temp + 0.011566*hum;
  val.MassPM2 = 1.0579*val.MassPM2 + 0.02534*temp + 0.011566*hum;
  val.MassPM4 = 1.0579*val.MassPM4 + 0.02534*temp + 0.011566*hum;
  val.MassPM10 = 1.0579*val.MassPM10 + 0.02534*temp + 0.011566*hum;


  pm25_sps=val.MassPM2;
  pm1_sps=val.MassPM1;
  pm10_sps=val.MassPM10;

  for (int i = 0; i < 3; i++)
  {
    dato_sps30.medmassPM1 = dato_sps30.medmassPM1 + val.MassPM1;
    dato_sps30.medmassPM2 = dato_sps30.medmassPM2 + val.MassPM2;
    dato_sps30.medmassPM4 = dato_sps30.medmassPM4 + val.MassPM4;
    dato_sps30.medmassPM10 = dato_sps30.medmassPM10 + val.MassPM10;
    dato_sps30.medPartSize = dato_sps30.medPartSize + val.PartSize;

    dato_sps30.mednumPM0 = dato_sps30.mednumPM0 + val.NumPM0;
    dato_sps30.mednumPM1 = dato_sps30.mednumPM1 + val.NumPM1;
    dato_sps30.mednumPM2 = dato_sps30.mednumPM2 + val.NumPM2;
    dato_sps30.mednumPM4 = dato_sps30.mednumPM4 + val.NumPM4;
    dato_sps30.mednumPM10 = dato_sps30.mednumPM10 + val.NumPM10;

    dato_sps30.medidas++;
  }

  return(true);
}

void Errorloop(char *mess, uint8_t r)
{
  if (r) ErrtoMess(mess, r);
  else Serial.println(mess);
  Serial.println(F("Program on hold"));
  for(;;) delay(100000);
}

void ErrtoMess(char *mess, uint8_t r)
{
  char buf[80];
  Serial.print(mess);
```

```
  sps30.GetErrDescription(r, buf, 80);
  Serial.println(buf);
}

void read_co2(){

  CO2 = myMHZ19.getCO2(); // Request CO2 (as ppm)
  if(CO2<400)
  {
    CO2=400;
  }
  else if(CO2>5000)
  {
    CO2=5000;
  }

  #ifdef DiTodos
  Serial.print("CO2 (ppm): ");
  Serial.println(CO2);
  #endif

  for (int i = 0; i < 3; i++)
    {
      dato_mhz19b.medco2 = dato_mhz19b.medco2 + CO2;
      dato_mhz19b.medidas++;
    }
}

void read_bme280()
{
  BMESensor.refresh(); // read current sensor data

  #ifdef DiTodos
    Serial.print("Temperature: ");
    Serial.print(BMESensor.temperature); // display temperature in Celsius
    Serial.println("C");

    Serial.print("Humidity:    ");
    Serial.print(BMESensor.humidity); // display humidity in %
    Serial.println("%");

    Serial.print("Pressure:    ");
    Serial.print(BMESensor.pressure  / 100.0F); // display pressure in hPa
    Serial.println("hPa");
  #endif

    temp=BMESensor.temperature;
    hum=BMESensor.humidity;
```

```
1799      press=BMESensor.pressure/100;
1800
1801      for (int i = 0; i < 3; i++)
1802      {
1803        dato_bme280.medhum = dato_bme280.medhum + hum;
1804        dato_bme280.medpress = dato_bme280.medpress + press;
1805        dato_bme280.medtemp = dato_bme280.medtemp + temp;
1806        dato_bme280.medidas++;
1807      }
1808  }
```

7.2. ANEXO 2 – CÓDIGO BAJO COSUMO

Código completo del programa para bajo consumo para el dispositivo AirKnowledge:

```
#include <Arduino.h>

//IOT - Thingspeak
#include "ThingSpeak.h" // always include thingspeak header file after
other header files and custom macros

//MH-Z19B
#include "MHZ19.h"
#include <SoftwareSerial.h>

//Wifi
#include <WiFiClientSecure.h>
#include <WiFi.h>
#include "time.h"

//SPS30 - Air quality sensor
#include "sps30.h"

//BME280
#include <Wire.h> // required by BME280 library
#include <BME280_t.h>

//SD
//#include "FS.h"
#include <SD.h>
//#include <SPI.h>

//RTClib - Pila
#include "RTClib.h"
//#include <Wire.h>
#include <SPI.h>

//BLYNK

// Template ID, Device Name and Auth Token are provided by the Blynk.Cloud
// See the Device Info tab, or Template settings
#define BLYNK_TEMPLATE_ID "********"
#define BLYNK_DEVICE_NAME "TFG"
#define BLYNK_AUTH_TOKEN "********"

// Comment this out to disable prints and save space
//#define BLYNK_PRINT Serial

```

```
//#include <WiFi.h>
//#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char auth[] = BLYNK_AUTH_TOKEN;

//MH-Z19B
#define FORCE_SPAN 0 // < --- set to 1 as an absoloute final resort
MHZ19 myMHZ19;
SoftwareSerial mySerial(25, 33);  //SoftwareSerial mySerial(RX_PIN,
TX_PIN);

//NTP
const char *ntpServer1 = "pool.ntp.org";
const char *ntpServer2 = "hora.roa.es";
const char *ntpServer3 = "europe.pool.ntp.org";
const long gmtOffset_sec = 7200;
const int daylightOffset_sec = 0;

//wifi
char ssid[20] = "********";            // Aqui va ti identificador
char password[40] = "********"; //Aqui va tu contraseña

//thingspeak
unsigned long myChannelNumber = 1853883;
const char * myWriteAPIKey = "********";

//SPS30
#define SP30_COMMS SERIALPORT2
#define USE_I2C   // I2C using library wire (+10k code, 0k2 mem, 124 iram)
#define USE_SPS30

#define TX_PIN 17
#define RX_PIN 16

#define DEBUG 0

//#define DiTodos 0

#define blynk_enable 1

//SD
const int chipSelect = 5;

//IOT - thingspeak
WiFiClient  client;

// create constructor
```

```
92  SPS30 sps30;
93
94  //Hora
95  DateTime now;
96
97  //SD
98  File myFile;
99
100 //BME280
101 BME280<> BMESensor;
102
103 //PILA
104 RTC_DS3231 rtc;
105
106 //ESTRUCTURAS
107 struct sps30datos
108 {
109
110   float medmassPM1;
111   float medmassPM2;
112   float medmassPM4;
113   float medmassPM10;
114   float medPartSize;
115
116   float mednumPM0;
117   float mednumPM1;
118   float mednumPM2;
119   float mednumPM4;
120   float mednumPM10;
121   float medidas;
122
123 } dato_sps30;
124
125 struct bme280datos
126 {
127   float medpress;
128   float medtemp;
129   float medhum;
130   float medidas;
131   String horainic;
132
133 } dato_bme280;
134
135 struct mhz19b_datos
136 {
137   float medco2;
138   float medidas;
139
```

```
} dato_mhz19b;

//datos globales pantalla
float pm1_sps,pm10_sps,pm25_sps,hum,temp,press;
int CO2=400, sps30notdetected=0;

void save_sd_todos();

void read_bme280();

void ErrtoMess(char *mess, uint8_t r);
void Errorloop(char *mess, uint8_t r);
void GetDeviceInfo();
bool read_sps30();

void mostrarmenu();
void mostrarendisplay();
void draw();
void drawLogo(void);

void printErrorCode();
void setRange(int range);
void read_co2();

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  while (!Serial){} //Wait for serial port to connect

  //MHZ19B
  mySerial.begin(9600); // Uno example: Begin Stream with MHZ19 baudrate
  myMHZ19.begin(mySerial); // *Imporant, Pass your Stream reference
  myMHZ19.autoCalibration(); // Turn auto calibration ON (OFF
autoCalibration(false))

  //RTC
  if (!rtc.begin())
  {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    abort();
  }
  else
  {
    Serial.println("El RTC funciona correctamente");
  }
  //FIN RTC

```

```
  //BME280
  BMESensor.begin();

  //SD
  Serial.println("Initializing SD card...");
  if (!SD.begin(chipSelect))
  {
    Serial.println("Initialization failed!");
    while (1);
  }
  Serial.println("initialization done.");

  //WIFI
  Serial.println("Conectando... ");
  Serial.println(ssid);

  WiFi.begin(ssid, password); // Intentamos la conexion a la WIFI
  delay(5000);

  if (WiFi.status() != WL_CONNECTED)
  {
    Serial.println("Aun no se ha conectado al wifi");
      delay(5000);
  }
  else
  {
    Serial.println("WiFi connected..!");
    Serial.println("");

    // Initialize ThingSpeak
    ThingSpeak.begin(client);
    #ifdef blynk_enable
      // Initialize Blynk
      Blynk.connectWiFi(ssid, password);
      Blynk.config(auth, BLYNK_DEFAULT_DOMAIN, BLYNK_DEFAULT_PORT);
      if(Blynk.connect() != true){delay(1000);}
    #endif
  } //FIN WIFI

  ////////////////////////////////////////////////////
  //init and get the time
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer1, ntpServer2,
ntpServer3);
  ////////////////////////////////////////////////////

  //RTC
  if (rtc.lostPower()) {
    //Si se ha ido la luz no entra
```

```
    rtc.adjust(DateTime(2022, 1, 1, 0, 0, 0));
  }

//SPS30
  Serial.println(F("Trying to connect to sps30"));
  // set driver debug level
  sps30.EnableDebugging(DEBUG);

  // set pins to use for softserial and Serial1 on ESP32
  if (TX_PIN != 0 && RX_PIN != 0) sps30.SetSerialPin(RX_PIN,TX_PIN);

  // Begin communication channel;
  if (! sps30.begin(SP30_COMMS))
    Errorloop((char *) "could not initialize communication channel.", 0);

  // check for SPS30 connection
  if (! sps30.probe()) {
      //Errorloop((char *) "could not probe / connect with SPS30.", 0);
      Serial.print("could not probe / connect with SPS30.");
      sps30notdetected=1;
  }
  else  Serial.println(F("Detected SPS30."));

  // reset SPS30 connection
  if (! sps30.reset()) Errorloop((char *) "could not reset.", 0);

  // read device info
  if(!sps30notdetected) {

    GetDeviceInfo();

    // start measurement
    if (sps30.start()) Serial.println(F("Measurement started"));
    else Errorloop((char *) "Could NOT start measurement", 0);

    delay(300);

    if (SP30_COMMS == I2C_COMMS) {
      if (sps30.I2C_expect() == 4)
        Serial.println(F(" !!! Due to I2C buffersize only the SPS30 MASS
concentration is available !!! \n"));
    }
  }

  //En microsegundos el tiempo que va estar dormido
  esp_sleep_enable_timer_wakeup(10*60*1000000); //Cada 10 min se despierta
el microcontrolador
```

```
284   Serial.println("Me acabo de despertar");
285
286   if ((WiFi.status() != WL_CONNECTED))
287   {
288     delay(200);
289     WiFi.begin(ssid, password); // Intentamos la conexion a la WIFI
290     delay(5000);
291   }
292
293   if ((WiFi.status() == WL_CONNECTED)){
294       // Initialize ThingSpeak
295       ThingSpeak.begin(client);
296
297       #ifdef blynk_enable
298         // Initialize Blynk
299         Blynk.connectWiFi(ssid, password);
300         Blynk.config(auth, BLYNK_DEFAULT_DOMAIN, BLYNK_DEFAULT_PORT);
301         if(Blynk.connect() != true){delay(1000);}
302       #endif
303
304       time_t timeepoch;
305       struct tm timeinfo;
306
307       Serial.println("Estoy conectado al wifi");
308       if (!getLocalTime(&timeinfo))
309       {
310         Serial.println("Failed to obtain time");
311       }
312       else
313       {
314         time(&timeepoch);
315         timeepoch = timeepoch + 7200; // GTM + 1
316         rtc.adjust(timeepoch);
317         delay(100);
318         now = rtc.now();
319       }
320   }
321
322   //Leemos el sensor sps30
323   read_sps30();
324
325   //Leemos el sensor bme280
326   read_bme280();
327
328   //Leemos el sensor mhz19b
329   read_co2();
330
331   //Subimos los datos a internet
```

```
  if((WiFi.status() == WL_CONNECTED)){
    #ifdef blynk_enable
      Blynk.run();
    #endif

    ThingSpeak.setField(1, hum);
    ThingSpeak.setField(2, temp);
    ThingSpeak.setField(3, press);
    ThingSpeak.setField(4, CO2);
    ThingSpeak.setField(5, pm1_sps);
    ThingSpeak.setField(6, pm25_sps);
    ThingSpeak.setField(7, pm10_sps);

    #ifdef blynk_enable
      Blynk.virtualWrite(V1, hum);
      Blynk.virtualWrite(V2, temp);
      Blynk.virtualWrite(V3, press);
      Blynk.virtualWrite(V4, CO2);
      Blynk.virtualWrite(V5, pm1_sps);
      Blynk.virtualWrite(V6, pm25_sps);
      Blynk.virtualWrite(V7, pm10_sps);
    #endif

    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    Serial.println("Datos subidos a thingiverse y blynk");
  }

  //Guardamos los datos en la microSD
  save_sd_todos();

  Serial.println("Me voy a dormir");

  Serial.flush();
  esp_deep_sleep_start(); //A dormir
}

void loop() {
  //Nunca llegaremos aquí
}

//SAVING DATA
void save_sd_todos()
{

  static bool header = true;

  dato_sps30.medmassPM1 = dato_sps30.medmassPM1 / dato_sps30.medidas;
  dato_sps30.medmassPM2 = dato_sps30.medmassPM2 / dato_sps30.medidas;
```

```
  dato_sps30.medmassPM4 = dato_sps30.medmassPM4 / dato_sps30.medidas;
  dato_sps30.medmassPM10 = dato_sps30.medmassPM10 / dato_sps30.medidas;
  dato_sps30.medPartSize = dato_sps30.medPartSize / dato_sps30.medidas;

  dato_sps30.mednumPM0 = dato_sps30.mednumPM0 / dato_sps30.medidas;
  dato_sps30.mednumPM1 = dato_sps30.mednumPM1 / dato_sps30.medidas;
  dato_sps30.mednumPM2 = dato_sps30.mednumPM2 / dato_sps30.medidas;
  dato_sps30.mednumPM4 = dato_sps30.mednumPM4 / dato_sps30.medidas;
  dato_sps30.mednumPM10 = dato_sps30.mednumPM10 / dato_sps30.medidas;

  dato_bme280.medhum = dato_bme280.medhum / dato_bme280.medidas;
  dato_bme280.medtemp = dato_bme280.medtemp / dato_bme280.medidas;
  dato_bme280.medpress = dato_bme280.medpress / dato_bme280.medidas;

  dato_mhz19b.medco2 = dato_mhz19b.medco2 / dato_mhz19b.medidas;

  if (header)
  {
      char date_string[32] = "DD_MM_YY-hh_mm_ss-";
      dato_bme280.horainic = "/" + (String)now.toString(date_string);
      dato_bme280.horainic = dato_bme280.horainic + "datafile.csv";
      myFile = SD.open(dato_bme280.horainic, FILE_WRITE);
  }
  else
  {
    //Para no sobre escribir
    myFile = SD.open(dato_bme280.horainic, FILE_APPEND);
  }

  if (myFile)
  {
    Serial.print("opened datafile.csv...");
    if (header)
    {
      ///////////HORA//////////////
      myFile.print("Date");
      myFile.print(",");
      myFile.print("Time");
      myFile.print(",");
      ///////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////SPS30_1//////////////
      myFile.print("Mass");
      myFile.print(",");
      myFile.print("Mass");
      myFile.print(",");
      myFile.print("Mass");
```

```
428       myFile.print(",");
429       myFile.print("Mass");
430       myFile.print(",");
431       myFile.print("Number");
432       myFile.print(",");
433       myFile.print("Number");
434       myFile.print(",");
435       myFile.print("Number");
436       myFile.print(",");
437       myFile.print("Number");
438       myFile.print(",");
439       myFile.print("Number");
440       myFile.print(",");
441       myFile.print("Average");
442       myFile.print(",");
443       ///////////////SALTO////////////////
444       myFile.print("      ");
445       myFile.print(",");
446       //////////DHT11//////////////
447       myFile.print("Temperature");
448       myFile.print(",");
449       myFile.print("Humidity");
450       myFile.print(",");
451       myFile.print("Presion");
452       myFile.print(",");
453       ///////////////SALTO////////////////
454       myFile.print("      ");
455       myFile.print(",");
456       ////////////////MHZ19B/////////////////
457       myFile.println("CO2");
458       ///////////FILA 2//////////////
459       myFile.print("[DD/MM/AAAA]");
460       myFile.print(",");
461       myFile.print("[HH/mm/ss]");
462       myFile.print(",");
463       ///////////////SALTO////////////////
464       myFile.print("      ");
465       myFile.print(",");
466       //////////SPS30_2//////////////
467       myFile.print("[microgramo/m3]");
468       myFile.print(",");
469       myFile.print("[microgramo/m3]");
470       myFile.print(",");
471       myFile.print("[microgramo/m3]");
472       myFile.print(",");
473       myFile.print("[microgramo/m3]");
474       myFile.print(",");
475       myFile.print("[#/m3]");
```

```
476       myFile.print(",");
477       myFile.print("[#/m3]");
478       myFile.print(",");
479       myFile.print("[#/m3]");
480       myFile.print(",");
481       myFile.print("[#/m3]");
482       myFile.print(",");
483       myFile.print("[#/m3]");
484       myFile.print(",");
485       myFile.print("[microgramo]");
486       myFile.print(",");
487       ////////////////SALTO/////////////////
488       myFile.print("      ");
489       myFile.print(",");
490       ///////////BME280///////////////
491       myFile.print(F("[ºC]"));
492       myFile.print(",");
493       myFile.print("[%]");
494       myFile.print(",");
495       myFile.print("[hPa]");
496       myFile.print(",");
497       ////////////////SALTO/////////////////
498       myFile.print("      ");
499       myFile.print(",");
500       ////////////////MHZ19B/////////////////
501       myFile.println("ppm");
502       ///////////FILA 3////////////
503       myFile.print("");
504       myFile.print(",");
505       myFile.print("");
506       myFile.print(",");
507       ////////////////SALTO/////////////////
508       myFile.print("      ");
509       myFile.print(",");
510       ///////////SPS30_2///////////////
511       myFile.print("P1.0");
512       myFile.print(",");
513       myFile.print("P2.5");
514       myFile.print(",");
515       myFile.print("P4.0");
516       myFile.print(",");
517       myFile.print("P10");
518       myFile.print(",");
519       myFile.print("P0.5");
520       myFile.print(",");
521       myFile.print("P1.0");
522       myFile.print(",");
523       myFile.print("P2.5");
```

```
      myFile.print(",");
      myFile.print("P4.0");
      myFile.print(",");
      myFile.print("P10");
      myFile.print(",");
      myFile.print("PartSize");
      myFile.print(",");
      ////////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ///////////BME280///////////////
      myFile.print(" ");
      myFile.print(",");
      myFile.print(" ");
      myFile.print(",");
      myFile.print(" ");
      myFile.print(",");
      ////////////////SALTO/////////////////
      myFile.print("      ");
      myFile.print(",");
      ////////////////////////////////
      myFile.println(" ");

      header = false;
    }

    char date_string[32] = "DD/MM/YY";
    String fecha = (String)now.toString(date_string);

    char hour_string[32] = "hh:mm:ss";
    String hora = (String)now.toString(hour_string);

    myFile.print(fecha);
    myFile.print(",");
    myFile.print(hora);
    myFile.print(",");
    ////////////////SALTO/////////////////
    myFile.print("      ");
    myFile.print(",");
    ///////////SPS30_2///////////////
    myFile.print(dato_sps30.medmassPM1);
    myFile.print(",");
    myFile.print(dato_sps30.medmassPM2);
    myFile.print(",");
    myFile.print(dato_sps30.medmassPM4);
    myFile.print(",");
    myFile.print(dato_sps30.medmassPM10);
    myFile.print(",");
```

```
    myFile.print(dato_sps30.mednumPM0);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM1);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM2);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM4);
    myFile.print(",");
    myFile.print(dato_sps30.mednumPM10);
    myFile.print(",");
    myFile.print(dato_sps30.medPartSize);
    myFile.print(",");
    ////////////////SALTO/////////////////
    myFile.print("      ");
    myFile.print(",");
    ///////////BME280///////////////
    myFile.print(dato_bme280.medtemp);
    myFile.print(",");
    myFile.print(dato_bme280.medhum);
    myFile.print(",");
    myFile.print(dato_bme280.medpress);
    myFile.print(",");
    ////////////////SALTO/////////////////
    myFile.print("      ");
    myFile.print(",");
    ////////////////MHZ19B/////////////////
    myFile.println(dato_mhz19b.medco2);

    // close the file:
    myFile.close();
    Serial.println("closed datafile.csv.");
  }
  else
  {
    // if the file didn't open, print an error:
    Serial.println("error opening datafile.csv");
  }

  //Ponemos los valores de las medias a 0
  dato_sps30.medmassPM1 = 0;
  dato_sps30.medmassPM2 = 0;
  dato_sps30.medmassPM4 = 0;
  dato_sps30.medmassPM10 = 0;
  dato_sps30.medPartSize = 0;

  dato_sps30.mednumPM0 = 0;
  dato_sps30.mednumPM1 = 0;
  dato_sps30.mednumPM2 = 0;
```

```
  dato_sps30.mednumPM4 = 0;
  dato_sps30.mednumPM10 = 0;
  dato_sps30.medidas = 0;

  dato_bme280.medhum = 0;
  dato_bme280.medidas = 0;
  dato_bme280.medtemp = 0;
  dato_bme280.medpress = 0;

  dato_mhz19b.medco2 = 0;
  dato_mhz19b.medidas = 0;

}

void GetDeviceInfo()
{
  char buf[32];
  uint8_t ret;
  SPS30_version v;

  //try to read serial number
  ret = sps30.GetSerialNumber(buf, 32);
  if (ret == SPS30_ERR_OK) {
    Serial.print(F("Serial number : "));
    if(strlen(buf) > 0)  Serial.println(buf);
    else Serial.println(F("not available"));
  }
  else
    ErrtoMess((char *) "could not get serial number", ret);

  // try to get product name
  ret = sps30.GetProductName(buf, 32);
  if (ret == SPS30_ERR_OK)  {
    Serial.print(F("Product name  : "));

    if(strlen(buf) > 0)  Serial.println(buf);
    else Serial.println(F("not available"));
  }
  else
    ErrtoMess((char *) "could not get product name.", ret);

  // try to get version info
  ret = sps30.GetVersion(&v);
  if (ret != SPS30_ERR_OK) {
    Serial.println(F("Can not read version info"));
    return;
  }

```

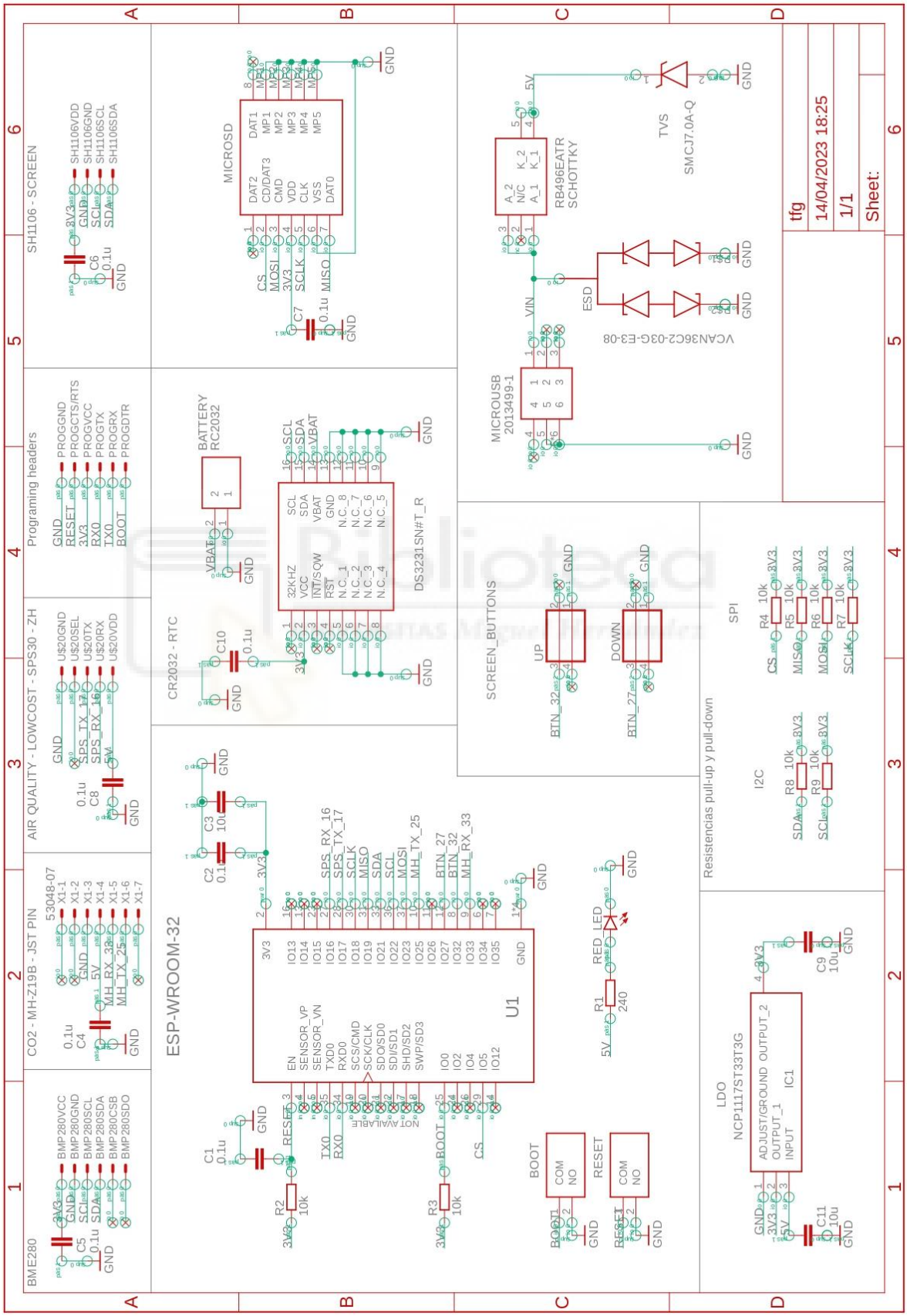
```
  Serial.print(F("Firmware level: "));  Serial.print(v.major);
  Serial.print("."); Serial.println(v.minor);

  if (SP30_COMMS != I2C_COMMS) {
    Serial.print(F("Hardware level: ")); Serial.println(v.HW_version);

    Serial.print(F("SHDLC protocol: ")); Serial.print(v.SHDLC_major);
    Serial.print("."); Serial.println(v.SHDLC_minor);
  }

  Serial.print(F("Library level : "));  Serial.print(v.DRV_major);
  Serial.print(".");  Serial.println(v.DRV_minor);
}

bool read_sps30()
{
  static bool header = true;
  uint8_t ret, error_cnt = 0;
  struct sps_values val;

  // loop to get data
  do {

    ret = sps30.GetValues(&val);

    // data might not have been ready
    if (ret == SPS30_ERR_DATALENGTH){

        if (error_cnt++ > 3) {
          ErrtoMess((char *) "Error during reading values: ",ret);
          return(false);
        }
        delay(1000);
    }

    // if other error
    else if(ret != SPS30_ERR_OK) {
      ErrtoMess((char *) "Error during reading values: ",ret);
      return(false);
    }

  } while (ret != SPS30_ERR_OK);

#ifdef DiTodos
  // only print header first time
  if (header) {
    Serial.println(F("-------------Mass -----------    ------------- Number
--------------   -Average-"));
```

```
    Serial.println(F("     Concentration [µg/m3]             Concentration
[#/cm3]             [µm]"));
    Serial.println(F("P1.0\tP2.5\tP4.0\tP10\tP0.5\tP1.0\tP2.5\tP4.0\tP10\tP
artSize\n"));
    header = false;
  }

  Serial.print(val.MassPM1);
  Serial.print(F("\t"));
  Serial.print(val.MassPM2);
  Serial.print(F("\t"));
  Serial.print(val.MassPM4);
  Serial.print(F("\t"));
  Serial.print(val.MassPM10);
  Serial.print(F("\t"));
  Serial.print(val.NumPM0);
  Serial.print(F("\t"));
  Serial.print(val.NumPM1);
  Serial.print(F("\t"));
  Serial.print(val.NumPM2);
  Serial.print(F("\t"));
  Serial.print(val.NumPM4);
  Serial.print(F("\t"));
  Serial.print(val.NumPM10);
  Serial.print(F("\t"));
  Serial.print(val.PartSize);
  Serial.print(F("\n"));
#endif

  //Multilinear regression
  val.MassPM1 = 1.0579*val.MassPM1 + 0.02534*temp + 0.011566*hum;
  val.MassPM2 = 1.0579*val.MassPM2 + 0.02534*temp + 0.011566*hum;
  val.MassPM4 = 1.0579*val.MassPM4 + 0.02534*temp + 0.011566*hum;
  val.MassPM10 = 1.0579*val.MassPM10 + 0.02534*temp + 0.011566*hum;


  pm25_sps=val.MassPM2;
  pm1_sps=val.MassPM1;
  pm10_sps=val.MassPM10;

  for (int i = 0; i < 3; i++)
  {

    dato_sps30.medmassPM1 = dato_sps30.medmassPM1 + val.MassPM1;
    dato_sps30.medmassPM2 = dato_sps30.medmassPM2 + val.MassPM2;
    dato_sps30.medmassPM4 = dato_sps30.medmassPM4 + val.MassPM4;
    dato_sps30.medmassPM10 = dato_sps30.medmassPM10 + val.MassPM10;
    dato_sps30.medPartSize = dato_sps30.medPartSize + val.PartSize;
```

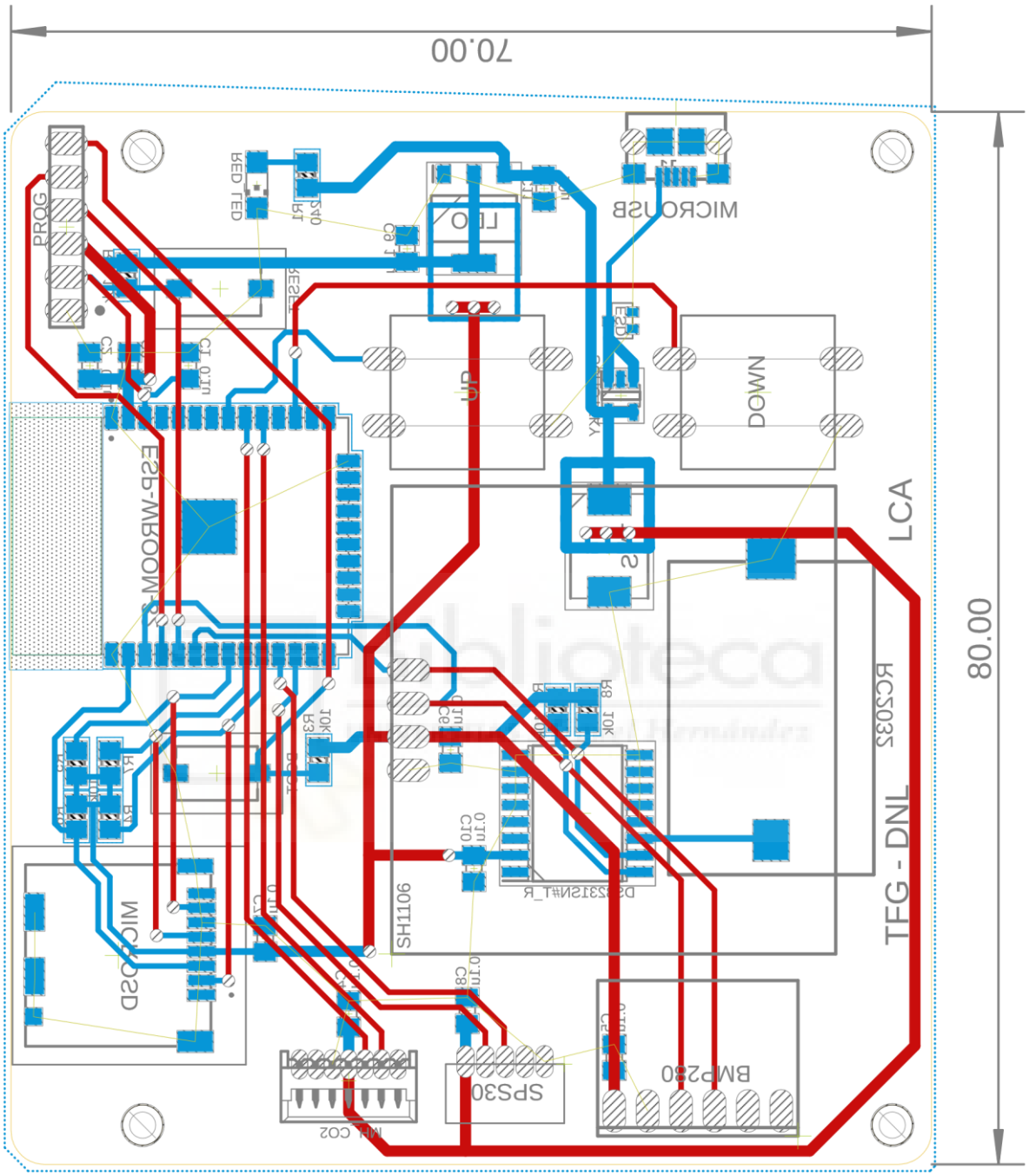
```
764
765       dato_sps30.mednumPM0 = dato_sps30.mednumPM0 + val.NumPM0;
766       dato_sps30.mednumPM1 = dato_sps30.mednumPM1 + val.NumPM1;
767       dato_sps30.mednumPM2 = dato_sps30.mednumPM2 + val.NumPM2;
768       dato_sps30.mednumPM4 = dato_sps30.mednumPM4 + val.NumPM4;
769       dato_sps30.mednumPM10 = dato_sps30.mednumPM10 + val.NumPM10;
770
771       dato_sps30.medidas++;
772     }
773
774     return(true);
775   }
776
777   void Errorloop(char *mess, uint8_t r)
778   {
779     if (r) ErrtoMess(mess, r);
780     else Serial.println(mess);
781     Serial.println(F("Program on hold"));
782     for(;;) delay(100000);
783   }
784
785   void ErrtoMess(char *mess, uint8_t r)
786   {
787     char buf[80];
788
789     Serial.print(mess);
790
791     sps30.GetErrDescription(r, buf, 80);
792     Serial.println(buf);
793   }
794
795   void read_co2(){
796
797     CO2 = myMHZ19.getCO2(); // Request CO2 (as ppm)
798     if(CO2<400)
799     {
800       CO2=400;
801     }
802     else if(CO2>10000)
803     {
804       CO2=10000;
805     }
806
807     #ifdef DiTodos
808     Serial.print("CO2 (ppm): ");
809     Serial.println(CO2);
810     #endif
811
```

```
812   for (int i = 0; i < 3; i++)
813     {
814       dato_mhz19b.medco2 = dato_mhz19b.medco2 + CO2;
815       dato_mhz19b.medidas++;
816     }
817 }
818 
819 void read_bme280()
820 {
821   BMESensor.refresh(); // read current sensor data
822 
823   #ifdef DiTodos
824     Serial.print("Temperature: ");
825     Serial.print(BMESensor.temperature); // display temperature in Celsius
826     Serial.println("C");
827 
828     Serial.print("Humidity:    ");
829     Serial.print(BMESensor.humidity); // display humidity in %
830     Serial.println("%");
831 
832     Serial.print("Pressure:    ");
833     Serial.print(BMESensor.pressure  / 100.0F); // display pressure in hPa
834     Serial.println("hPa");
835   #endif
836 
837     temp=BMESensor.temperature;
838     hum=BMESensor.humidity;
839     press=BMESensor.pressure/100;
840 
841     for (int i = 0; i < 3; i++)
842     {
843       dato_bme280.medhum = dato_bme280.medhum + hum;
844       dato_bme280.medpress = dato_bme280.medpress + press;
845       dato_bme280.medtemp = dato_bme280.medtemp + temp;
846       dato_bme280.medidas++;
847     }
848 }
```

7.3. ANEXO 3 – ESQUEMÁTICO



7.4. ANEXO 4 - DISEÑO DE LA PCB



7.5. ANEXO 5 – LISTA DE COMPONENTES

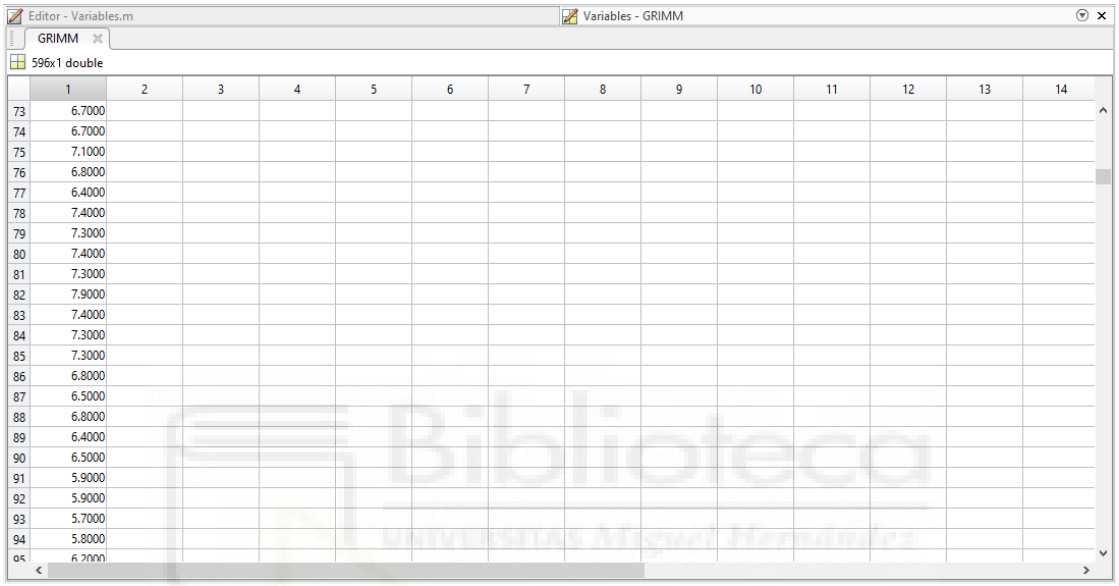
Listado de componentes o BOM, de las siglas en inglés de *bill of materials*, generado a través de Eagle:

Qty	Parts	Value	Device	Package	Description
1	R1	240	R-EU_R0805	R0805	Resistencia 240 Ohm - 0805 - BOURNS
8	R2	10k	R-EU_R0805	R0805	Resistencia 10k Ohm - 0805 - BOURNS
8	C1	0.1u	C-EUC0805K	C0805K	Condensador 0805 0.1u 50V - KEMET
3	C3	10u	C-EUC0805K	C0805K	Condensador 0805 10u 50V - KEMET
2	BOTON_THT		BOTON_THT	BOTON_THT	Botón de 4 pines, THT
2	BUTTON_SMD		BUTTON_SMD	BUTTON_SMD	Botón de 2 pines, SMD
1	PANTALLA		SH1106	PIN_4	Pantalla - SH1106 - 1,3" - Oled monocolor - I2C
1	DS3231SN		DS3231SN#T_R	SO16	RTC - DS3231SN#T&R -MAXIM INTEGRATED PRODUCTS
1	BATTERY		RC2032	S8421-45R_1	Soporte para pila tipo RC2032
1	MICROSD		DM3CS-SF	DM3CSSF	Conector para memoria microSD
1	RED_LED		BR1101W-TR	LED_1206	Led estandar SMD, color rojo, empaquetado 1206
1	PROG_PINS		PROGRAMING	PIN_6	6 pines tipo THT con 2.54mm de separación
1	U1		ESP-WROOM-32	ESP-WROOM-32	Microcontrolador de Espressif, con modulo bluetooth y wifi
1	IC1		NCP1117ST33T3G	SOT-223-3	LDO - NCP1117ST33T3G - 5V a 3.3V
1	TVS		SMCJ7.0A-Q	DIOM7959X262N	TVS DIODE - 7V 1500W - AECQ101
1	SCHOTTKY		RB496EATR	SOT95P280X100-5N	ROHM SCHOTTKY DIODE - Vr=20V, Vf=0.4V y 1A
1	ESD		VCAN36C2-03G-E3-08	VCAN36C2-03G	TVS bidireccional tipo ESD
1	MICROUSB		2013499-1	2013499-1	Conector microUSB - TE CONNECTIVITY
1	SPS30		SPS30	ZH - 5 pines	Sensor de particulas de la empresa Sensirion
1	BME280		BME280	PIN_6	Sensor meteorológico de la empresa Bosch
1	X1		MHZ-19B	JST - 7 pines	Sensor de CO2 de la empresa Winsen

7.6. ANEXO 6 – REGRESIÓN MULTILINEAL

En este anexo se va a determinar los pasos para obtener la ecuación de dependencia de la medida leída por el sensor de PM con factores meteorológicos utilizando la aplicación llamada Matlab.

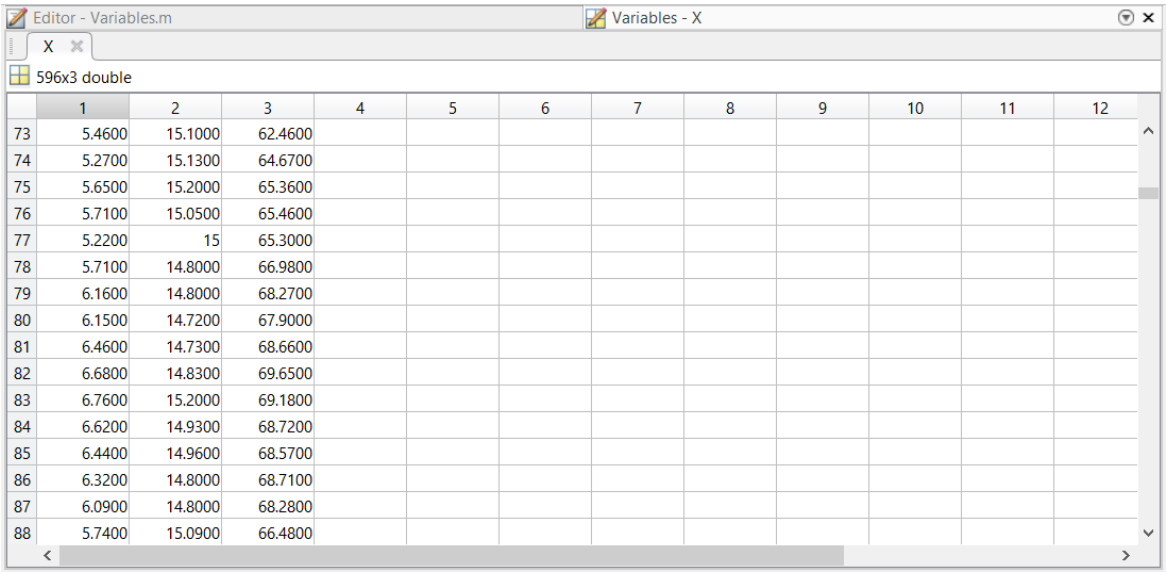
En primer lugar, se introduce el vector de valores del sensor de referencia (Fig. 111).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
73	6.7000													
74	6.7000													
75	7.1000													
76	6.8000													
77	6.4000													
78	7.4000													
79	7.3000													
80	7.4000													
81	7.3000													
82	7.9000													
83	7.4000													
84	7.3000													
85	7.3000													
86	6.8000													
87	6.5000													
88	6.8000													
89	6.4000													
90	6.5000													
91	5.9000													
92	5.9000													
93	5.7000													
94	5.8000													
95	6.2000													

Figura 111 – Vector de valores – Sensor referencia

En segundo lugar, se introduce la matriz de valores medidos por el sensor de bajo coste, humedad y temperatura (Fig. 112).



	1	2	3	4	5	6	7	8	9	10	11	12
73	5.4600	15.1000	62.4600									
74	5.2700	15.1300	64.6700									
75	5.6500	15.2000	65.3600									
76	5.7100	15.0500	65.4600									
77	5.2200	15	65.3000									
78	5.7100	14.8000	66.9800									
79	6.1600	14.8000	68.2700									
80	6.1500	14.7200	67.9000									
81	6.4600	14.7300	68.6600									
82	6.6800	14.8300	69.6500									
83	6.7600	15.2000	69.1800									
84	6.6200	14.9300	68.7200									
85	6.4400	14.9600	68.5700									
86	6.3200	14.8000	68.7100									
87	6.0900	14.8000	68.2800									
88	5.7400	15.0900	66.4800									

Figura 112 – Matriz de valores de X=[SPS30, Temperatura, Humedad]

En tercer y último lugar, se utiliza la función **fitlm** para calcular la ecuación de la regresión multilineal (Fig. 113).

```
>> model = fitlm (X, GRIMM,'Intercept',false)

model =


Linear regression model:
    y ~ x1 + x2 + x3

Estimated Coefficients:
           Estimate       SE         tStat       pValue
           ________    _________    ______    ___________

    x1       1.0579     0.023741    44.562    1.9881e-191
    x2      0.02534    0.0037295    6.7944     2.6533e-11
    x3     0.011566    0.0024152     4.789       2.12e-06
```

Figura 113 – Cálculo de la ecuación con fitlm - Matlab

Se puede extraer de la última figura la ecuación de la regresión lineal.

8. BIBLIOGRAFÍA

[1] Fine particles. Minnesota Pollution Control Agency. Recuperado 20 de enero de 2023, de https://www.pca.state.mn.us/pollutants-and-contaminants/fine-particles

[2] WHO global air quality guidelines. (2021). World Health Organization (WHO). https://apps.who.int/iris/bitstream/handle/10665/345329/9789240034228-eng.pdf

[3] MINISTERIO DE SANIDAD, SERVICIOS SOCIALES E IGUALDAD. (2013). Impactos del Cambio Climático en la Salud. Recuperado 8 de marzo de 2023, de https://www.sanidad.gob.es/ciudadanos/saludAmbLaboral/docs/CCCompleto.pdf

[4] Global Monitoring Laboratory - Carbon Cycle Greenhouse Gases. Recuperado 21 de mayo de 2023, de https://gml.noaa.gov/ccgg/trends/

[5] Consejería de agricultura - Generalitat Valenciana. Recuperado 10 de junio de 2023, de https://webcat-web.gva.es/webcat_web/datosOnlineRvvcca/recargarDatosOnline?codProv=3&codMuni=65&descMuni=Elx&codEst=234#sectionPestanyasEstacion

[6] Serinus® 31 - Acoem. (2023, 5 junio). Acoem. Recuperado 11 de junio de 2023, de https://www.acoem.com/en/products/air-quality-monitoring/air-quality-gas-monitors/co2-analyser/serinus-31/

[7] The World Smallest Powerful Portable Aerosol Spectrometer. Grimm 1109. ecoanalytics. Recuperado 12 de junio de 2023, de https://www.ecoanalytics.ch/admin/userfiles/produktedownloads/194/GRIMM_1.109%20E.pdf

[8] PurpleAir, Inc. PurpleAir. Real-time Air Quality Monitoring. Recuperado 12 de marzo de 2023, de https://www2.purpleair.com/

[9] AQMesh. AQMesh. Recuperado 23 de marzo de 12d. C., de https://www.aqmesh.com/

[10] How does Atmotube PM sensor work? Recuperado 16 de marzo de 2023, de https://atmotube.com/atmotube-support/how-does-atmotube-pm-sensor-work

[11] What is NDIR? Asahi Kasei Microdevices (AKM). Recuperado 20 de mayo de 2023, de https://www.akm.com/us/en/products/co2-sensor/tutorial/what-is-ndir/

[12] JL, B. (2023, abril 19). ¿Qué es un Termistor y Cómo Funciona? Electrónica Online. Recuperado 21 de mayo de 2023, de https://electronicaonline.net/componentes-electronicos/resistor/termistor/

[13] Condensador variable. laplace.us.es. Recuperado 21 de mayo de 2023, de https://n9.cl/4kbgq

[14] Barometric Pressure Sensor Basics. Pressure Sensor. Murata Manufacturing Co., Ltd. Murata Manufacturing Co., Ltd. https://www.murata.com/en-global/products/sensor/pressure/overview/basic

[15] Padial, J. (2020, 12 enero). Transferencia en paralelo y serie. Recuperado 20 de enero de 2023, de https://curiosoando.com/cual-es-la-diferencia-entre-un-puerto-serie-y-un-puerto-paralelo

[16] Campbell, S. (2021, 14 noviembre). Basics of the I2C Communication Protocol. Circuit Basics. Recuperado 25 de marzo de 2023, de https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[17] I2C Primer: What is I2C? Recuperado 23 de marzo de 2023, de https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html

[18] Peña, E., & Grace Legaspi, M. UART. Analog Dialogue. Recuperado 25 de marzo de 2023, de https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[19] Serial Communication From the Command Line – Onion. Recuperado 25 de marzo de 2023, de https://onion.io/2bt-serial-communication-from-the-command-line/

[20] Campbell, S. (2021, noviembre 14). Basics of the SPI Communication Protocol. Circuit Basics. Recuperado 25 de marzo de 2023, de https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/

[21] Alfreedom, V. T. L. E. de. (2018, 15 abril). Protocolo de comunicación SPI. Vida Embebida. Recuperado 20 de enero de 2023, de https://vidaembebida.wordpress.com/2017/02/08/protocolo-de-comunicacion-spi/

[22] SD Pull-up Requirements for ESP32. ESPRESSIF DOCS. Recuperado 8 de abril de 2023, de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/sd_pullup_requirements.html

[23] Ramirez, F. (2018, 27 abril). Resistencia Pull Up y Pull Down. TuElectronica.es. Recuperado 20 de enero de 2023, de https://tuelectronica.es/resistencia-pull-up-y-pull-down/

[24] L. Llamas (2019, abril 17). ¿Qué es MQTT? Su importancia como protocolo IoT. Luis Llamas. Recuperado 25 de marzo de 2023, de https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

[25] Protocolo MQTT. Recuperado 25 de marzo de 2023, de https://arduinoblocks.blogspot.com/2019/02/protocolo-mqtt.html

[26] Llamas, L. (2020, abril 21). Ventajas y y desventajas de componentes PTH frente a SMD. Luis Llamas. Recuperado 15 de abril de 2023, de https://www.luisllamas.es/ventajas-y-y-desventajas-de-componentes-pth-frente-a-smd/

[27] A. Thomas, “Gantt Project”. Recuperado 20 de enero de 2023, de https://www.ganttproject.biz/

[28] Giordano, M. R., Malings, C., Pandis, S. N., Presto, A. A., McNeill, V. F., Westervelt, D. M., Beekmann, M. and Subramanian, R. (2021) From low-cost

sensors to high-quality data: A summary of challenges and best practices for effectively calibrating low-cost particulate matter mass sensors. Journal of Aerosol Science. 158, 105833.

[29] Datasheet SPS30. Recuperado 31 de marzo de 2023, de https://cdn.sparkfun.com/assets/2/d/2/a/6/Sensirion_SPS30_Particulate_Matter_Sensor_v0.9_D1__1_.pdf

[30] La biblia del sensor de CO2 MH-Z19B. (2022, 3 febrero). eMariete. Recuperado 20 de enero de 2023, de https://emariete.com/sensor-co2-mh-z19b/

[31] Datasheet MH-Z19B. Recuperado 1 de abril de 2023, de https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf

[32] Datasheet BME280. Recuperado 31 de marzo de 2023, de https://www.mouser.com/datasheet/2/783/BST-BME280-DS002-1509607.pdf

[33] SSH1106. Recuperado 20 de enero de 2023, de https://www.tiendatec.es/3899-large_default/pantalla-oled-130-128x64-azul-i2c-iic-sdd1306-arduino.jpg

[34] Botón. HETPRO. Recuperado 20 de enero de 2023, de https://hetpro-store.com/boton-con-tapa-blanco/

[35] MH-Z19B. Amazon. Recuperado 20 de enero de 2023, de https://www.amazon.es/ICQUANZX-infrarrojo-di%C3%B3xido-Carbono-0-5000ppm/dp/B07VD15YRP

[36] Sensirion, SPS30. RS. Recuperado 20 de enero de 2023, de https://es.rs-online.com/web/p/circuitos-integrados-de-sensores-ambientales/1862298

[37] tiendatec.es. (2022, 29 enero). MODULO BME280. Recuperado 20 de enero de 2023, de https://www.tiendatec.es/maker-zone/modulos/1740-modulo-bme280-sensor-ambiental-8472496021586.html

[38] Aguirre, R. (2020, 22 agosto). Protoboard. Blog Arduino, LabVIEW y Electrónica. Recuperado 20 de enero de 2023, de https://electronicamade.com/protoboard-placa-de-prueba/

[39] Cables. Amazon. Recuperado 20 de enero de 2023, de https://www.amazon.es/AZDelivery-Jumper-Arduino-Raspberry-Breadboard/dp/B07KKJ69DV/ref=sr_1_18?keywords=cable+arduino

[40] Placa de Desarrollo ESP32. Amazon. Recuperado 20 de enero de 2023, de https://www.amazon.es/Desarrollo-Wireless-Bluetooth-Potencia-Frecuencia/dp/B07YPQW5WQ

[41] Iván (2022, 3 abril). MicroSD. Tarjetasdememoria.info. Recuperado 20 de enero de 2023, de https://www.tarjetasdememoria.info/micro-sd-para-arduino/

[42] Módulo RTC. (2023, 9 enero). Leantec.ES. Recuperado 20 de enero de 2023, de https://leantec.es/tienda/ds3231-modulo-rtc-reloj-tiempo-real-avr-arm-arduino-raspberry-pi/

[43] Módulo ESP-WROOM-32 ESP32 WiFi. Naylamp Mechatronics. Recuperado 20 de enero de 2023, de https://naylampmechatronics.com/espressif-esp/382-modulo-esp-wroom-32-esp32-wifi.html

[44] Conector micro USB. RS. Recuperado 20 de enero de 2023, de https://es.rs-online.com/web/p/conectores-usb/1864887

[45] LDO NCP1117ST33T3G. Mouser. Recuperado 20 de enero de 2023, de https://www.mouser.es/ProductDetail/onsemi/NCP1117ST33T3G?qs=Gev%2BmEvV0iZb/E8ahUDx3w==

[46] TVS SMCJ7.0A-Q. Mouser. Recuperado 20 de enero de 2023, de https://www.mouser.es/ProductDetail/Bourns/SMCJ7.0A-Q?qs=zW32dvEIR3va6dDh0nJ%2BPQ==

[47] ESD - VCAN36C2-03G-E3-08. Recuperado 20 de enero de 2023, de https://www.mouser.es/ProductDetail/Vishay-Semiconductors/VCAN36C2-03G-E3-08?qs=tlsG/Ow5FFj9EZ8TpzRDWQ==

[48] Diodo Schottky RB496EATR. Mouser. Recuperado 20 de enero de 2023, de https://www.mouser.es/ProductDetail/ROHM-Semiconductor/RB496EATR?qs=4kLU8WoGk0tzYHEk1W59vw%3D%3D

[49] LED ROJO KP-3216SRC-PRV. RS. Recuperado 20 de enero de 2023, de https://es.rs-online.com/web/p/leds/4663914

[50] Módulo Convertidor Serial - FTDI232. https://www.amazon.es/Yongse-Ft232Rl-Adaptador-Convertidor-Arduino/dp/B00Q6WRLRG

[51] RTC DS3231SN. AliExpress. aliexpress.com. Recuperado 20 de enero de 2023, de https://es.aliexpress.com/item/1005003483636437.html

[52] Portapilas CR2032. RS. Recuperado 20 de enero de 2023, de https://es.rs-online.com/web/p/portapilas/1854716

[53] Conector MicroSD Molex. RS. Recuperado 20 de enero de 2023, de https://es.rs-online.com/web/p/conectores-para-tarjeta-de-memoria/8967560

[54] Resistencia CR0805. RS. Recuperado 20 de enero de 2023, de https://es.rs-online.com/web/p/resistencias-de-montaje-en-superficie/7409072

[55] Condensador 0805 KEMET. Mouser. Recuperado 20 de enero de 2023, de https://www.mouser.es/ProductDetail/KEMET/C0805C104K5RAC7411?qs=jbRM9o5BbzPcVXF0wW9AYg==

[56] Visual Studio Code. Recuperado 5 de abril de 2023, de https://code.visualstudio.com/opengraphimg/opengraph-home.png

[57] Autodesk Fusion 360. (2020, 19 febrero). Creating an Enclosure from a PCB Board in Fusion 360. YouTube. Recuperado 22 de mayo de 2023, de https://www.youtube.com/watch?v=8Ny4kWdhTbg

[58] Santos, S. (2019, 2 octubre). Pines de salida SP32. Random Nerd Tutorials. https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

[59] Crespo, J. (2016, 13 noviembre). Interrupciones. Aprendiendo Arduino. Recuperado 20 de enero de 2023, de https://aprendiendoarduino.wordpress.com/tag/isr/

[60] Xukyo, X. (2021, 6 diciembre). Rebotes Botón. AranaCorp. Recuperado 20 de enero de 2023, de https://www.aranacorp.com/es/aplicacion-de-la-logica-anti-rebotes-en-arduino/

[61] Llamas, L. (2020, abril 7). Cómo se fabrica una PCB. Luis Llamas. Recuperado 15 de abril de 2023, de https://www.luisllamas.es/como-se-fabrica-una-pcb/

[62] Pasta de soldadura - Chip Quik SMDLTLFP. Amazon. Recuperado 15 de abril de 2023, de https://amzn.eu/d/iUBmfbv

[63] Horno de reflujo T962. VEVOR. Vevor. Recuperado 20 de enero de 2023, de https://www.vevor.es/horno-de-reflujo-automatico-c_10068/maquina-de-soldadura-de-horno-de-reflujo-infrarrojo-t962-800-w-180-x-235-mm-p_010227410114?v_tag=9262c8c0-9787-11ed-8eca-2fafa943c1b3.1

[64] Soldador y estaño. (2018, 23 noviembre). TuElectronica.es. Recuperado 20 de enero de 2023, de https://tuelectronica.es/soldar-con-estano/

[65] Impresora 3D. Amazon. Recuperado 19 de abril de 2023, de https://www.amazon.es/Comgrow-3D-Printers-Ender-3X/dp/B07BR3F9N6?th=1

[66] NRF-PPK2. Farnell. Recuperado 20 de abril de 2023, de https://es.farnell.com/nordic-semiconductor/nrf-ppk2/power-profiler-kit-ii/dp/3595499