Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

ESTACIÓN METEOROLÓGICA BASADA EN MICROCONTROLADOR ARDUINO

Alumno: Fernandez Gallego, Unai

Director: Fernandez Rodríguez, Pablo

Curso: 2020-2021

Fecha: Bilbao, 5 de julio del 2021

Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

ESTACIÓN METEOROLÓGICA BASADA EN MICROCONTROLADOR ARDUINO

DOCUMENTO 1- *MEMORIA*

Alumno: Fernandez Gallego, Unai

Director: Fernandez Rodríguez, Pablo

Curso: 2020-2021

Fecha: Bilbao, 5 de julio del 2021

RESUMEN

El trabajo de grado presenta el diseño, estudio y montaje de una estación meteorológica basada en el microcontrolador Arduino. A lo largo del proyecto, se va a realizar un estudio previo para analizar la viabilidad del mismo. A continuación, se realizará un diseño y primer testeo de todos los componentes que se van a usar durante el trabajo. Una vez se han analizado de forma individual cada uno se los componentes se realizará la fabricación y montaje de la estación meteorológica al completo. Por último, se hará una última comprobación de funcionamiento antes de su puesta en marcha en el exterior.

Para la realización del proyecto se va a hacer uso de programas y aplicaciones de programación y diseño 3D, además de programas de simulación de circuitos electrónicos.

ABSTRACT

The final Degree Project presents the design, study and assembly of a weather station based on an Arduino microcontroller. Throughout the project, a first preliminary study will be carried out to analyze its viability. Then, will be performed a design and a fist test of all the components that are going to be used during the elaboration. Once each component has been individually analyzed, the entire weather station will be manufactured and piece together. Finally, a last functional check will be made before the start up on the outside.

To carry out the project, it will be used programming and 3D design programs and applications, as well as electronic circuit simulation programs

LABURPENA

Gradu Amairako Lan Honetan Arduino mikrokontrolagailuan oinarritutako estazio meteorologiko baten diseinua, azterketa eta muntaia aurkezten ditu. Proiektuan zehar, aurretiazko azterlan bat egingo da proiektuaren bideragarritasuna aztertzeko. Ondoren, lanean erabiliko diren osagai guztien diseinua eta lehen testa egingo da. Osagai bakoitza banaka aztertu ondoren, estazio meteorologiko osoa fabrikatu eta muntatuko da. Azkenik, funtzionamendua egiaztatuko da azken aldiz, kanpoan abian jarri aurretik.

Proiektua egiteko, 3Dko programazio- eta diseinu-programak eta -aplikazioak erabiliko dira, baita zirkuitu elektronikoak simulatzeko programak ere.

Contenido

TABLAS

IMÁGENES

GRÁFICAS

GLOSARIO

- TFG: Trabajo Fin de Grado
- µC: Microcontrolador
- µP: Microprocesador
- Módulo: Elemento con función propia concebido para poder ser agrupado de distintas maneras con otros elementos constituyendo una unidad mayor
- Comunicación serie: proceso de envío de datos de un bit a la vez
- Bit: Binary digit / dígito binario
- Byte: Conjunto ordenado de 8 bits
- Float: variable de coma flotante
- MSB: Most significant bit / Bit más significativo
- LSB: Least Significant Bit / Bit menos significativo
- Loop: Bucle
- Standby: Modo de espera
- PowerDown: Apagado del sistema eléctrico
- Pinout: disposición de los pines / patillaje
- Datasheet: hoja de características
- BMS: Battery management system / Sistema de gestión de baterías
- RF: Radiofrecuencia
- LDR: Light-Dependent Resistor/ Resistor dependiente de la luz, fotorresistor
- CO2: Monóxido de carbono
- COV: Compuesto Orgánico Volátil
- LCD: liquid-crystal display / pantalla de cristal líquido
- Deadline: Plazo máximo

1. Introducción

A lo largo de este proyecto como trabajo de fin de grado en Ingeniería Electrónica Industrial y Automática se va a realizar el estudio y puesta en marcha de una estación meteorológica.

La idea del proyecto será la realización de una estación meteorológica totalmente autosuficiente, que no requerirá de suministro de corriente externa, y será capaz de afrontar a la intemperie condiciones meteorológicas de todo tipo. A esta estación externa le acompaña un módulo interno con capacidad de almacenar y mostrar en tiempo real los datos recogidos en el exterior.

Para ello, se hará un estudio previo de las posibilidades disponibles y los parámetros más importantes a analizar. Posteriormente, se hará un testeo individual de cada uno de los sensores, realizando las pruebas pertinentes para su correcto funcionamiento y acondicionamiento. Y finalmente, se realizará una prueba con todos los componentes debidamente conectados antes de sacarlo al exterior.

Para la implementación y control de los sensores, se hará uso de un microcontrolador Arduino, el cual se caracteriza por su facilidad de manejo y capacidad de adaptación a gran variedad de módulos digitales y analógicos.

Durante la realización de este proyecto, se hará uso tanto de componentes digitales como analógicos, por lo que se necesitará una base en programación y automatización, así como conocimiento en electricidad y electrónica de baja potencia. Por tanto, será necesario poner en práctica la materia aprendida a lo largo de los últimos años.
Pero con el conocimiento adquirido previamente no será suficiente, ya que será necesario un estudio individual y un aprendizaje continuo a lo largo de la realización del proyecto, aumentando de forma considerable el conocimiento en esta rama de la electrónica, así como la capacidad de resolución de problemas.

2. Objetivos y alcance del proyecto

El objetivo general del proyecto es el diseño y fabricación de una estación meteorológica completamente autosuficiente.

Aun siendo el objetivo general la puesta en marcha de un modelo completamente funcional de una estación meteorológica, también se disponen de subobjetivos a tener en cuenta para llevar a cabo el proyecto.

- Autonomía y autosuficiencia.
  Estos posiblemente son los dos objetivos de mayor importancia dentro del proyecto, ya que son la base para una estación meteorológica capaz de funcionar al exterior con total independencia.
  Disponiendo de autonomía, la estación será capaz de funcionar sin la necesidad de alimentación externa, pudiendo ser instalada tanto en una finca particular como al aire libre.
  Mediante la autosuficiencia, la estación es capaz de seguir funcionando sin necesidad de intervención de terceros, pudiendo funcionar de forma cíclica o actuar ante imprevistos.
- Reducción de costes.
  Siendo este un objetivo no principal, el valor del producto a construir siempre ha de estar presente, para analizar previamente la viabilidad del proyecto y ser capaces de encontrar alternativas o abaratar costes.
- Realizar una primera aproximación a un proyecto de diseño y fabricación de componentes electrónicos.
  Como todo proyecto pensado para poner en práctica lo aprendido durante el estudio de grado, este trabajo se va a realizar con la finalidad de utilizar lo aprendido y profundizar en el temario a desarrollar.
  Durante el proyecto se hará uso de conocimientos en electrónica y electricidad, así como programación y automatización.

3. Requisitos y presupuesto

Con el fin de cumplir los objetivos de diseño y fabricación, se han analizado detenidamente todos y cada uno de los requisitos para conseguir una estación: autosuficiente, de bajo coste y duradera.

- Componentes de calidad, pero no sobredimensionados.
  Este es un requisito de gran importancia para ser capaces de cumplir con los objetivos de durabilidad y bajo coste dentro del proyecto.
  Seleccionando los componentes que mejor se ajusten a las necesidades, se evitarán circuitos altamente costosos, aprovechando al máximo los recursos disponibles.
  Mediante la reducción de todo aquello prescindible, será posible disminuir en gran medida los costes finales, consiguiendo un producto final completamente operativo a bajo precio.
- Protección ante el clima.
  Este posiblemente sea uno de los requisitos de mayor importancia y más complicados a la hora de diseñar el proyecto.
  Al tratarse de una estación meteorológica, se encontrará continuamente a la intemperie, requiriendo un diseño óptimo para ser capaz de afrontar el clima sea cual sea el momento o lugar.
- Suministro propio de energía.
  Es de vital importancia que la estación meteorológica nunca deje de funcionar, ya que se perderían datos y por lo tanto no cumpliría su función.
  Para ello, requerirá de una fuente de alimentación distinta al suministro eléctrico de la ciudad, necesitando para ello algún componente para su generación y su posterior almacenamiento.

Tal y como se ha mencionado, el objetivo es realizar una estación de bajo coste, pero totalmente autosuficiente. Para ello, se va a realizar un presupuesto inicial mediante la comparación de modelos similares que se puedan obtener online.

Es complicado decir un valor exacto de la estación que se va a realizar durante el proyecto, ya que se trata de un modelo más completo que los disponibles ya a la venta. Estos sobrepasan los 150€, por ello, se marcará como objetivo unos 100€, para poder disponer de un margen de maniobra de 50€ para posibles ampliaciones o mejoras.

4. Funcionamiento y primeros pasos

La estación meteorológica que se va a desarrollar a lo largo del proyecto va a estar formada por diferentes partes, conectadas entre sí, a fin de conseguir los objetivos y cumplir los requisitos mencionados con anterioridad.

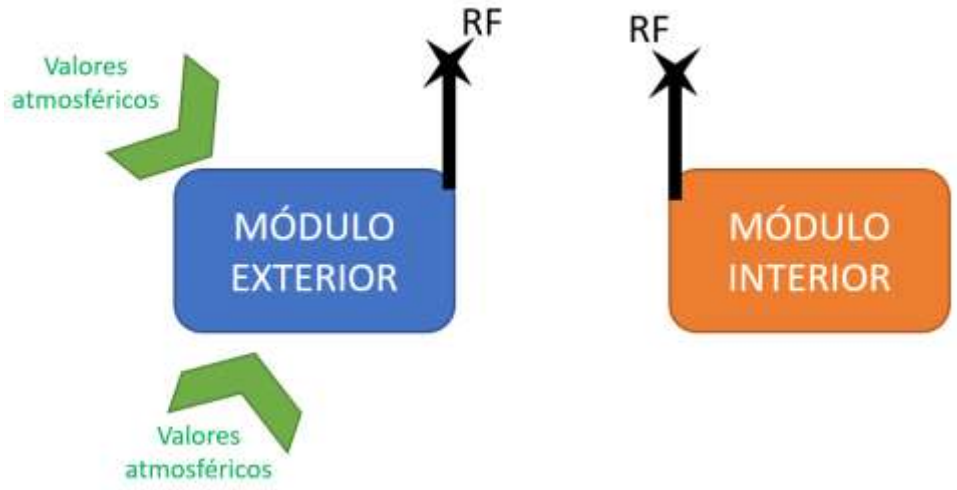
Para comprender el funcionamiento completo, se va a comenzar explicando su funcionamiento en conjunto y separando en diferentes módulos el proyecto dependiendo de su finalidad y ubicación.

Lo primero que se va a tener en cuenta, es que el proyecto va a estar separado en dos módulos con finalidades distintas: un módulo exterior que realizará el papel de estación meteorológica, y un segundo módulo interior encargado de mostrar los valores del exterior. Concretando un poco más en las funciones de cada módulo:

- Módulo exterior: será el encargado de leer los sensores, recoger los datos y analizarlos
- Modulo interior: guardará y mostrará los datos

Estos dos módulos estarán conectados mediante radio frecuencia, y realizarán intercambios de información constantes.

Imagen 1: Módulo exterior e interior



Siendo el módulo exterior la parte más importante de la estación meteorológica, se comenzará con la explicación y funcionamiento de cada una de sus partes, detallando su composición y módulos que lo forman. El módulo interior, al tratarse de un módulo de apoyo, se explicará y analizará a posteriori, dejando claras las funciones a desarrollar.

4.1. Módulo exterior

Tal y como se ha mencionado, este módulo será el encargado de recoger la información, procesarla y finalmente enviarla para ser guardada. Para ello, será necesario la utilización de sensores para la lectura de los datos ambientales, módulos de control para su correcto funcionamiento y componentes electrónicos de unión y alimentación.

Para explicar el funcionamiento de la estación, esta se va a separar en 4 partes distintas, a fin de facilitar la explicación y poder hacer un seguimiento del funcionamiento de cada una de las partes:

- Microcontrolador: dotará al circuito de suficiente potencia de procesamiento.
- Sensores: para la lectura de los parámetros ambientales.
- Módulos Extra: ayudarán a la ejecución del programa.
- Componentes de Potencia: alimentarán el circuito.

Microcontrolador

Como microcontrolador se usará el Arduino nano, el modelo de menor tamaño y potencia dentro de la familia Arduino. Mediante este µC (microcontrolador) será posible realizar la lectura de todos los sensores, ya que dispone de puertos I/O (Input/Output o Entrada/Salida) tanto analógicos como digitales. Además, dispone de un microprocesador programable capaz de realizar el control y gestión de información.

Al ser un µC de pequeño tamaño, se cumplirá el requisito de reducción de tamaño, y además su baja potencia brinda la oportunidad de reducir el consumo y disponer de mayor autonomía.

Para aumenta aún más la autonomía de este módulo, se va a hacer uso de la capacidad de suspensión del microcontrolador, permitiendo un apagado parcial del sistema y recogiendo datos cada cierto tiempo.

Además, este µC es de fácil manejo, fiable y de bajo coste, siendo por tanto un modelo perfecto para cumplir con los objetivos propuestos para la realización de la estación meteorológica.

Sensores

Los sensores ambientales serán de todo tipo, se requerirá un estudio individualizado para cada uno, y se necesitará en muchos de ellos un amplio conocimiento de su funcionamiento interno para conseguir una lectura fiable.

Para conseguir una lectura que abarque distintos parámetros de las condiciones ambientales, se leerán 11 de las más importantes a tener en cuenta. Para ello, se usarán distintos componentes y sensores, que se nombran a continuación:

- Temperatura – Sensor BME280
- Presión – Sensor BME280
- Humedad – Sensor BME280
- Velocidad del viento – Anemómetro
- Sentido del viento – Veleta
- Sensor de lluvia – Módulo MH-RD
- Sensor Ultravioleta – LM8511
- Sensor Acústico – MAX9814
- Sensor de luminosidad – LDR / fotorresistor
- CO_2 (dióxido de carbono) – CCS811
- COV (carbono orgánico total) – CCS811

Los sensores que se van a utilizar se pueden agrupar de distinta forma, dependiendo de si se observa el funcionamiento interno, la conexión disponible o la finalidad de haber optado por leer este parámetro.

Si se observa el funcionamiento interno, entre estos sensores se encuentran:

- Módulos con el circuito integrado, como puede ser el BME280 de Bosch
- Componentes electrónicos que requerirán un acondicionamiento de señal, como el fotorresistor
- Módulos con partes móviles como la veleta y anemómetro.

Centrando la clasificación en la conexión que requerirán para su lectura, se pueden clasificar en:

- Sensores digitales, los cuales requerirán de comunicación digital y la utilización de su respectivo protocolo
- Sensores analógicos, los cuales se podrán leer mediante los pines analógicos del Arduino tras acondicionar la lectura.

Para la conexión de cada uno de los sensores, se hará un estudio individualizado de cada uno, explicando finalmente antes del montaje el cableado final empleado a fin de adjudicar una conexión concreta a cada sensor.

Por último, agrupando los sensores según su la finalidad dentro del proyecto, se obtienen sensores de 2 tipos:

- Un primer tipo cuya finalidad es meramente informativa, como puede ser los parámetros ambientales de temperatura, presión, humedad, velocidad y sentido del viento…
- Un segundo tipo cuyos valores dependen de la calidad ambiental, teniendo estos parámetros unos valores límite que pueden perjudicar en mayor o menor medida la calidad de vida.

El primer tipo de sensores, informarán de las condiciones meteorológicas actuales sin ir más allá del estudio de los valores obtenidos. Siendo cierto que condiciones extremas de estos valores pueden acarrear una disminución en la calidad de vida, los sensores agrupados en el segundo tipo, alcanzando unos valores límite, pueden ser perjudiciales para la salud.

Valores como una alta incidencia de luz ultravioleta durante el verano, un ruido de fondo constante incluso a altas hora de la noche, o gran presencia de dióxido de carbono (CO_2) en el ambiente, pueden acarrear serios problemas a la salud, necesitando un estudio más preciso de las partes y pudiendo analizar la evolución a lo largo del tiempo.

Módulos extra

Además de los sensores, el módulo exterior también dispondrá de otros dos componentes:

- Reloj – DS3231
- Antena de Radiofrecuencia – nRF24L01

Estos componentes a diferencia de los otros, no se encargarán de leer los datos ambientales, pero van a ser completamente imprescindibles en el módulo exterior para llevar el control del tiempo y realizar el envío de información.

El reloj, como se explicará más adelante, será quien marce el tiempo de muestreo. Este módulo será configurado para controlar el transcurso del tiempo y ser capaces de realizar una medición de los parámetros cada cierto tiempo fijado.

El modelo de Reloj a usar deberá permitir la generación de una señal que permita el encendido del microcontrolador. Mediante el ajuste de esta alarma, será posible no solo fijar el tiempo de ejecución del programa, sino de controlar los tiempos de encendido y apagado del µC.

La antena de Radiofrecuencia, tal como su nombre indica, se tratará de una antena física, con la cual será posible realizar en envío de información desde la estación exterior hasta el módulo de guardado en el interior.

Este envío de información se realizará cada cierto tiempo, fijado por el reloj y durante el encendido del controlador, y será transmitida mediante señal de radio al módulo interior.

Componentes de potencia

Para finalizar con la enumeración de los componentes que van a formar el módulo exterior, se va a analizar los componentes de la parte de potencia. Por ello se estudiarán todos los requisitos necesarios para que el módulo tenga suministro eléctrico ininterrumpido sin necesidad de alimentación externa.

- La sección de potencia requerirá de una primera parte, que se encargará de la generación de energía. Esta energía se obtendrá íntegramente de recursos naturales, no generando ningún tipo de residuo y convirtiéndola en energía 100% renovable.
- La segunda parte estará formada por el almacenamiento de la energía. Este almacenamiento deberá ser suficiente para dotar al sistema de potencia cuando el generador no sea capaz de suministrar energía.
- La tercera parte serán todos aquellos circuitos de protección y transformación de energía. Siendo necesario que cumplan con las características de diseño y aportando una eficiencia suficiente para no tener grandes pérdidas de potencia.

Para la generación de energía, se hará uso de unas placas solares modelo YD-107X61. Estas placas suministran 1W de potencia a pleno rendimiento, alcanzando valores de 200mA para 5V de tensión de salida.

Con el fin de aprovechar al máximo la potencia solar, se colocarán en serie 2 placas solares alcanzando un valor cercano a 10V a la salida y requiriendo un convertidor Reductor (Buck converter) para ajustarse a los 5V del controlador. Se colocarán en paralelo todas las placas necesarias a fin de conseguir potencia suficiente para evitar un apagón del sistema.

Para el almacenamiento de esa energía se usarán baterías de litio-ion recargables de la marca Liitokala. Estas baterías serán las encargadas de almacenar la energía generada con las placas, y alimentar el circuito cuando este no dispone de sol o energía solar suficiente para alcanzar la potencia requerida.

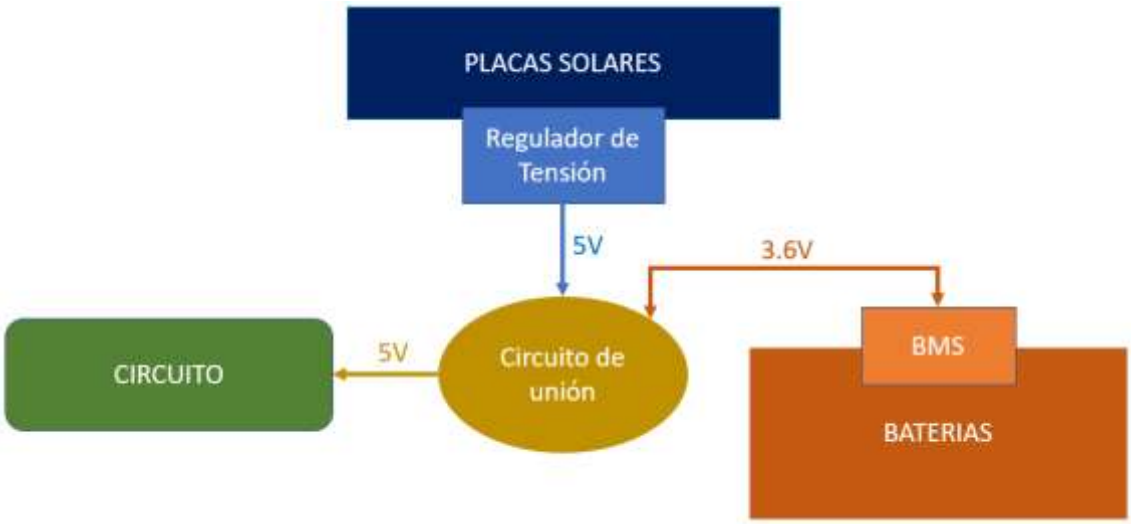
El pack de baterías a usar durante el proyecto se definirá una vez analizados los parámetros que ofrece el fabricante, y tras hacer las pruebas pertinentes. Para ello se realizarán pruebas de descarga y se fijarán la cantidad de baterías necesarias para poder suministrar potencia suficiente al circuito durante un rango de tiempo viable.

Estas baterías al estar formadas por compuestos químicos, requerirán protección tanto de corriente como de tensión, necesitando un circuito BMS (Battery management system / Sistema de gestión de baterías) para asegurar un funcionamiento correcto dentro de los parámetros del fabricante.

Para la unión de la parte de generación con la parte de almacenamiento y con el propio circuito, será necesario un circuito de unión Buck-boost converter (Convertidor reductor-elevador), que sea capaz de ajustarse a las tensiones de funcionamiento de cada una de las partes y que funcione como protección para todo el conjunto.

La conexión de cada una de las partes se detalla en el diagrama de bloques inferior, destacando el tipo de unión, la estructura y las tensiones de funcionamiento dentro de la parte de potencia.

Imagen 2: Diagrama de potencia



4.2. Módulo interior

Este módulo será el encargado de dar apoyo al módulo exterior. Para ello se almacenará los datos recogidos y los mostrará cuando el usuario los requiera.

Para el control de este módulo, se van a fijar distintos requisitos que los fijados en el módulo exterior, ya que al estar conectado a la red eléctrica no requerirá de un estudio detallado de la potencia, tampoco se tendrá que reducir su tamaño ni adecuarlo a condiciones meteorológicas adversas, pero necesitará gran potencia de calculo que permita gran flujo de información.

Es por esto que el microcontrolador usado en este módulo (Arduino Meda Pro) será uno de los de mayor potencia dentro de la familia Arduino, teniendo gran cantidad de pines de entrada y salida, mayor memoria, y claro está, mayor potencia que el módulo exterior.

Al encargarse este módulo tanto de mostrar como de almacenar los datos, se va a analizar de forma separada cada una de las funciones que realiza.

Receptor

El circuito receptor será el encargado de leer mediante la antena de radiofrecuencia los datos recogidos en el exterior. Estos datos contendrán los valores de todos los parámetros leídos mediante los sensores de la estación meteorológica.

En esta parte del módulo interior, se colocará un Arduino RF-nano, el cual tendrá la antena de radio previamente instalada y será capaz de leer la información enviada. Para ello, estará continuamente a la escucha de un posible dato, y preparado para recoger esa información y mandarla guardar.

Estos datos serán recibidos según el tiempo marcado en el reloj, y serán transmitidos desde la estación exterior, mediante su correspondiente antena de RF.

El guardado y presentación de los datos lo realizará otro circuito del módulo interior, por lo que la información recogida será enviada desde el receptor a la pantalla mediante comunicación cableada.

Pantalla y SD

La pantalla de este módulo, será un display LCD que mostrará de forma cíclica y continuada los 11 parámetros ambientales recogidos en el exterior y recibidos a través del receptor.

Cuando la pantalla recibe nuevos datos desde el receptor, detiene su ejecución, almacena dentro de su memoria los nuevos datos, y posteriormente los guarda de forma definitiva en la tarjeta de memoria microSD, para ser mostrados de nuevo de forma cíclica y continuada.

Por lo tanto, el circuito de la pantalla en realidad realizará dos funciones:

- Una primera función cíclica de muestra de datos.
- Y una segunda función que únicamente se ejecutará cuando un nuevo dato es recibido, y se encargará de almacenarlos.

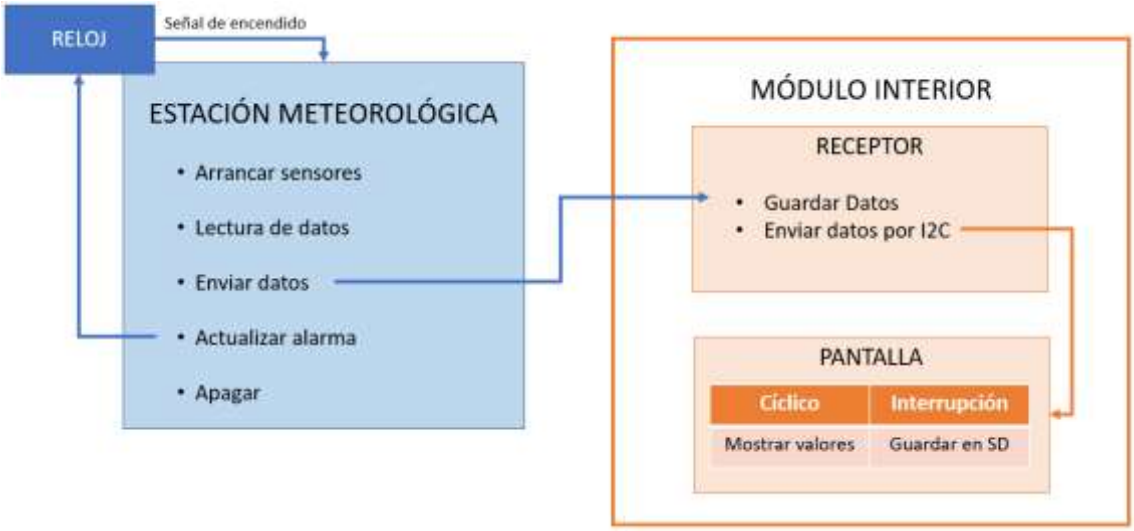
Se usará el modelo de pantalla ST7920, con una matriz completamente controlable de 128x64 pixeles, donde será posible mostrar los datos de forma numérica y gráficamente.

Para la escritura en SD, se usará un módulo de comunicación SPI con el que se podrá escribir y leer dentro de la memoria microSD.

4.3. Diagrama de flujo

Teniendo en cuenta el funcionamiento de cada una de las partes, se ha realizado un diagrama de flujo detallado de los pasos que va a seguir la estación meteorológica en su completo y las funciones a desarrollar por cada uno de los módulos que lo componen.

Imagen 3: Diagrama de flujo



Esta estructura se va a seguir de forma cíclica, y va a ser el reloj quien se encargue de marcar la ejecución.

El reloj elegido para este proyecto permite la programación de interrupciones, fundamental para marcar los tiempos de ejecución. Mediante la programación de la interrupción en el reloj, será posible generar una señal de encendido al módulo exterior, encendiendo el µC solo cada cierto tiempo. Al permanecer encendido durante espacios cortos de tiempo, permitirá estar la mayoría del tiempo en standby ahorrando energía.

La estación será arrancada por el reloj cada 5 minutos, ajustándose para que salte a las horas en punto y a continuación a y 5, a y 10… Y tras ser arrancada, ésta alimentará a todos los sensores de forma escalonada realizando la lectura y si procede apagándolos.

Durante la inicialización de los sensores, se tendrá en cuenta el tipo de alimentación que disponen, siendo necesario en algunos casos ser alimentados, mientras que otros estarán conectados de forma continua. Además, en muchos casos se necesitará una configuración previa o una señal de encendido que los arranque. Esta inicialización se estudiará detalladamente para cada uno de los componentes, observando en las hojas de características las condiciones óptimas para su lectura.

Para la lectura de los parámetros se realizará un barrido de todos los sensores instalados, realizando las acciones pertinentes dependiendo del modelo y forma de lectura. Cada sensor es un mundo, pero todos requerirán un encendido e inicialización, ya sea del propio modulo o de los pines del microcontrolador. A continuación, se realizará la lectura de forma analógica o digital, y los datos obtenidos se adaptarán al parámetro de lectura.

Para adaptar la lectura, en muchos casos se podrá obtener del datasheet la forma correcta de convertir el valor leído al parámetro deseado, en algunos otros, se requerirá de cálculos propios y pruebas de funcionamiento para ajustarlo.

Una vez obtenidos los datos, se enviarán mediante radiofrecuencia al módulo interior, donde serán guardados y mostrados. Para realizar ese envío se usará la Antena de radiofrecuencia previamente nombrada, con la cual será posible enviar de golpe un paquete de datos que contenga todos los parámetros y valores que se deseen mostrar y guardar.

Antes de volver a colocar el módulo en standby, se actualizará la hora de la próxima alarma en el reloj.

Para controlar la cantidad de datos a manejar, se va a realizar una toma de datos cada 5 minutos, obteniendo un total de 12 datos cada hora y 288 diarios. Estos datos serán mostrados en la pantalla, pero no todos se van a guardar, ya que esta cantidad de datos produciría un exceso de información innecesario.

Se ha decidido guardar uno de cada 3 datos, o lo que es lo mismo cada 15 minutos, de esta forma se tendrá un total de 96 datos diarios guardados, pero la frecuencia de refresco de la pantalla será cada 5 minutos. Obteniendo así valores más actualizados, y suficientes datos guardados como para poder ser analizados a posteriori.

El módulo interior, tal y como se ha comentado con anterioridad, dispone de dos partes perfectamente diferenciadas. La parte de recepción, que contendrá la antena de radio y un microcontrolador que guarde temporalmente los datos; y un módulo con pantalla y SD, que mostrará y guardará los valores recogidos.

Este módulo interior tendrá una ejecución cíclica que no estará delimitada por tiempo, y se mostrará de forma continuada los valores obtenidos en el exterior.

Además de la ejecución cíclica tendrá una interrupción cada 5 minutos, producida por la llegada de un nuevo dato, en este momento la pantalla deja de mostrar los datos por un tiempo y se dedica exclusivamente a guardar los valores y actualizar las variables del sistema.

4.4. Diagrama de Gantt

Un diagrama de Gantt es una herramienta gráfica que tiene como objetivo hacer una previsión y organización del tiempo previsto y orden a seguir. Mediante esta herramienta, será posible generar de forma visual una planificación y acotar los deadlines (plazos máximos) para cada una de las partes.

Para comenzar a con la realización del diagrama, se fijará como deadline final a principios de Julio, con el objetivo de tener la estación operativa de cara a la entrega de la memoria del proyecto, y pueda estar funcionando durante todo el verano hasta la presentación en septiembre.

A fin de facilitar el diagrama, se van a agrupar todas las tareas en grupos, y estas en subgrupos, consiguiendo una estructura que nos facilitará la ejecución del proyecto sin saltos.

Tabla 1: Diagrama de Gantt

Para realizar el diagrama, se ha tenido en cuenta:

- Lo primero que se hará es la selección y compra de componentes, ya que tras su compra es necesaria una espera de cerca de un mes para que dichos componentes sean entregados.
- Mientras se espera, se aprovechará para ir avanzando en la redacción de la memoria, dejando claros los objetivos y requisitos seguidos para la selección de los componentes.
- En todo momento se redactará lo que se va haciendo, con el fin de recordar con exactitud lo realizado en cada momento.
- Una vez se termina de probar uno a uno los componentes, llega el momento de probar el conjunto, y asegurarse de que funciona.
- Tras comprobar la estación por completo, se inicia el diseño y fabricación del exterior.
- Con todo probado y fabricado, comienza el montaje y puesta en marcha de los módulos al completo.
- Para finalizar, se harán las ultimas comprobaciones al informe.

5. Fundamento Teórico

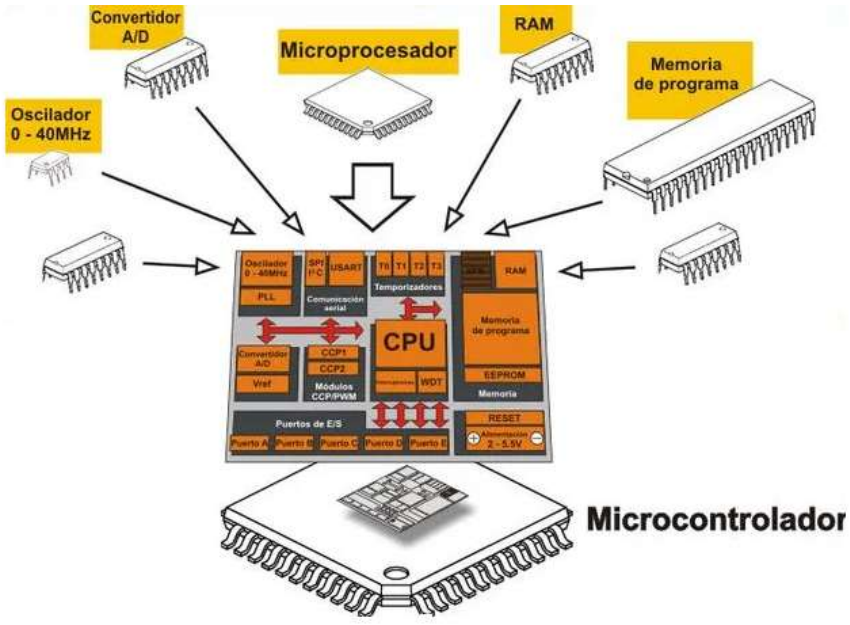
5.1. El Arduino

Hoy en día hay en el mercado, a disposición de cualquiera, gran cantidad de sistemas embebidos de diferentes características y prestaciones para todas las necesidades. Estos sistemas, brindan la posibilidad de realizar tanto a pequeña como a media escala el control de un sistema de complejidad reducida. Al tratarse de sistemas sencillos de controlar, no requerirá de gran potencia para su control, permitiendo reducir el tamaño del circuito de control, y permitiendo integrar todos los componentes en un solo chip.

Para este proyecto se va a hacer uso de un Microcontrolador Arduino, el cual tiene las características y facilidad de uso necesarias para un proyecto de estas características. A lo largo del proyecto se hará uso de diferentes modelos dentro de la familia Arduino, los cuales se ajustarán en cada momento a los requisitos del sistema.

Los Arduinos que se usaran durante el proyecto montan un integrado de la marca Atmel, una empresa del Grupo Microchip Technology Inc. Estos microcontroladores (µC), tal y como se ha comentado anteriormente, tendrán en su interior todos los componentes necesarios para su correcto funcionamiento, teniendo en el exterior Hardware necesario para su funcionamiento y control de pines de entrada y salida.

Imagen 4: Interior microcontroladores



Para la programación de estos microcontroladores se hará uso del IDE propio del Arduino. Mediante este compilador es posible realizar el programa completo y subirlo al Arduino. El lenguaje de programación que se usará en este compilador será el C++, teniendo pequeñas variaciones y funciones previamente predefinidas para ajustar el lenguaje de programación a este tipo de µC y facilitar su uso.

Arduino Nano

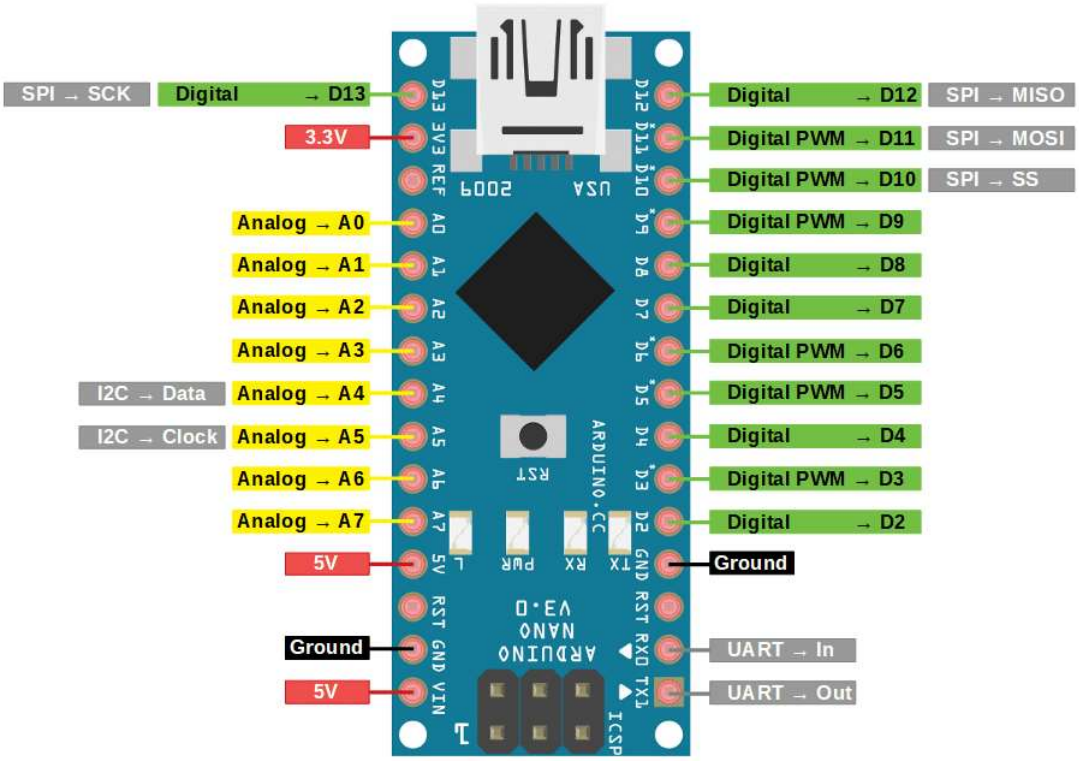
El primer µC que se va a usar será el Arduino nano. Este controlador irá instalado en la estación meteorológica base, ya que, por sus características, dispone de potencia y puertos I/O suficientes para el control de módulo, y es el micro de menor consumo y tamaño dentro de la familia Arduino.

El Arduino nano dispone de un procesador Atmega 328P, el cual dispone de un microcontrolador AVR de 8 bits y 32kbytes de memoria integrada. Entre todas las posibilidades que brinda este controlador, se hará especial hincapié en las capacidades de los pines de entrada y salida, ya que serán el requisito mínimo para conocer si el uso de este controlador satisfará los requerimientos de uso.

Tal y como se puede obtener desde la hoja de características de este controlador, el Arduino Nano dispone de:

- 8 pines de entrada/salida analógica, con un ADC (convertidor analógico-digital) de 10 bits
- 2 pines para comunicación I2C, dentro de los 8 analógicos disponibles
- 12 pines I/O digitales, 6 de ellos con PWM (Modulación por ancho de pulsos)
- 4 pines para comunicación SPI, dentro de los 12 pines digitales disponibles
- 2 pines para la comunicación serial UART (TX/RX)
- Pines de alimentación a periféricos tanto a 5V como a 3.3V

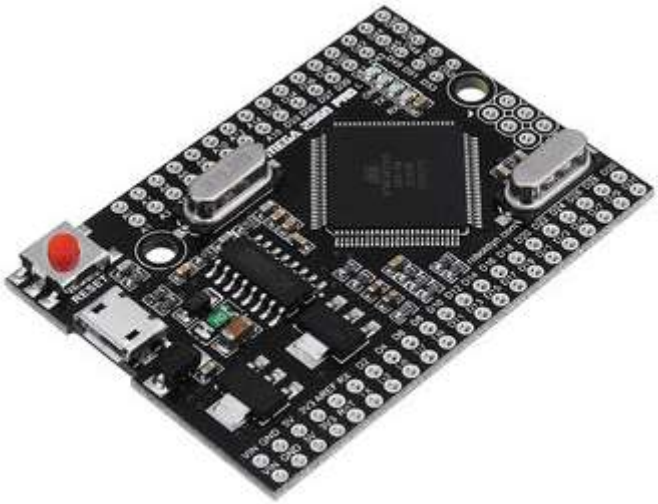
Imagen 5: Arduino nano - pinout



Arduino Mega Pro

El otro microcontrolador que se usará durante el proyecto será el Arduino Mega Pro. Este Arduino a diferencia del anterior es mucho más robusto y potente, haciéndolo un controlador de mayor consumo y potencia que será de gran ayuda para el tratamiento de gran cantidad de información.

Imagen 6: Arduino Mega Pro



Se ha elegido el Arduino Mega Pro por su potencia, y porque dispone de mayor memoria, lo cual será de gran ayuda para el tratamiento y muestra de datos. Este Arduino tiene un procesador ATmega 2560, controlador con todas las características del nano, además de muchas más salidas y características que las que finalmente se van a usar, pero permite el almacenamiento de datos y su tratamiento con gran capacidad, evitando bugs y problemas durante la ejecución.

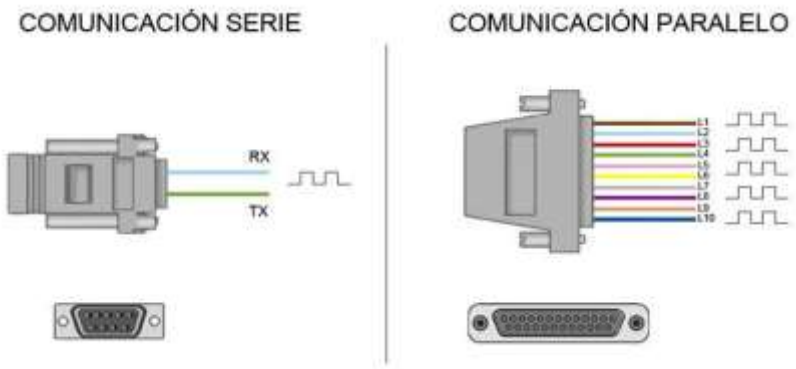
5.2. Protocolo de comunicación serie

La gran mayoría de microcontroladores tienen la capacidad de comunicarse con otros sistemas de su contorno. Es esta capacidad lo que les proporciona la posibilidad de controlar lo que les rodea u obtener información de su entorno. Estos dispositivos pueden estar integrados en la propia tarjeta, o puedes ser añadidas a posteriori haciendo estos µCs capaces de adaptarse a cualquier requerimiento.

Para la gestión y conexión con los módulos que se van a usar durante el proyecto, será necesaria la utilización de unos protocolos previamente definidos y compatibles con ambas partes. Existen gran cantidad de protocolos, que, dependiendo del medio usado y los requisitos, se ajustaran en mayor o menor medida al objetivo buscado.

Los protocolos de comunicación se podrían clasificar en dos grandes grupos: Formato paralelo y Formato serie. El Formato paralelo, como su propio nombre lo indica, usa buses paralelos, esto es, un cable para cada bit. Este tipo de comunicación tiene la ventaja de ser más rápida que la serie, pero esa cantidad de cable puede suponer un problema para buses largos y en la aparición de interferencias. Al contrario, en el formato serie, simplifica del hardware y dependiendo las circunstancias puede abaratar costes. Aunque el uso de este formato ralentizará la comunicación y hará necesarios conversores de paralelo a serie y viceversa.

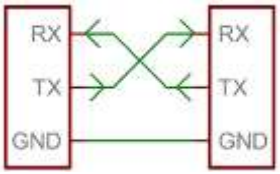
Imagen 7: Comunicación serie y paralelo



Para los µCs Arduino se hará uso del formato serie, ya que además de reducir el número de pines usados para la comunicación, el ancho de banda será más que suficiente para la capacidad de este micro. El Arduino dispone de tres distintos interfaces de comunicación serie:

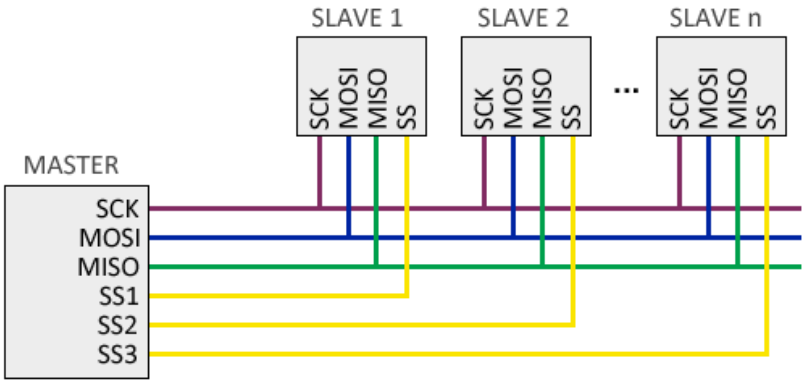
- **UART:** Universal Asynchronous Receiver-Transmitter (recepción-transmisión asíncrona universal). Este protocolo es uno de los más utilizados en sistemas embebidos gracias a su simplicidad y facilidad de implementación, aunque en gran cantidad de aplicaciones ha sido reemplazado por estándares más nuevos. Para este protocolo generalmente se usan dos señales de datos, ya que al ser asíncrona no hay señal de reloj. El pin RX será el encargado de la recepción de los datos, mientras que el pin TX será quien los transmita.

Imagen 8: UART



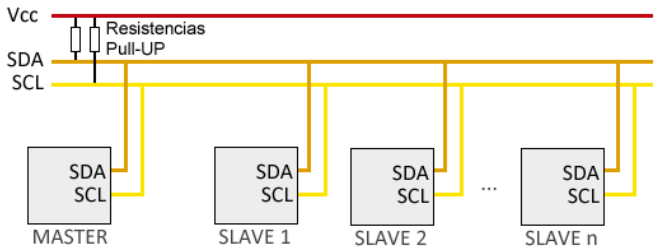
- **SPI:** (Serial Peripheral Interface). Este protocolo utiliza 4 líneas de comunicación: MISO para Maestro In Esclavo Out; MOSI, para el Maestro Out Esclavo In; SCK para la señal del reloj; y SS para seleccionar el esclavo. Este protocolo tiene la ventaja de ser capaz de comunicarse en Full-Duplex (enviar y recibir de forma simultánea), pero con muchos esclavos los pines de selección pueden aumentar considerablemente.

Imagen 9: SPI



- **I2C:** (Inter-Integrated Circuit). Este protocolo a diferencia de los anteriormente nombrados solo usa dos pines: uno para la señal de reloj (SCL) y el otro para la transmisión de datos (SDA). Al haber una única línea de transmisión de datos, tanto el maestro como el esclavo transmitirán por la misma línea, y será el maestro quien controle la comunicación. Al no disponer de pin de selección de esclavo, se hará uso del direccionamiento, adjudcando a cada módulo una dirección.

Imagen 10: I2C



Para este proyecto se hará uso de dos de los protocolos previamente nombrados: I2C y SPI. Los cuales, estando disponibles en ambos Arduinos a utilizar, y realizarán gran parte de las comunicaciones con los periféricos conectados al µC.

Ahora se procederá a explicar más detalladamente el funcionamiento de cada uno de los protocolos y los pasos a seguir para implementar dicho protocolo en el circuito.

5.2.1. Protocolo SPI

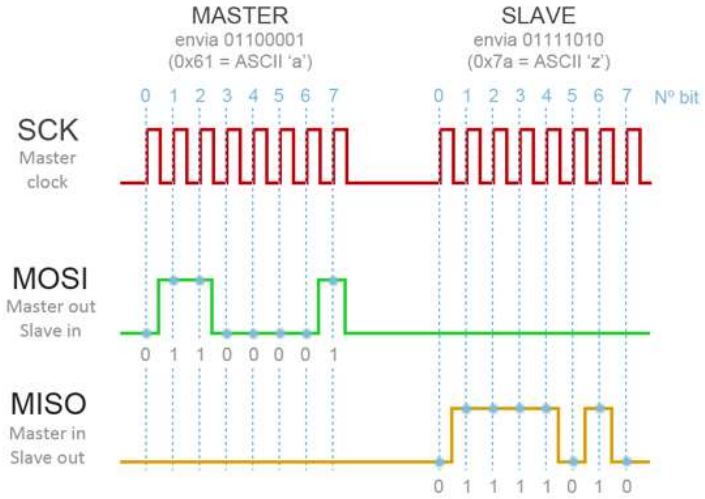
El Bus SPI es un estándar de comunicaciones usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. La ventaja de un bus serie es que minimiza el número de pines y el tamaño del circuito integrado a diferencia de comunicación en paralelo.

Este protocolo a diferencia del UART, es un protocolo síncrono, por lo que uno de los 4 pines necesarios para la comunicación enviará la señal de reloj. Otra de las líneas será para la selección de esclavo (SS). Mediante esta línea el maestro será capaz de seleccionar con cuál de los esclavos realizará la comunicación. Esto puede ser un inconveniente ya que, a mayor cantidad de esclavos, mayor número de líneas de selección serán necesarias y aumentará en gran medida el hardware utilizado. Los últimos dos pines serán los encargados de transportar la información. Al tratarse de una comunicación full-duplex, la comunicación puede ser en ambos sentidos de forma simultánea. El pin MISO (Master In, Slave Out) será el pin de comunicación de entrada de información al maestro, mientras que el pin MOSI (Master Out, Slave In) será el pin de salida de información desde el maestro.

Imagen 11: SPI - Maestro-Esclavo

Por defecto el maestro mantiene en estado alto todas las líneas de selección, y es cuando el maestro quiere establecer comunicación con el esclavo, cuando pone a 0 la línea de selección correspondiente. Es por esto que cuando el esclavo recibe un flanco de bajada en la línea SS se prepara para iniciar la comunicación.

Imagen 12: SPI - Comunicación



El maestro será el encargado de transmitir la señal de reloj por la línea SCK, y en cada flanco ascendente enviará un bit de información a través de la línea MOSI. Al ser una comunicación full-duplex el esclavo manda por la línea MISO de forma simultánea bits de información hacia el maestro.

Este protocolo tiene la ventaja de ser muy rápido y flexible, pudiendo comunicarse a velocidades de hasta 8Mhz en Arduino, y pudiendo enviar tramas de cualquier tamaño sin cortes ni interrupciones. Pero esto conlleva desventajas a tener en cuenta. Como se ha mencionado, al no estar los dispositivos direccionados, hará falta un pin de selección. En cuanto a las desventajas en la comunicación, al no haber paradas en las tramas no se puede realizar comprobaciones de si la comunicación ha sido exitosa o algún valor se ha perdido por el camino. Además, la longitud de la trama debe ser conocida por ambos dispositivos.

Comunicación SPI en Arduino

Centrando la atención en el funcionamiento del SPI en Arduino, tenemos la siguiente tabla de los pines asociados a esta comunicación.

Tabla 2: SPI – Pines Arduino

MODELO	***SS***	***MOSI***	***MISO***	***SCK***
Uno	10	11	12	13
Nano	10	11	12	13
Mini Pro	10	11	12	13
Mega	53	51	50	52

El pin SS únicamente será necesario en el caso de usar el Arduino como esclavo. No siendo este el caso para el proyecto, se usará otro pin digital para la selección del esclavo.

Para poder uso del SPI, el IDE de Arduino dispone de una librería propia que facilitará en gran medida la comunicación. La librería a usar será *SPI.h* y se puede obtener todas las funcionalidades y opciones de la página www.arduino.cc/en/reference/SPI.

Las funciones disponibles dentro de esta librería y que se van a usar son las siguientes:

SPI.begin()

Escribiendo esta función en el *setup()* se iniciará el hardware y los pines necesarios para inicializar el bus SPI.

SPISettings(Hz, DataShift, SPI_MODEx)

Esta función será la encargada de fijar los parámetros previos a la conversación. Mediante los tres parámetros dentro de la función se ajustará la velocidad de transmisión, el sentido de los bits y el modo de escritura de los datos.

- Hz: hace referencia a la velocidad del reloj. El propio Arduino se ajustará a la velocidad máxima a la que puede comunicarse, por lo que se recomienda colocar en este valor la velocidad máxima del esclavo. El valor se escribirá en Hz.
- DataShift: Mediante este parámetro se ajustará si la trama de datos se enviará comenzando por el bit más o menos significativo. Para mandar el más significativo primero, se escribirá: MSBFIRST; para mandar el LSB: LSBFIRST.
- SPI_MODEx: Este parámetro será de utilidad para fijar el modo de escritura:

Tabla 3: SPI - Modo se escritura

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)	Output Edge	Data Capture
SPI_MODE0	0	0	Falling	Rising
SPI_MODE1	0	1	Rising	Falling
SPI_MODE2	1	0	Rising	Falling
SPI_MODE3	1	1	Falling	Rising

Esta función se usa en conjunto con *SPI.beginTransaction().* Por lo que la forma correcta de enviar la instrucción anterior será:

SPI.beginTransaction(SPISettings(MHz, DataShift, SPI_MODEx))

SPI.beginTransaction()

Esta función usada sin argumentos colocará la línea SS del esclavo a LOW para iniciar la comunicación.

SPI.transfer()

Mediante esta función será posible enviar los datos. Puede ser llamada las veces que sea necesario hasta enviar la trama completa. Cada vez que se llama a la función se enviará un byte de información. Usando transfer16() es posible mandar dos bytes de información.

SPI.endTransaction()

Esta función colocará a 1 la línea SS, deteniendo la comunicación con el esclavo.

Se ha realizado un pequeño ejemplo de la utilización de las funciones anteriormente nombradas dentro de una función encargada de enviar un byte y recoger la respuesta del esclavo. Este código está disponible ene le Anexo, y muestra la estructura necesaria para ajustar el código al protocolo de comunicación.

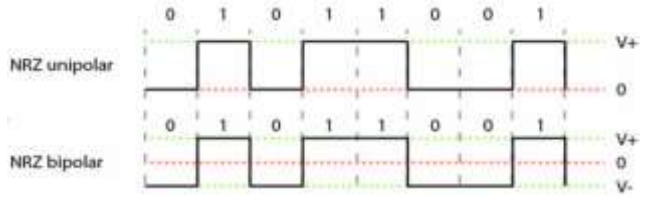
5.2.2. Protocolo I2C

Como se ha mencionado anteriormente, este protocolo solo hace uso de dos pines: SCL y SDA. Al tratarse de una comunicación síncrona, una de las líneas será la encargada de enviar la señal de reloj, por lo que únicamente se dispone de una línea para la comunicación. La línea SCL será la encargada de transmitir la señal de reloj y controlar la comunicación, mientras que la línea SDA será la encargada de comunicar ambos dispositivos.

Al realizar la comunicación por una única línea de datos, este protocolo será usado para transmisiones de pequeño ancho de banda (10kbit/s – 3,4Mbit/s), estando el rango de funcionamiento muy condicionado con las capacidades de ambos dispositivos y convirtiendo la comunicación en un sistema Half Duplex (comunicación unidireccional). La transmisión de estos datos requiere de un sistema de codificación que permita la transferencia de información de forma digital, es por esto que se hace uso de la codificación NRZ (Non Return to Zero).

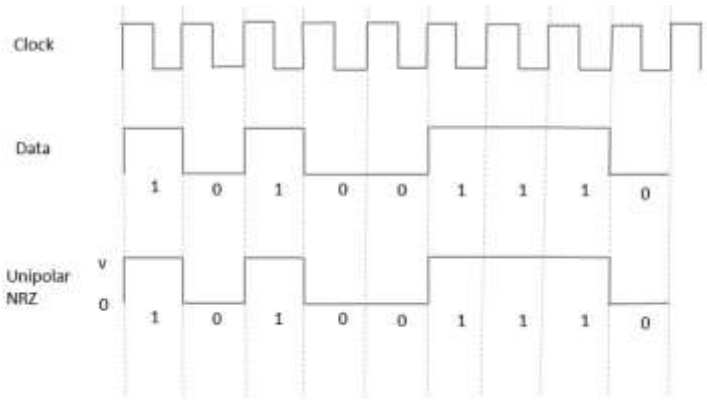
El código NRZ es una de las formas más frecuentes y fáciles de transmitir señales digitales usando únicamente dos valores de tensión. A cada uno de los valores de tensión se le asigna un valor y mediante la transición entre estos dos valores se consigue transmitir los valores deseados. Esta codificación se denomina Non Return to Zero (No Retorno a Cero) porque no vuelve a 0 entre dos bits consecutivos de valor 1. La codificación NRZ dispone de dos posibilidades dependiendo de los valores de tensión entre los que oscile: siendo NRZ unipolar si la señal es entre un valor de tensión V y 0; y NRZ bipolar si oscila entre +V y -V.

Imagen 13: Non Return to Zero



Para la comunicación I2C en Arduino se hará uso de la unipolar, ya que los valores de tensiones en los que funcionará el circuito nunca tendrá valores negativos.

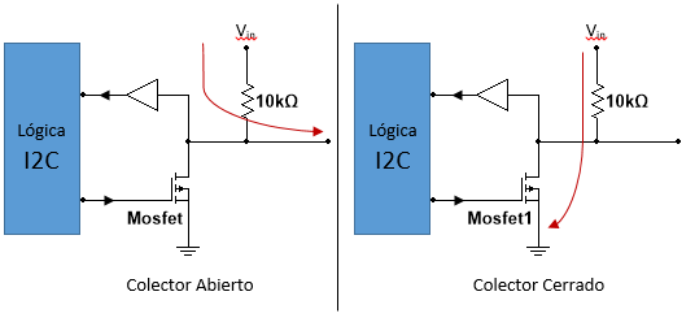
Imagen 14: Unipolar NRZ



Como se puede observar en la imagen, en cada ciclo de reloj se transmite un valor, siendo mayor la velocidad de transmisión a mayor frecuencia de reloj.

Volviendo al protocolo I2C, en las líneas de comunicación, será necesario el uso de resistencias pull-up, ya que ambos dispositivos pueden tomar el control del bus. Mediante estas resistencias la línea se mantendrá a nivel alto, y quien está transmitiendo únicamente tendrá que enviar los niveles bajos.

Imagen 15: I2C - Resistencias Pull-up



Mediante el control del Mosfet, podremos realizar al completo la comunicación. Manteniendo el Mosfet abierto, la línea se mantendrá a nivel alto, o en el caso de que alguien comunique se podrá leer el valor de la línea. Cerrando el Mosfet, se conectaría la línea a tierra, mandando así un 0. Cabe destacar, que además de las dos líneas de comunicación, es necesario que tengan la misma línea de referencia conectando entre si las masas.

Tal y como se ha mencionado anteriormente, al no haber pin de selección, cada uno de los dispositivos conectados al bus I2C tienen una dirección única. Además, al tratarse de un bus multimaestro, un mismo dispositivo puede ser o maestro o esclavo. Esto hace que no sea necesario que el maestro sea siempre el mismo dispositivo, esta característica se la pueden ir pasando los dispositivos que tengan esa capacidad.

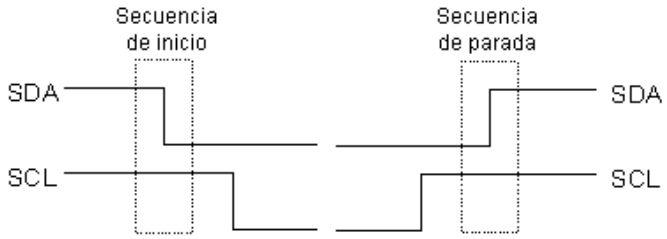
El protocolo está perfectamente definido, a fin de poder realizar la comunicación con únicamente los dos pines que tiene a su disposición.

- SDA sólo puede cambiar de valor cuando SCL está en baja.
- Cuando SCL está alta el estado de SDA indica el valor del bit.

Estas dos condiciones no se cumplirán al inicio ni al final de la comunicación. De esta forma se tendrá control de la duración de la comunicación.

- Start Condition: La línea SDA cambia de alta (1) a baja (0) con SCL a 1
- Stop Condition: La línea SDA cambia de baja (0) a alta (1) con SCL a 1.

Imagen 16: I2C - Líneas SDA-SCL

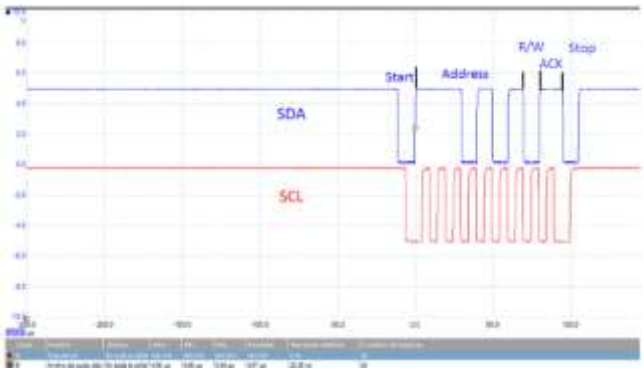


Teniendo estas condiciones claras, el proceso de comunicación siempre sigue los mismos pasos:

1. El maestro comienza tirando de la línea SDA y poniéndola a 0 mientras SCL está en 1. Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transacción.
2. El maestro empieza a mandar la señal de reloj por la línea SCL, mientras que por la SDA se dirige al dispositivo con el que quiere hablar. Para ello envía un byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con el que se quiere comunicar, y el octavo bit (A0) el LSB (Least Significant Bit, Bit menos significativo) se corresponde con la operación deseada (R/W), lectura=1 (recibir del esclavo) y escritura=0 (enviar al esclavo).
3. La dirección enviada es comparada por cada esclavo del bus con su propia dirección, si ambas coinciden, el esclavo continuará con la transmisión, y los otros ignorarán el resto de los intercambios esperando la próxima secuencia de inicio.
4. Cada byte leído/escrito por el maestro debe ser obligatoriamente respondido por un bit de ACK generado por el dispositivo receptor. El bite ACK (Acknowledge) es un pulso de reconocimiento que sirve como respuesta cuando los datos se han recibido correctamente. Este pulso se logra colocando la línea de datos a un nivel lógico bajo durante el transcurso del noveno pulso de reloj.

5. Cuando la comunicación finaliza, el maestro transmite una “stop condition” para dejar libre el bus.

Imagen 17: I2C - Señales SDA-SCL



Al transmitirse un bit por cada pulso de reloj, a mayor velocidad de reloj mayor tasa de bit/s. Para la gran mayoría de controladores se puede ajustar la velocidad para una velocidad estándar de 100kbit/s, pudiendo subir hasta los 400kbit/s en el “fast mode”. Algunos controladores con mayor velocidad y capacidad son capaces de aumentar la velocidad hasta los 3,2Mbit/s.

Comunicación I2C en Arduino

Arduino dispone de hardware para soportar la comunicación I2C. Este hardware está conectado a unos pines concretos, para poder conectarse fácilmente. Dependiendo del modelo puede cambiar la ubicación de estos pines.

Tabla 4: I2C - Pines Arduino

MODELO	SDA	SCK
Uno	A4	A5
Nano	A4	A5
Mini Pro	A4	A5
Mega	20	21

En nuestro caso se hará uso del modelo Arduino Nano, por lo que conectando a los pines analógicos A4 y A5 las líneas de comunicación será posible comunicarse mediante este protocolo.

Para poder usar el bus I2C, el IDE Standard proporciona la librería Wire.h, la cual contiene todas las funciones necesarias para controlar el hardware integrado.

A continuación, se muestra un listado con todas las funciones disponibles de esta librería y la función de cada una.

Wire.begin();

> Esta función es usada para iniciar el hardware integrado para controlar el bus I2C. Si la función se envía sin parámetros, el Arduino será el maestro de la comunicación. Por el contrario, si se desea que se comporte como esclavo, será su dirección la que se tenga que escribir en su interior.

Wire.beginTransmission(address);

Mediante esta función se prepara la secuencia de inicio y la dirección del esclavo que se desea que responda a esta transmisión.

Wire.write(data);

Esta función será la encargada de cargar en el buffer la información que se va a mandar en la siguiente transmisión. Este buffer será quien guarde los valores y los prepare para ser enviados. Estos datos en el caso de querer recibir de vuelta un valor, será el número de registro donde esa información estará guardada.

Wire.endTransmission();

Por medio de esta función se enviará por el bus toda la información que se ha preparado mediante las funciones anteriores. Realizará la secuencia de inicio, mandará la dirección del esclavo, enviará los datos guardados en el buffer y se realizará la secuencia de parada. Esta función devolverá el valor del ACK, siendo 0 si la comunicación ha sido exitosa.

Wire.requestFrom(address, quantity);

Esta función solicitara un número concreto de bytes (quantity) a uno de los esclavos, escribiendo la dirección (address) de ese esclavo.

Wire.available();

Esta función devolverá el número de bytes disponibles para ser leídos. Es muy útil esta función para esperar hasta que todos los bytes solicitados estén preparados.

Wire.read();

Mediante esta función se obtiene el valor enviado por el esclavo, y es posible guardarlo en una variable para ser utilizado.

onReceive();

Con esta función, se adjudica una función concreta del código como respuesta cuando el maestro envía un dato. De esta forma, cuando el maestro escribe un dato, el esclavo a quien va dirigido dicho dato, puede procesar esa solicitud y prepararse para la solicitud.

onRequest();

Con esta función, se adjudica una función concreta del código como respuesta cuando el maestro solicita un dato. De esta manera, cuando el maestro solicite un dato, el esclavo podrá realizar su respuesta.

Wire.setClock();

Mediante esta función se podrá elegir la frecuencia del reloj, para ser capaces de controlar la velocidad de transmisión.

6. Componentes

En este apartado se va a realizar un estudio individualizado de cada uno de los componentes que se van a utilizar durante el proyecto.

Para analizar el funcionamiento y ser capaces de hacerlos funcionar, se van a seguir los siguientes pasos:

- Comprobar las características del componente y el modo de comunicación:
  - Comunicación serial (digital)
  - Comunicación analógica
  - Tipo de conexión

  Para ello se va a obtener de la hoja de características toda la información correspondiente al componente

- Realizar el código correspondiente que siguiendo la teoría debería funcionar
- Comprobar el código y que la ejecución sea la correcta
- En caso de necesitar alguna modificación, realizar los cambios necesarios en el código
- Ajustar el componente a los anteriormente probados para evitar solapes:
  - Mismos pines de comunicación
  - Potencia reducida por alimentación en paralelo errónea

Una vez probados todos los componentes por separado se va a probar el conjunto antes de realizar el montaje.

6.1. Temperatura, Presión y Humedad – BME280

Los tres parámetros más comunes a la hora de hablar de la meteorología son la temperatura, la presión y la humedad. Y para ser capaces de leer estos parámetros se hará uso del módulo BME280. El sensor BME280 es un sensor ambiental del fabricante Bosch que combina un termómetro, barómetro e higrómetro en un único dispositivo.

Respecto al rango, precisión y resolución de este módulo, podemos encontrar en la hoja de características que:

Tabla 5: Características BME280

	RANGO	PRECISIÓN	RESOLUCIÓN
TEMPERATURA	-40 a + 85 °C	±1 °C	0,01 °C
PRESIÓN	300-1100 hPa	±1 Pa	0,18 Pa
HUMEDAD	0 a 100%	±3 %	0,008 %

La comunicación con este módulo se realiza mediante comunicación serie, y puede realizarse mediante SPI entre otros. Al disponer el Arduino también este protocolo, mediante una conexión de 6 cables se puede establecer la comunicación. Para el montaje se hará uso de 2 señales de datos, 1 señal para el reloj, 1 para la selección de esclavo y otras 2 de alimentación. Este módulo funciona a 3.3V, por lo que, al no disponer de regulador de tensión integrado, se alimentará con el pin de 3.3V del Arduino.

Los 4 cables de comunicación serán los siguientes:

Tabla 6: BME280 Pines Arduino

SCL	D13	CLOCK
SDO	D12	MISO
SDA	D11	MOSI
CSB	D7	Selector Esclavo

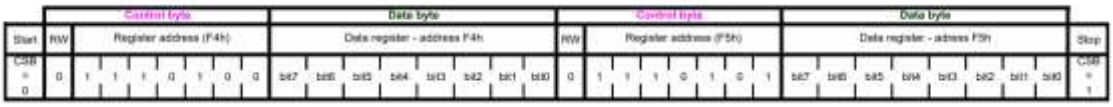
Comunicación

Tal y como se ha comentado anteriormente, se va a hacer uso de la comunicación SPI, por lo que, yendo a la hoja de características, al apartado de comunicación SPI, se dispone de un ejemplo de comunicación tanto de escritura como de lectura.

Para realizar la comunicación hay que entender de forma clara el funcionamiento de este módulo, ya que será importante configurar de forma correcta la comunicación mediante el SPISettings(). Para la velocidad de comunicación (la variable *velocidad_SPI*) se elegirá 10MHz; se enviará el bit más significativo primero: MSBFIRT; y el modo de escritura será el SPI_MODE0.

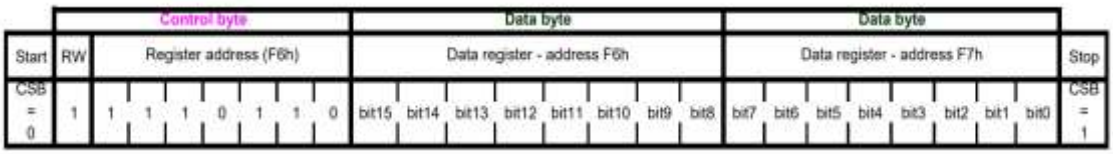
Para el caso de la escritura, se enviará en un inicio el registro a escribir y a continuación el byte a escribir en ese registro. Esta estructura de dirección y byte se podrá repetir las veces que sea necesario hasta enviar toda la información necesaria.

Imagen 18: BME280 - Escritura SPI



Para el caso de la lectura, se enviará un único registro, pudiendo hacerse varias lecturas consecutivas sin la necesidad de enviar el valor de dicho registro. En el ejemplo del datasheet, se puede observar como con cada lectura el registro aumenta en una posición. Eso será de gran utilidad, ya que se podrá leer un valor de más de un byte con un único comando.

Imagen 19: BME280 - Lectura SPI



El código completo de cómo quedaría la estructura para la escritura y para la lectura está disponible en el Anexo.

Registros y configuración

Para poder configurar el módulo previo a la lectura se irá al registro mapa de memoria de la hoja de características. En este mapa además de la configuración también está disponible las lecturas del sensor.

Tabla 7: Mapa de memoria

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1...0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88...0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

En este mapa de memoria se puede separar en 3 partes perfectamente diferenciadas.

Los registros de configuración:

Estos registros son los encargados de configurar el módulo y mostrar en todo momento lo que está sucediendo en dicho modulo.

- ID: este registro contendrá la información del módulo. Será de gran utilidad este registro, ya que cualquier lectura de este módulo que no dé como respuesta 60h indicará que el módulo lo ha arrancado correctamente.
- Reset: será un registro de solo escritura, enviando un B6h al registro se reiniciará el módulo
- Ctrl_hum: este registro configurará el sobremuestreo de la humedad. A mayor cantidad de valores el resultado será más preciso pero la lectura más lenta.
- Status: en este registro se hará uso del bit 3, el cual estará a 1 cuando una conversión se está realizando. Es por esto que durante el programa se espera a que este valor se vuelva 0.
- Ctrl_meas: configurará el sobremuestreo de la temperatura y de la presión. Además, se podrá seleccionar el modo de funcionamiento. El módulo dispone de 3 estados: Modo Sleep, en el cual no se recogerán datos; Modo Normal, recogerá datos de forma continua cada cierto tiempo fijado por el valor de standby; y Modo Forced, mediante el cual solo se recogerá un dato y volverá de forma automática al Modo Sleep.
- Config: en este registro se configurará el valor de standby anteriormente mencionado, el filtro IRR y el modo de comunicación SPI.

Los registros de datos:

Estos registros serán del F7h al FEh, y contendrán los valores recogidos por el sensor. Mediante lecturas consecutivas se obtendrán los valores para cada momento. El problema de los datos obtenidos es que a causa de las condiciones climáticas el valor obtenido se puede ver alterado y no ser preciso respecto a la realidad. Es por esto que se dispone de unos datos de calibración para dichos valores.

Los registros de calibración:

En estos registros están guardados unos valores dados por el fabricante para que la lectura sea precisa. Al inicializar el módulo será necesario leer estos datos para, realizando los cálculos necesarios, obtener el valor con mayor precisión.

Una vez leídos al inicio del programa se guardarán en variables que se usarán a la hora de calibrar la medición.

Programa

Cada vez que se quieran obtener los parámetros de temperatura, presión y humedad, se van a ejecutar en orden una serie de funciones creadas:

- Iniciar BME: mediante esta función se iniciará el módulo en el orden que sigue.
  - Resetear el módulo
  - Esperar a que arranque
  - Se obtiene del registro estado si se ha terminado la lectura y conversión
  - Se leen los registros de calibración
  - Se configura el sobremuestreo, se obtiene 8 veces cada valor para evitar datos erróneos
  - En el registro de control se pone en modo sleep para ahorrar energía
  - Se configura el tiempo en standby, el filtro IRR y se selecciona comunicación SPI.
- Leer Temperatura: esta función en realidad no lee el valor de la temperatura, ya que esa medición se ha hecho al arrancar el módulo. Tras reiniciarlo y esperar que arranque, el bit 3 del registro de estado se pone a 1. Mediante esta función se lee el valor guardado en el registro de la temperatura y se hacen los cálculos necesarios para convertir el valor en un numero decimal. Las funciones de presión y Humedad realizan el mismo procedimiento, aunque con distintos cálculos de conversión.
- Leer Presión
- Leer Humedad

Al ser una comunicación SPI con el módulo, los valores se envían de byte a byte, por lo que los valores contenidos en dos bytes hay que unirlos. Para ello se ha realizado la función dos_a_uno, la cual convierte dos bytes a una única variable word de 16bits.

6.2. Anemómetro

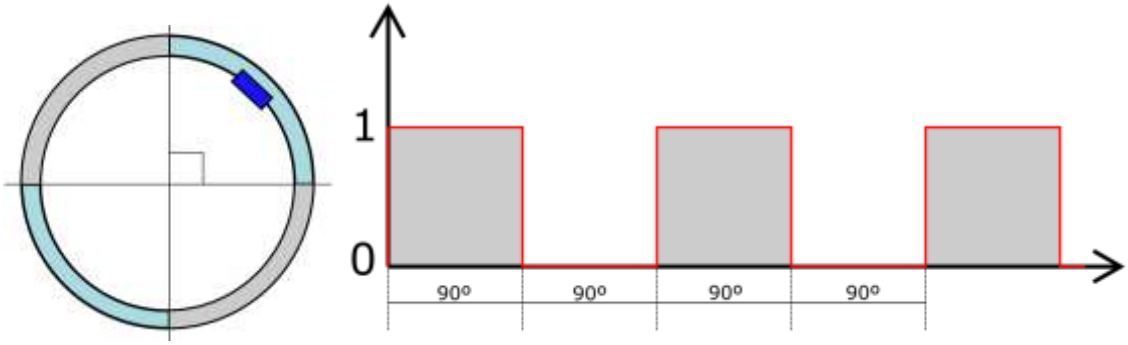
Para calcular la velocidad del viento, se va a hacer uso de un anemómetro, el cual, tras unas pruebas, se realizará un programa para acondicionar la señal.

Imagen 20: Anemómetro

Funcionamiento interno

Primero se realizará unas pruebas para entender su funcionamiento interno, a fin de ajustar el código al funcionamiento real. Lo primero que se observa es que el anemómetro dispone de 2 pines de salida, los cuales, en diferentes posiciones de las aspas, obtenemos un circuito cerrado o un circuito abierto, como si de un interruptor se tratase. Haciéndolo girar se observa que el estado de abierto o cerrado se mantiene durante 90º del movimiento de las aspas, por lo que obtendremos dos franjas en el que esos pines formarán un circuito abierto y otras dos un circuito cerrado de forma alterna por cada vuelta.

Imagen 21: Anemómetro - Funcionamiento interno



Sabiendo esto, si se alimenta uno de los pines con los 5V del Arduino, conectando el otro pin a una entrada digital, se podrá saber en qué momento pasa de cuadrante y calcular los tiempos de vuelta.

Acondicionamiento de señal

Para calcular la velocidad, se contará el tiempo que tarda el anemómetro en dar una vuelta, y teniendo el tiempo, será posible calcular la velocidad angular, y con ello la velocidad lineal.

Sabiendo que la velocidad lineal (v) es igual a la velocidad angular (w) por el radio de giro (R):

$$v = w \cdot R$$

La velocidad angular en radianes se puede obtener a partir de la fórmula:

$$w = 2\pi f$$

Llegados a este punto solo será necesario obtener el valor de la frecuencia (f), con la fórmula que lo relaciona con el periodo:

$$f = \frac{1}{T}$$

Siendo el periodo de vuelta 4 veces el tiempo que tarda en recorrer uno de los cuadrantes, el cálculo se convierte en una simple formula que agrupa todas las anteriormente nombradas.

$$f = \frac{1}{4t} \rightarrow t = \frac{1}{4f} \rightarrow t = \frac{2\pi}{4w} = \frac{2\pi R}{4v} \rightarrow \qquad v = \frac{\pi R}{2t}$$

Programa

Para el programa se va a realizar una única función llamada leer_v_viento, con la cual se seguirán los siguientes pasos para su correcta lectura:

- Creación de variables temporales
- Alimentar el anemómetro
- Bucle que realizará 4 medidas: de esta forma se realizará una medida más precisa, ya que será mayor el muestreo. Además, 4 valores corresponderían a la suma de una vuelta competa.
    - Se espera a que el valor esté bajo
    - Se espera a que pase a alto
    - Se recoge de la función millis el valor actual
    - Se espera a que vuelva a estar a 0
    - Se recoge nuevamente el valor de millis
    - Se restan ambos valores y se obtiene el valor de milisegundos transcurridos
    - Si hay algún error durante la ejecución se devuelve un 1 y se inicia de nuevo
- Se convierte el valor obtenido en milisegundos en km/h
- Se desconecta la alimentación

Pruebas de funcionamiento

Tal y como se ve en la formula usada en el programa, se va a hacer uso de un factor de corrección para que el valor obtenido se ajuste lo máximo posible a la realidad. Durante estas pruebas se han detectados varios fallos a la hora de detectar los flancos de subida y da bajada en los cambios de cuadrante. Es por esto, que se ha realizado un filtro pasa-bajo con un condensador de 105nF.

Imagen 22: Rebote señal

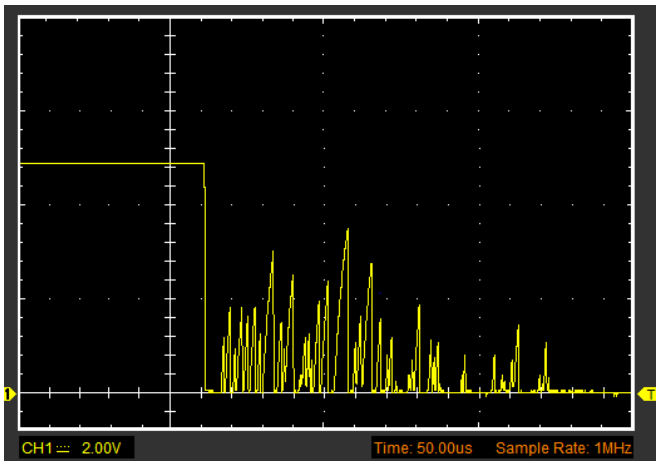
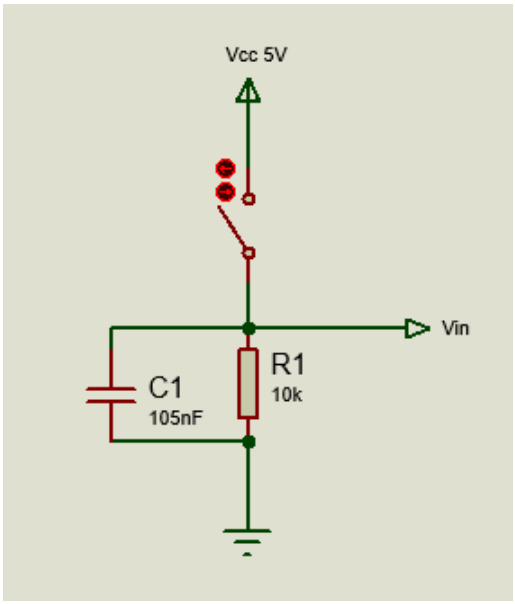


Imagen 23: Filtro pasa-bajo



Este filtro será de gran utilidad para reducir el ruido generado en los estados de transición y únicamente leer cuando efectivamente se ha realizado ese cambio de valor.

Para el factor de corrección se han realizado varias pruebas a distintas velocidades y se ha obtenido un valor de 1,2.

6.3. Veleta

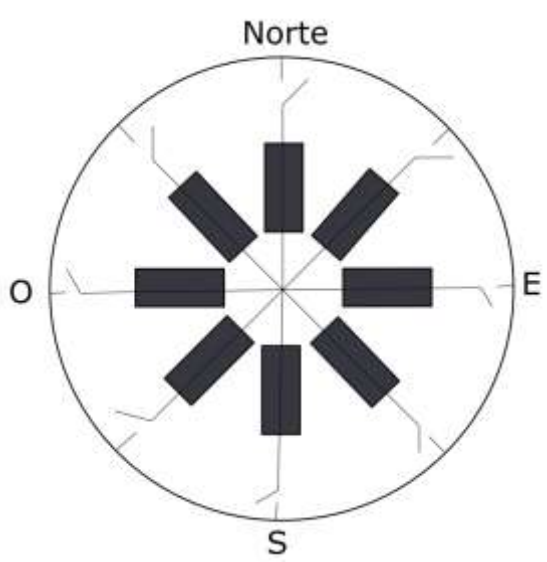
Para conocer el sentido del viento, se va a hacer uso de una pequeña veleta que se colocará en la zona más alta de la estación junto al anemómetro.

Imagen 24: Veleta

Funcionamiento interno

La veleta dispone de dos pines de salida, los cuales, para diferentes posiciones de la veleta, muestran una impedancia distinta entre sus bornes. Tras varias pruebas se ha podido obtener que dispone de 8 valores distintos, correspondiendo cada valor a uno de los puntos cardinales.

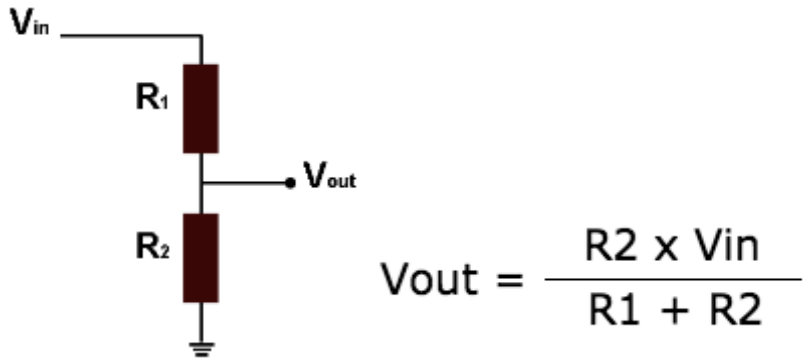
Imagen 25: Veleta - Funcionamiento interno



Acondicionamiento de señal

A fin de ser capaces de detectar el sentido del viento, será necesario poder leer el valor de la resistencia en cada momento. Para obtener los valores se va a realizar un divisor de tensión y se leerá con el Arduino los valores de tensión. Se conectará uno de los pines de la veleta a tierra y el otro tendrá en serie una resistencia conectada a 5V.

Imagen 26: Veleta - Circuito acondicionador



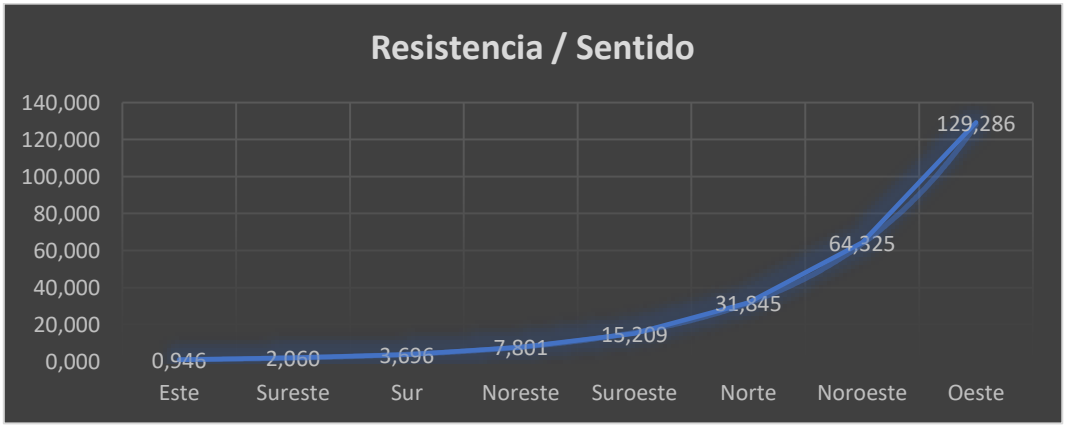
En esta prueba la resistencia que se usará será de 10kΩ y obtendremos los siguientes valores:

Tabla 8: Veleta - Lecturas

	Lectura	Tensión divisor (V)	Resistencia (Ω)
Este	612	3,010944282	0,946
Sureste	782	3,847317693	2,060
Sur	873	4,29502346	3,696
Noreste	946	4,654172043	7,801
Suroeste	982	4,831286413	15,209
Norte	1003	4,934603128	31,845
Noroeste	1013	4,983801564	64,325
Oeste	1018	5,008400782	129,286

Haciendo una gráfica con los valores obtenidos que relaciona la el valor de la resistencia y del punto cardinal, se vería que las resistencias siguen una escala exponencial.

Gráfica 1: Veleta - Resistencia/Sentido



Para linealizar los valores, se va a optar por aumentar el valor de la resistencia que se colocará en serie con la veleta.

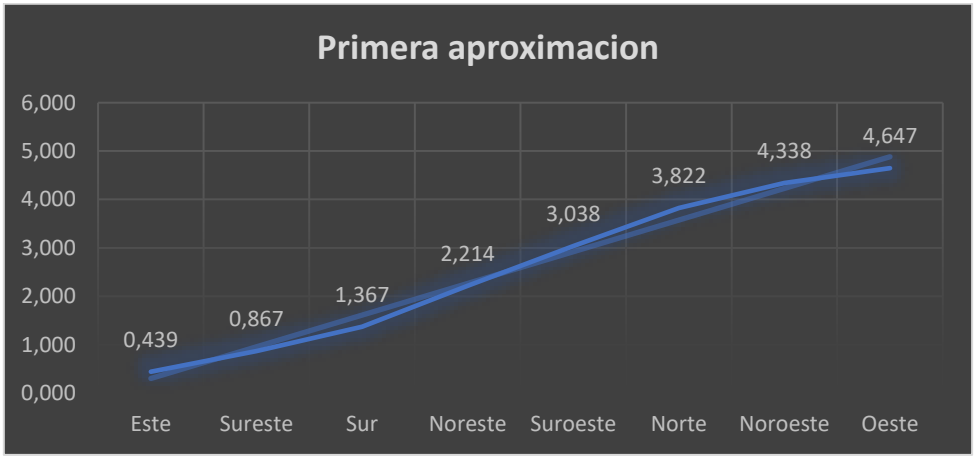
Se ha realizado la prueba con una resistencia de 10kΩ (9,82kΩ reales), y estos son los valores obtenidos:

Tabla 9: Veleta - Acondicionamiento de señal

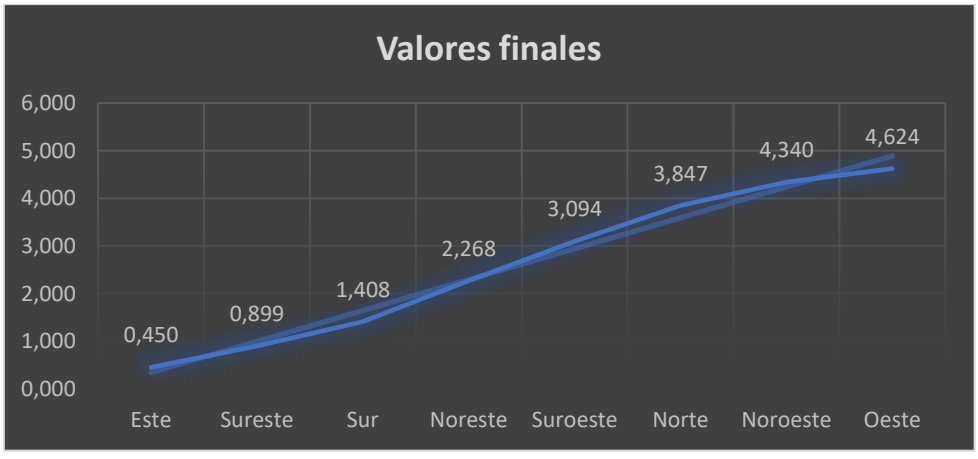
	Resistencia	Tensión divisor 10k	Lectura equiv.	Lectura obtenida	Tensión leída
Este	0,946	0,439	89	92	0,450
Sureste	2,060	0,867	176	184	0,899
Sur	3,696	1,367	278	288	1,408
Noreste	7,801	2,214	450	464	2,268
Suroeste	15,209	3,038	618	633	3,094
Norte	31,845	3,822	777	787	3,847
Noroeste	64,325	4,338	882	888	4,340
Oeste	129,286	4,647	945	946	4,624

Graficando los valores calculados y obtenidos tras la lectura:

Gráfica 2: Veleta - Primera aproximación



Gráfica 3: Veleta - Valores finales



Programa

Para la lectura y adjudicación del valor, se ha realizado el siguiente código. En él, se ha leído mediante un pin analógico el valor de entrada en un valor de 0 a 1023, y se le ha asignado a cada franja un sentido del viento. De esta forma, para cualquier valor dentro del intervalo fijado corresponderá al sentido previamente calculado.

De esta forma los intervalos quedarían de la siguiente manera:

Tabla 10: Veleta - Intervalos

Intervalo	*Punto Cardinal*
[000:099]	Este
[100:199]	Sureste
[200:299]	Sur
[300:499]	Noreste
[500:699]	Suroeste
[700:799]	Norte
[800:899]	Noroeste
[900:1023]	Oeste

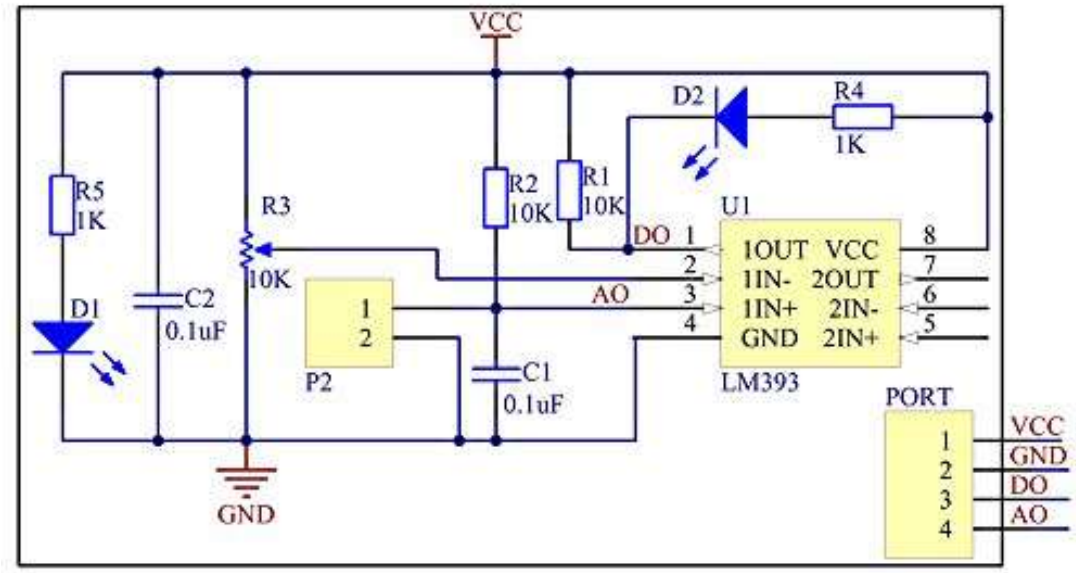
6.4. Sensor de lluvia

Para detectar la presencia de lluvia se hará uso del módulo MH-RD. Este sensor está compuesto por una plancha semimetálica que detectará la lluvia, y un circuito acondicionador de señal. La plancha funciona como una resistencia variable, la cual usa el agua para variar su valor. De esta forma, en un día completamente seco, la impedancia será muy grande, al estar las placas suficientemente separadas para evitar la circulación de corriente. Se comportará distinto cuando tenga agua sobre la plancha, ya que esta agua permitirá la circulación de corriente, variando el valor de la impedancia. En un día muy lluvioso, la resistencia será muy pequeña y a efectos de cálculo será equivalente a un cortocircuito.

Circuito acondicionador

El circuito acondicionador será muy sencillo de analizar, ya que únicamente tendrá un amplificador operacional (LM393) y un potenciómetro.

Imagen 27: Sensor Lluvia - LM393



Tal y como se puede ver en el esquema, este módulo dispone tanto de salida analógica como de salida digital.

En este proyecto se hará uso de la salida analógica, de esta manera será posible medir la cantidad de lluvia presente. La entrada del sensor de lluvia formará parte de un divisor de tensión, por lo que las variaciones en el valor del sensor será posible detectarlas a la salida.

El amplificador operacional y el potenciómetro formarán parte de la salida digital. El LM393 se encargará de comparar el valor obtenido del sensor mediante el divisor de tensión previamente nombrado, con un segundo divisor ajustable con el potenciómetro. Cuando el valor obtenido del sensor supere el ajustado en el potenciómetro, la salida digital se pondrá a 0.

Programación

Para la parte de programación de este módulo, al ser una salida analógica lo que obtendremos a la salida, el código se simplificará bastante.

Se usarán dos pines, uno analógico como entrada del valor y uno digital para controlar la alimentación del módulo.

Se realizarán dos funciones para el funcionamiento de este sensor.

La primera función:

- Encenderá el módulo
- Leerá el valor
- Y separará ese valor en una de las tres franjas de intensidad que se van a fijar

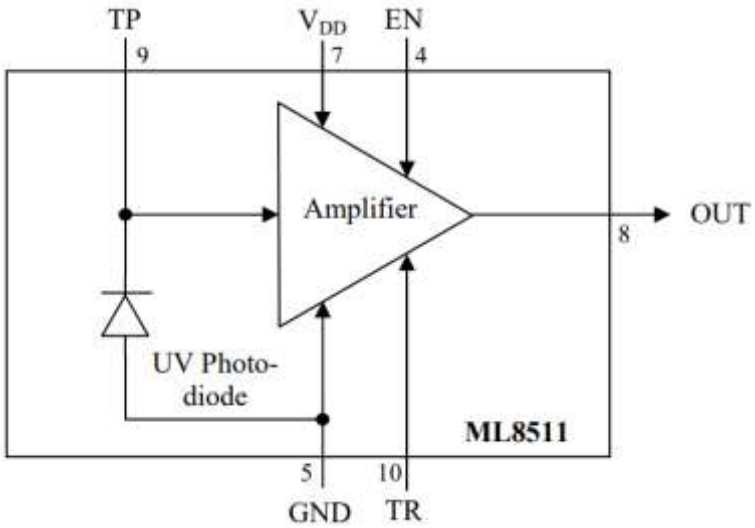
La otra función que se realizará, escribirá en pantalla en modo texto el valor de la intensidad. Esta función se ajustará más adelante para que en vez de mostrar en pantalla, guarde ese valor en una SD.

6.5. UltraVioleta – LM8511

La luz ultravioleta, también denominada luz UV, es un tipo de radiación con ondas de aproximadamente 400 nanómetros. Esta frecuencia se encuentra por debajo de la frecuencia de luz perceptible por el ojo humano. La luz ultravioleta está presente en los rayos solares, y según el uso e intensidad, puede traer beneficios o consecuencias graves al ser humano.

El sensor a usar para detectar la intensidad de luz UV será el GY-LM8511. Este módulo tendrá en su interior un fotodiodo ultravioleta junto con un amplificador operacional, tal y como sale en la hoja de características del componente.

Imagen 28: Circuito sensor UV



Si observamos la tabla de alimentación, la tensión de alimentación del sensor es de como máximo 3.6V, por lo que el sensor no se podría conectar a los 5V del Arduino. Por suerte en este módulo se dispone de un regulador de tensión, el 662K, el cual regula la tensión de entrada en un rango de 1.8 a 6V, a los necesitados 3.3V.

Para la conexión de los pines se obtiene la siguiente tabla del datasheet.

Tabla 11: Sensor UV - Pines

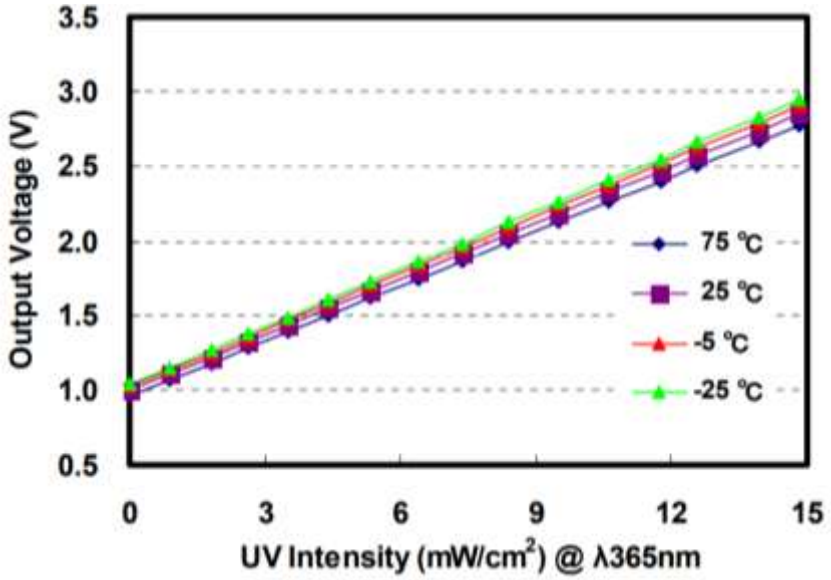
Pin	Symbol	I/O	Function
7	VDD	PW	Supply voltage. Decouple this pin to ground with 0.1 μF capacitor.
5	GND	PW	Ground
4	EN	I	Active high enable pin. (High: Active mode, Low: Standby mode)
8	OUT	O	Output (Low in power down or standby mode)
9	TP	I/O	Test pin. Do not connect.
10	TR	I/O	Internal reference voltage. Decouple this pin to ground with 1 nF capacitor.
1,2,3, 6,11,12	NC	-	No Connection. Do not connect.

En el módulo no se tendrá acceso a todos los pines, ya que los pines TP y TR estarán internamente conectados.

En el pin Vin se conectarán los 5V del Arduino, el cual con el regulador suministrará 3.3V a VDD. El pin Enable se conectará a VDD, de esta forma cada vez que se alimente desde el Arduino, el regulador habilitará también la lectura. La salida (pin OUT) se conectará a un pin analógico para poder leer el nivel de tensión.

Por último, se obtendrá de la hoja de características la relación de la tensión de salida con la intensidad lumínica.

Imagen 29: UV - Relación Intensidad/Voltage



Se puede observar que la salida es totalmente lineal y con muy pequeña variación para diferentes valores de temperatura. Para interpretar el valor de tensión se realizará una función que convierta el valor de tensión leído en mW/cm^2.

Sabiendo que el valor de tensión leído entre la franja de intensidad de 0 a 15 es de 1 a 2.9V:

$$Intensidad = (V_{OUT} - 1) * \frac{15}{(2.9 - 1)}$$

El código completo está disponible en el Anexo.

Para estudiar la resolución se tendrá en cuenta la partición que hace el Arduino para su lectura analógica. Tal y como se ha comentado, Arduino digitaliza la señal analógica leída en un valor de 0 a 1023, teniendo así 1024 franjas de lectura. La tensión de alimentación, en este caso 5.036V equivale al valor 1023 y GND es 0. Como el sensor da un valor entre 1V y 2.9V, el mínimo valor obtenido por el puesto será:

$$Valor_{min} = \frac{1 * 1023}{5.036} = 203$$

El valor máximo en cambio será:

$$Valor_{max} = \frac{2.9 * 1023}{5.036} = 589$$

Esto deja: 589 – 203 = 386 valores diferentes. Al tener un rango de 0 a 15mW/cm^2, la resolución será de 0.038mW/cm^2.

6.6. Módulo Reloj Real-Time

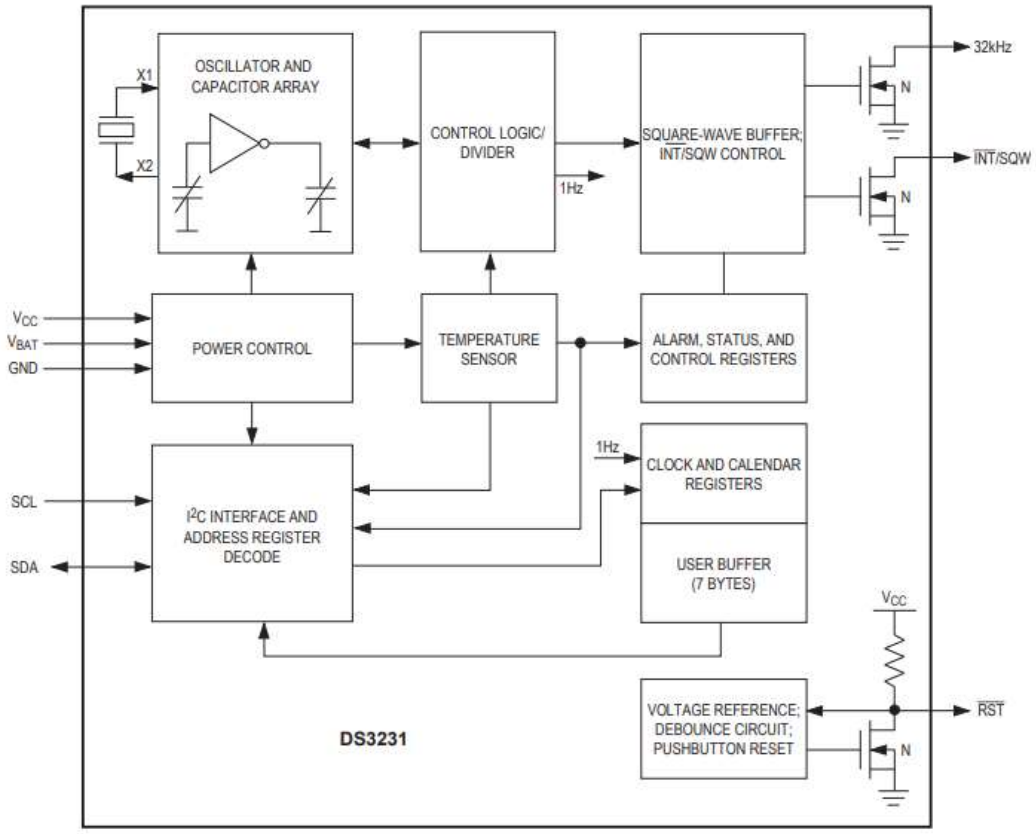
Para tener constancia en todo momento y ser capaces de controlar el tiempo transcurrido se hará uso de un módulo Reloj. Este módulo será el encargado de realizar el conteo y será de gran ayuda para "despertar" el sistema, ya que dispone de la posibilidad de crear una alarma.

El módulo estará alimentado con el pin 3.3V del Arduino, aunque también se le colocará una pila de botón por si en algún momento pierde la alimentación parta que no se detenga.

Comunicación y configuración

Este módulo dispone de comunicación I2C, por lo que se realizara tanto la configuración de los registros como la obtención de la información mediante este protocolo de comunicación. Será del datasheet (hoja de características) de donde se obtendrá toda la información necesaria para el control de los registros y las memorias.

Imagen 30: Reloj - Diagrama de bloques



Si se observa el diagrama de bloques, se puede ver que el decodificador de la interfaz I2C está conectado al registro del calendario, por lo que será posible modificar la información escribiendo en ese registro. El registro está directamente conectado a un buffer, el cual será de gran ayuda para la obtención de la información previo al envío por el bus.

Registros

A continuación, se muestra la tabla con los registros sobre los cuales tiene control el usuario.

Tabla 12: Reloj - Registros

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	$\overline{BBSQW}$	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Cada una de las líneas corresponde a un byte de información. La dirección de estos bytes vendrá dada en hexadecimal, siendo los siete primeros la fecha y hora actual, los siete siguientes la configuración de la alarma, y los cinco últimos información de control y temperatura.

Los registros de fecha y hora (00h a 06h) serán tanto de escritura como de lectura, por lo que durante la configuración se fijarán estos valores y luego se usarán únicamente de lectura.

Los registros de alarma (07h a 0Dh) funcionaran como comparadores. Configurando los valores de AxMy será posible usar esos comparadores en diferentes configuraciones. Según lo obtenido en el datasheet, a diferentes configuraciones saltara la alarma para diferentes comparaciones.

Tabla 13: Reloj - Registros Alarma

DY/$\overline{DT}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

DY/$\overline{DT}$	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE
	A2M4	A2M3	A2M2	
X	1	1	1	Alarm once per minute (00 seconds of every minute)
X	1	1	0	Alarm when minutes match
X	1	0	0	Alarm when hours and minutes match
0	0	0	0	Alarm when date, hours, and minutes match
1	0	0	0	Alarm when day, hours, and minutes match

Para este proyecto se hará uso de la alarma 2. Esto se debe a que da la capacidad de crear una interrupción cuando los minutos concuerden, en cambio en la alarma 1 esta alarma se creará teniendo en cuanta los segundos.

En el resto de registros únicamente se hará uso del Registro de control (0Eh) y del Registro de estado (0Fh).

Tabla 14: Reloj - Registro de control

Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR:	0	0	0	1	1	1	0	0

Dentro de este registro se configurarán los bits 1 y 2, ya que son los únicos que tienen interés en este proyecto. El bit 1 permitirá habilitar la interrupción mediante la alarma 2, y el bit 2 permitirá sacar la interrupción por el pin SQW. Para habilitar la extracción de la alarma por ese pin, el bit deberá estar en estado alto (1), y para habilitar la interrupción mediante la alarma el bit A2IE también deberá estar en alto.

Observando el registro de estado se pueden encontrar otros dos pines que hacen referencia a las alarmas.

Tabla 15: Reloj - Registro de estado

Status Register (0Fh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	OSF	0	0	0	EN32kHz	BSY	A2F	A1F
POR:	1	0	0	0	1	X	X	X

El bit 1 y 0 será los encargados de notificar cuando una de las alarmas ha saltado. Estos bits (cada uno con su respectiva alarma) se pondrán a 1 cuando la interrupción haya saltado. Este valor no se resetea solo, por lo que es importante borrar el flag tras cada interrupción para que ser capaces de detectar la siguiente interrupción.

Habiendo revisado todos los registros, obtenemos la siguiente configuración inicial a fin de conseguir el funcionamiento deseado.

- 0Bh -> 00000000b = 00h //Minuto Alarma 2
- 0Ch -> 10000000b = 80h //Hora Alarma 2
- 0Dh -> 10000000b = 80h //Dia Alarma2
- 0Eh -> 00011110b = 1Eh //Registro de Control

En los tres registros de la alarma 2 (0Bh, 0Ch y 0Dh) se escribirá esa configuración inicial pero cada vez se aumentará en 5 los minutos para que salte la alarma pasado ese tiempo.

En el registro de configuración (0Eh) únicamente se modificarán los bits relacionados con la alarma, dejando el resto como aparecen por defecto.

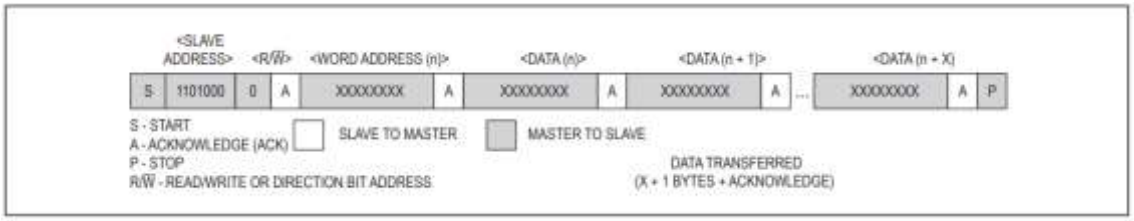
También se escribirá sobre el registro (0Fh) de estado, poniendo un 10001000 = 88h, limpiando así el flag.

Comunicación

Para la estructura de la comunicación, obtendremos de la hoja de características la estructura que se necesitará tanto para escribir como para leer de los registros previamente nombrados.

Para escribir siempre se seguirá la misma estructura: Primero se enviará la secuencia de inicio junto con la dirección del reloj. A continuación, se mandará la dirección del registro al que se querrá acceder. Tras eso, se enviará la información que se guardará en dicha dirección. Por último, se enviará la secuencia de parada.

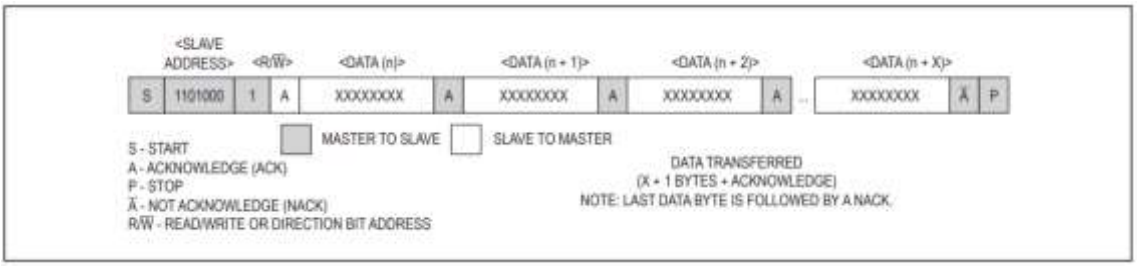
Imagen 31: Reloj - Escribir registro



Además, este módulo nos da la capacidad de escribir en varios registros de forma consecutiva. Esto es gracias a que cuando se manda la dirección del registro, el puntero se coloca sobre esa posición, y cada dato que obtiene hará avanzar en una posición el puntero, permitiendo escribir el siguiente dato mandado en la siguiente posición de memoria sin la necesidad de enviar la nueva dirección.

Para leer se usará una estructura similar. Primero se enviará la secuencia de inicio junto con la dirección del reloj. Y a continuación, se mandará la dirección del registro al que se querrá acceder. Tras eso, el esclavo enviará la información solicitada del registro. En el caso de la lectura también es posible leer registros consecutivos al igual que sucedía con la escritura.

Imagen 32: Reloj - Leer registro



Teniendo en cuenta la estructura explicada, se han implementado dos funciones, una para la lectura y otra para la escritura.

Configuración de la alarma

El objetivo de esta alarma será la de obtener una señal en la salida SQW que cada 5 minutos tenga un flanco descendente. Esta señal se usará en el Arduino para ser capaces de controlar el tiempo de ejecución del programa.

Para configurar la alarma se realizará una función del programa que accederá a los registros previamente nombrados. Además, para facilitar el código de configuración, se hará una escritura en cascada desde el primer registro de configuración hasta el último de la alarma 2.

Es muy importante tener presente que, al saltar la alarma, en el registro de estado se activa el flag de interrupción, por lo que para que la alarma vuelva a saltar, hay que limpiar el flag. Para resetear el valor del flag, se realizará una función que será llamada nada más salte la interrupción, a fin de detectar la siguiente.

Por último, se realizará una función que actualice el valor de la próxima alarma. Para ello se requerirá haber guardado el valor actual, sumar 5 minutos, y cargar en la memoria del reloj la hora de la siguiente interrupción.

6.7. Pantalla LCD

Para ser capaces mostrar en todo momento los valores obtenidos en la estación, se hará uso de una pantalla LDC suficientemente grande como para poder graficar de forma clara los valores. Tras realizar una búsqueda entre los diferentes modelos del mercado, se ha optado por usar el modelo ST7920, una pantalla de una única matriz de 128*64 pixeles.

Imagen 33: Pantalla LCD

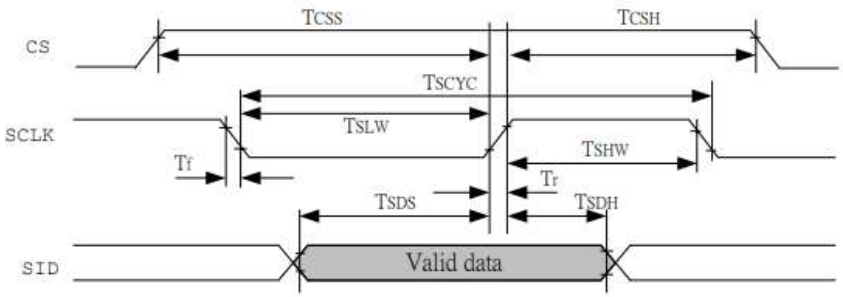
A diferencia de otros displays, este modelo es capaz de mostrar en su pantalla gran variedad de formas y letras, ya que la pantalla está formada por una única matriz totalmente controlable. Esta matriz dispone de un total de 8192 pixeles organizados en 128 columnas y 64 filas, y al ser posible la programación pixel a pixel, las posibilidades que da esta pantalla son muy extensas.

Este modelo concreto de pantalla dispone de 20 pines, de los cuales haremos uso de 8: 2 de alimentación para la placa, 2 para la iluminación led, 1 para el modo de comunicación y 3 para la comunicación.

Observando en el datasheet, se puede obtener los valores de alimentación y los modos de comunicación entre otras cosas. Para el caso de la alimentación, la pantalla se conectará a la salida de 5V del Arduino, en cambio para la retroiluminación led se conectará a la salida de 3.3V, a fin de reducir el consumo y observando que la iluminación a esa tensión es suficiente. Para el tipo de comunicación, disponemos tanto de comunicación serie como de comunicación paralelo. Para este proyecto se va a hacer uso de la comunicación serie SPI, y es por esto que únicamente se requerirán 3 pines para realizar la comunicación entre la pantalla y Arduino.

En el apartado Comunicación SPI en Arduino se menciona el protocolo de comunicación y las funciones a utilizar, pero será necesario obtener del datasheet la estructura correcta para realizar la comunicación.

Imagen 34: Display - Comunicación



Los datos enviados mediante SPI serán los 8bits pertenecientes al código del comando que se desee acceder. Para poder acceder a cada uno de los registros, se dispone de la siguiente tabla en las hojas de características correspondientes.

Tabla 16: Display - Registro de instrucciones

Instruction set 1: (RE=0: basic instruction)

Ins	code										Description	Exec time (540KHZ)
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
CLEAR	0	0	0	0	0	0	0	0	0	1	Fill DDRAM with "20H", and set DDRAM address counter (AC) to "00H"	1.6 ms
HOME	0	0	0	0	0	0	0	0	1	X	Set DDRAM address counter (AC) to "00H", and put cursor to origin ; the content of DDRAM are not changed	72us
ENTRY MODE	0	0	0	0	0	0	0	1	I/D	S	Set cursor position and display shift when doing write or read operation	72us
DISPLAY ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1: display ON C=1: cursor ON B=1: blink ON	72 us
CURSOR DISPLAY CONTROL	0	0	0	0	0	1	S/C	R/L	X	X	Cursor position and display shift control ; the content of DDRAM are not changed	72 us
FUNCTION SET	0	0	0	0	1	DL	X	0 RE	X	X	DL=1 8-BIT interface DL=0 4-BIT interface **RE=1: extended instruction** **RE=0: basic instruction**	72 us
SET CGRAM ADDR	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address to address counter (AC) **Make sure that in extended instruction SR=0 (scroll or RAM address select)**	72 us
SET DDRAM ADDR	0	0	1	0 AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address to address counter (AC) AC6 is fixed to 0	72 us
READ BUSY FLAG (BF) & ADDR	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Read busy flag (BF) for completion of internal operation, also Read out the value of address counter (AC)	0 us
WRITE RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data to internal RAM (DDRAM/CGRAM/IRAM/GDRAM)	72 us
READ RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM/IRAM/GDRAM)	72 us

El hecho de saber cómo funciona el display, abre un amplio abanico de posibilidades a la hora de usar el módulo y poder controlarlo. Pero al final, esto no es de gran utilidad, ya que realizar la programación desde 0 puede ser un inferno. Es por esto que para este proyecto se va a hacer uso de una librería, que, configurándola para nuestro modelo de pantalla y características, va a ser posible alcanzar un nivel de manejo mayor, reduciendo el tiempo invertido y aumentando la eficiencia del código.

La librería que se va a usar será la U8g2lib.h. La gran ventaja de esta librería es que se puede ajustar para cualquier modelo de pantalla, siendo posible utilizar las funciones de este proyecto en futuros proyectos.

Librería U8g2

Para que la librería reconozca el modelo de display, será necesario llamar a la función correspondiente a nuestro modelo: *U8G2_ST7920_128X64_F_SW_SPI u8g2()*.

Con esta función se configura el modelo ST7920; la cantidad de pixeles: 128 de ancho y 64 de alto; y el tipo de comunicación: en este caso serial SPI.

Una vez añadida la librería *.h y la función correspondiente al modelo de pantalla, se incluirán en el IDE varias funciones incluidas en la librería con las que será posible realizar el control y comunicación con el módulo.

Las funciones disponibles y que se van a usar en este proyecto son las siguientes:

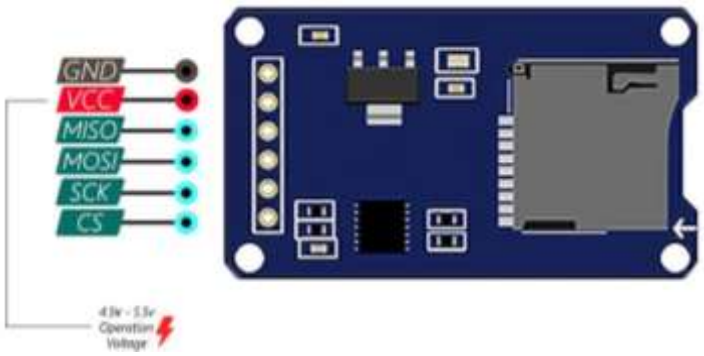
- u8g2.begin: para inicializar la pantalla con los ajustes de la función anteriormente nombrada.
- u8g2.enableUTF8Print: esta función en necesaria ejecutar tras iniciar para usar la función print y poder mostrar símbolos especiales.
- u8g2.sendBuffer: envía al display lo almacenado en la memoria y se muestra en pantalla.
- u8g2.clearBuffer: para limpiar la memoria y borrar el contenido de la pantalla.
- u8g2.setFont: sirve para fijar la fuente. Hay disponible una amplia variedad de letras y números para distintos tamaños. También hay símbolos y caracteres especiales.
- u8g2.setCursor: permite elegir el pixel concreto dentro de la matriz.
- u8g2.print: muestra el texto, numero o variable en la ubicación del cursor.
- u8g2.drawStr: a diferencia del print, permite elegir la posición del cursor. Muestra el valor del String en pantalla, dando problemas si se desea mostrar otra variable.
- u8g2.drawLine: dando 2 valores de la matriz traza una línea recta. Cada valor tiene que ir con sus coordenadas XY.
- u8g2.drawFrame: dibuja un recuadro hueco dando los valores del primer vertice, y su valor horizontal y vertical.
- u8g2.drawPixel: dibuja un único pixel en las coordenadas indicadas.

6.8. Lector de tarjetas microSD

Con el fin de poder acceder a posteriori a los datos recogidos, se va a hacer uso de una tarjeta SD en la cual se guardarán todos los datos. Para facilitar su extracción e implantación en el circuito, se va a usar un módulo que conectará de forma directa la alimentación con el Arduino, pero que tendrá un conversor SPI de por medio.

Mediante este convertidor SPI, será posible acceder directamente a la memora SD sin necesidad de usar pines extra del microcontrolador, haciendo su manejo más sencillo y rápido.

Imagen 35: Adaptador microSD



Para realizar la comunicación con la SD es necesario entender cómo es su estructura interna y sus propios protocolos de comunicación, ya que cada fabricante puede modificar ligeramente sus características a fin de hacerlo más eficiente o seguro, pero es una traba a la hora de usar una SD externa.

Es por esto que, para acceder a la memoria interna de la SD, se va a hacer uso de una librería universal y proporcionada por el propio IDE del Arduino. Esta librería facilitará el acceso a los datos, ya que englobará la estructura y protocolos de comunicación necesarios para leer y escribir dentro de la SD.

La librería que se va a usar será SD.h y entre otras funciones, se van a usar las siguientes funciones a lo largo del proyecto.

- SD.begin: mediante esta función se inicializará el módulo con su correspondiente configuración. Si la función devuelve un 0 significará que la SD no se ha iniciado correctamente, por lo que lo más apropiado sería apagar el módulo y volver a intentar la inicialización.
- File: crea una variable con estructura fichero. Tras crear esta variable los datos se almacenan en esta variable y cualquier modificación o lectura que se desee hacer del fichero se hará mediante el uso de esta variable.
- “fichero”.print: escribirá en el archivo abierto mediante la variable fichero.
- “fichero”.close: se cerrará el fichero almacenado en la variable fichero
- SD.open: abre el archivo seleccionado. En caso de que ese archivo no existiese primero lo crearía.
- “fichero”.read: lee un carácter del fichero seleccionado. Al principio se leerá el primer carácter, y cada lectura que se haga leerá el carácter siguiente.
- SD.exist: comprobará si existe el archivo especificado.
- SD.remove: elimina el fichero seleccionado
- SD.rmdir: elimina un directorio de memoria, como puede ser una carpeta de archivos

6.9. CO_2 y COV – CCS811

La estación meteorológica estará dotada de varios componentes de medida que llevarán un seguimiento de la contaminación medio ambiental producido tanto por gases como por partículas en suspensión. El primer módulo que se colocará será un medidor de dióxido de carbono y de compuestos orgánicos volátiles.

El CO2, dióxido de carbono, es un gas incoloro e inodoro y, además, es el principal gas de efecto invernadero de origen humano. Eso significa que contribuye al calentamiento global, cuyas consecuencias se notan a diario. Está presente de forma natural en la atmósfera, siendo la concentración cercana a los 405 según los datos de la OMM (Organización Meteorológica Mundial). La presencia y el aumento de estas emisiones se debe principalmente a: la combustión de carbono, petróleo y gas; la deforestación, ya que son los árboles quienes absorben CO_2; y la ganadería, ya que estos animales producen gran cantidad de metano durante la digestión.

Los compuestos orgánicos volátiles (COV) son todos aquellos hidrocarburos que se presentan en estado gaseoso a la temperatura ambiente normal. Suelen presentar una cadena con un número de carbonos inferior a doce y contienen otros elementos como oxígeno, flúor, cloro, bromo, azufre o nitrógeno. Su número supera el millar, pero los más abundantes en el aire son metano, tolueno, n-butano, i-pentano, etano, benceno, n-pentano, propano y etileno. Entre los COV hay compuestos extremadamente peligrosos para la salud están el Benceno, el cloruro de vinilo y el dicloroetano.

Alimentación y configuración de pines

Para entender el patillaje y alimentación del sensor se hará uso del datasheet. Si se contempla la tabla de características eléctricas, se verá que la tensión máxima de alimentación es de 3.3V.

Una vez sabemos la alimentación se observará la tabla de asignación de pines.

Tabla 17: CCS811 - Pines

Pin No.	Pin Name	Description
1	ADDR	Single address select bit to allow alternate address to be selected • When ADDR is low the 7 bit I²C address is decimal 90 / hex 0x5A • When ADDR is high the 7 bit I²C address is decimal 91 / hex 0x5B.
2	nRESET	nRESET is an active low input and is pulled up to V_{DD} by default. nRESET is optional but external 4.7KΩ pull-up and/or decoupling of the nRESET pin may be necessary to avoid erroneous noise-induced resets.
3	nINT	nINT is an active low optional output. It is pulled low by the CCS811 to indicate end of measurement or a set threshold value has been triggered.
4	PWM	Heater driver PWM output. Pins 4 and 5 must be connected together.
5	Sense	Heater current sense. Pins 4 and 5 must be connected together.
6	V_{DD}	Supply voltage
7	nWAKE	nWAKE is an active low input and should be asserted by the host prior to an I²C transaction and held low throughout.
8	AUX	Optional AUX pin which can be used for ambient temperature sensing with an external NTC resistor. If not used leave unconnected.
9	SDA	SDA pin is used for I²C data. Should be pulled up to V_{DD} with a resistor
10	SCL	SCL pin is used for I²C clock. Should be pulled up to V_{DD} with a resistor
EP	Exposed Pad	Connect to ground

Para la función que se va a realizar con este módulo nos fijaremos en varios de los pines. Se empezará colocando ADD a tierra para fijar la dirección del módulo en 0x5A. Tal y como se indica ahí, se colocará una resistencia pull-up en Reset. El pin INT se dejará sin conectar, ya que no se hará uso de las interrupciones generadas por este módulo. El pin que si se conectará será el de Wake (despertar) el cual se activará justo antes de realizar la medición, a fin de ahorrar energía. Este pin se conectará a una salida digital de Arduino, y para no dañar en modulo alimentando a 5V se realizará un divisor de tensión que ajuste a lo máximo posible a 3V.

Tabla 18: CCS811 - Pines alimentación

Logic High Input	nRESET, nWAKE, ADDR	V_{DD} - 0.6		V_{DD}	V

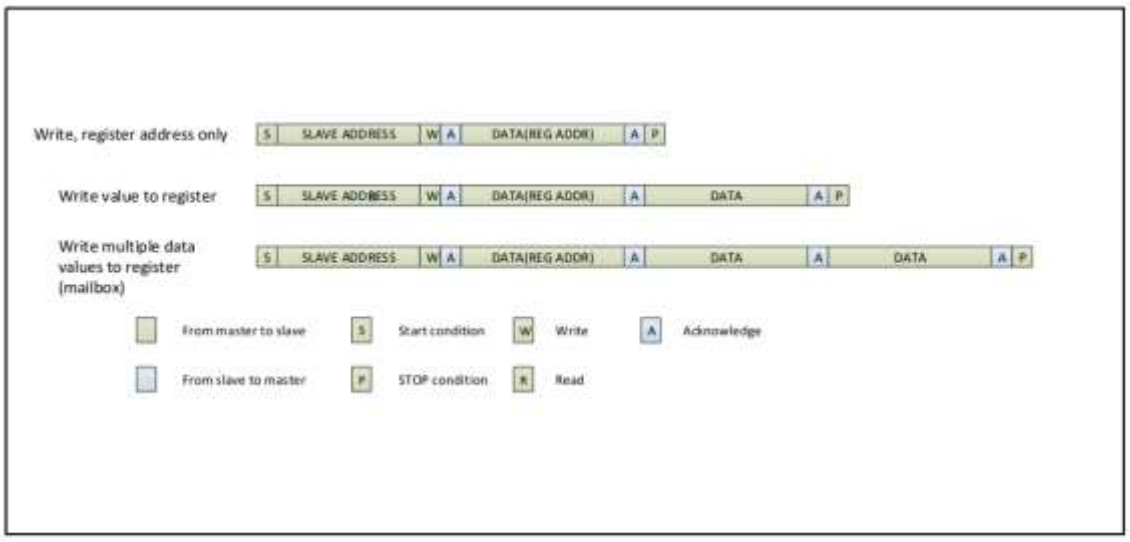
Comunicación y funcionamiento

Este módulo, al igual que otros usados en el proyecto, hace uso de la comunicación I2C. Es por esto que con únicamente dos cables y el uso de un protocolo de comunicación serie será posible realizar la comunicación con este módulo.

Observando en el datasheet el modo de lectura y escritura, será posible realizar la programación siguiendo los requisitos fijados.

Para escribir en un registro se realizará lo siguiente:

Imagen 36: CCS811 - Escritura Registros

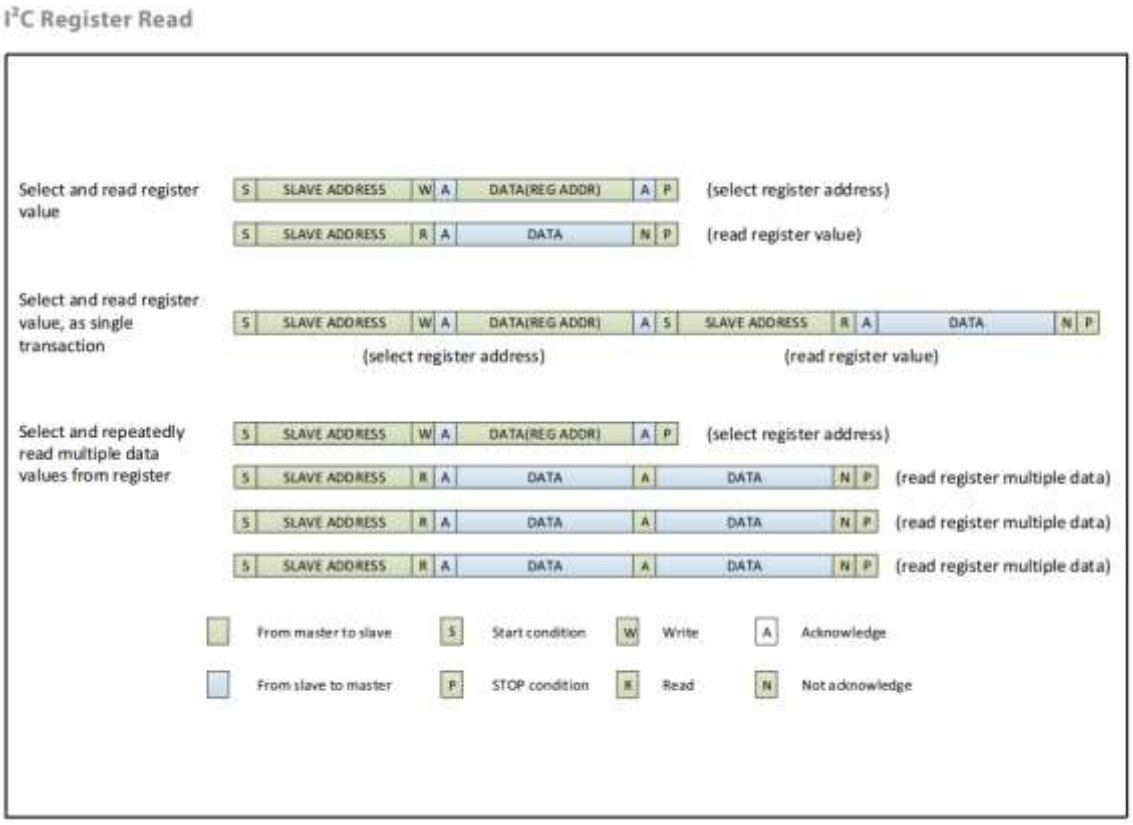


En el primer caso, únicamente se colocará el puntero sobre ese registro. De esa forma en caso de querer leer posteriormente lo de ese registro únicamente haría falta realizar una solicitud.

En el segundo y tercer caso se sigue la misma estructura. Primero se llama al esclavo con la dirección, luego se envía la dirección del registro, y a continuación los datos a escribir byte a byte.

Para el caso de la lectura la estructura es muy similar:

Imagen 37: CCS811 - Lectura Registros



En este caso se enviará la dirección del esclavo, y una vez seleccionado el registro a leer, cada lectura que se haga de dicho registro contendrá un byte de información.

Para entender el funcionamiento interno se deberá entender primero los registros y la estructura interna del módulo. Primero se observará el mapa de registros de la aplicación.

Tabla 19: CCS811 - Direcciones de registros

Address	Register	R/W	Size	Description
0x00	STATUS	R	1 byte	Status register
0x01	MEAS_MODE	R/W	1 byte	Measurement mode and conditions register
0x02	ALG_RESULT_DATA	R	up to 8 bytes	Algorithm result. The most significant 2 bytes contain a ppm estimate of the equivalent CO_2 (eCO_2) level, and the next two bytes contain a ppb estimate of the total VOC level.
0x03	RAW_DATA	R	2 bytes	Raw ADC data values for resistance and current source used.
0x05	ENV_DATA	W	4 bytes	Temperature and Humidity data can be written to enable compensation
0x06	NTC	R	4 bytes	Provides the voltage across the reference resistor and the voltage across the NTC resistor – from which the ambient temperature can be determined.
0x10	THRESHOLDS	W	5 bytes	Thresholds for operation when interrupts are only generated when eCO2 ppm crosses a threshold
0x11	BASELINE	R/W	2 bytes	The encoded current baseline value can be read. A previously saved encoded baseline can be written.
0x20	HW_ID	R	1 byte	Hardware ID. The value is 0x81
0x21	HW Version	R	1 byte	Hardware Version. The value is 0x1X
0x23	FW_Boot_Version	R	2 bytes	Firmware Boot Version. The first 2 bytes contain the firmware version number for the boot code.
0x24	FW_App_Version	R	2 bytes	Firmware Application Version. The first 2 bytes contain the firmware version number for the application code
0xE0	ERROR_ID	R	1 byte	Error ID. When the status register reports an error its source is located in this register
0xFF	SW_RESET	W	4 bytes	If the correct 4 bytes (0x11 0xE5 0x72 0x8A) are written to this register in a single sequence the device will reset and return to BOOT mode.

De este mapa se hará uso de únicamente de 5 que tienen interés en el proyecto:

Tabla 20: CCS811 - Registro de estado

Status Register

7	6	5	4	3	2	1	0
FW_MODE	-		APP_VALID	DATA_READY	-		ERROR

El registro de estado será de solo lectura, y, entre otras cosas, nos dará información de si se ha producido un error y si tenemos disponible un nuevo dato para ser leído.

Tabla 21: CCS811 - Registro Modo de medida

Measure Mode Register

7	6	5	4	3	2	1	0
-	DRIVE_MODE			INTERRUPT	THRESH	-	

Con registro de modo de medición se podrá ajustar la velocidad de lectura del módulo. Con esos tres pines será posible ajustar 4 modos o simplemente detenerlo.

Tabla 22: CCS811 - Medición

DRIVE_MODE	000: Mode 0 – Idle (Measurements are disabled in this mode) 001: Mode 1 – Constant power mode, IAQ measurement every second 010: Mode 2 – Pulse heating mode IAQ measurement every 10 seconds 011: Mode 3 – Low power pulse heating mode IAQ measurement every 60 seconds 100: Mode 4 – Constant power mode, sensor measurement every 250ms 1xx: Reserved modes (For future use) In mode 4, the ALG_RESULT_DATA is not updated, only RAW_DATA; the processing must be done on the host system. A new sample is placed in ALG_RESULT_DATA and RAW_DATA registers and the DATA_READY bit in the STATUS register is set at the defined measurement interval.

Tabla 23: CCS811 - Registro de resultados

Algorithm Results Register Byte Order

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6 & 7
eCO_2 High Byte	eCO_2 Low Byte	TVOC High Byte	TVOC Low Byte	STATUS	ERROR_ID	See RAW_DATA

En este registro se centrará la atención en los primeros 4 bytes, ya que serán los que muestren el valor del sensor. Realizando 4 lecturas a este sensor obtendremos en los dos primeros bytes las ppm de CO_2 y en los siguientes dos bytes las ppb de COV. Ambos valores vendrán dados en decimal.

Tabla 24: CCS811 - Registro de errores

Bit	ERROR_CODE	Description
0	WRITE_REG_INVALID	The CCS811 received an I^2C write request addressed to this station but with invalid register address ID
1	READ_REG_INVALID	The CCS811 received an I^2C read request to a mailbox ID that is invalid
2	MEASMODE_INVALID	The CCS811 received an I^2C request to write an unsupported mode to MEAS_MODE
3	MAX_RESISTANCE	The sensor resistance measurement has reached or exceeded the maximum range
4	HEATER_FAULT	The Heater current in the CCS811 is not in range
5	HEATER_SUPPLY	The Heater voltage is not being applied correctly
6	-	Reserved for Future Use
7	-	Reserved for Future Use

El registro de fallos tendrá especial interés en la programación inicial para comprobar que todo se está realizando correctamente.

A parte del mapa anteriormente mostrado, existe en el datasheet un segundo mapa para la configuración del Firmware. El único registro que se usará será un registro encargado de pasar del modo boot al modo aplicación. El modo boot solo será útil si queremos reprogramar el sensor, por lo que se pasará a modo aplicación para poder leer/escribir en los registros y poder realizar la lectura del sensor.

0xF4	APP_START	W	-	Application start. Used to transition the CCS811 state from boot to application mode, a write with no data is required. Before performing a write to APP_START the Status register should be accessed to check if there is a valid application present.

Para inicial la aplicación únicamente será necesario llamar a este registro sin la necesidad de escribir nada en él.

Programación

Para la parte de programación se harán diferentes funciones para mantener de forma estructurada el código.

Para iniciar el sensor se realizará una llamada al registro de APP-START (0xF4), con lo que pasará del estado carga al estado ejecución del programa. Y a continuación, se arrancará el sensor a una velocidad de lectura fijada mediante una variable #define.

Se ha realizado una función que comprueba que se dispone de un dato nuevo para ser leído. El dato del registro ALG_RESULT_DATA se limpia automáticamente cada vez que se lee, y el bit DATA_READY se colocará a 0. Cuando se obtenga un nuevo dato (fijado la velocidad mediante el registro de MEASURE MODE) el bit de DATA_READY se pondrá a uno, asegurándonos así de no leer un dato erróneo.

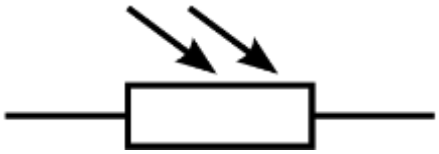
En esta función se realizará un bucle mediante un while hasta que el dato esté disponible para ser leído. En la variable dato únicamente se guardará el bit ubicado en la posición deseada del registro de estado.

La función de lectura será la más complicada ya que obtendremos los datos separados en dos bytes, y además ambos valores estarán en el mismo registro. Es por esto que se solicitaran 4 bytes, los dos primeros correspondientes al valor de CO2 y los dos siguientes al valor de COV. El primer registro serán los bits más significativos, por lo que habrá que guardar ese byte en un word de 16 bits, desplazarlo 8 posiciones a la izquierda y guardar a continuación los 8 bits de menor nivel del dato.

6.10. Sensor Luminosidad – LDR

Para poder controlar la hora a la que amanece y anochece, así como si ha salido un día nublado o soleado, se va a hacer uso de un sensor de luminosidad LDR. Estos sensores, tal y como indica su nombre: Light-dependent resistor, son componentes electrónicos que varían su resistencia dependiendo de la cantidad de luz que incide sobre ellos.

Imagen 38: LDR



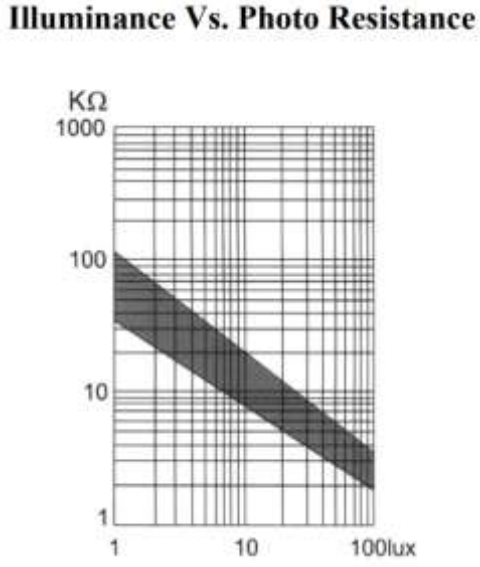
Estos componentes, también llamados fotorresistores, disponen de dos patillas, y están formados por un semiconductor de alta resistencia, el cual puede alcanzar valores de varios megaohmios en lugares oscuros y valores de pocos ohmios cuando están bajo una gran cantidad de luz. Esto se debe a que cuando no está expuesto a radiaciones luminosas, los electrones que lo forman están firmemente unidos, en cambio, cuando incide sobre ellos luz, está energía libera los electrones, permitiendo la conducción.

Los fotorresistores tienen gran cantidad de aplicaciones en la electrónica, ya que permite automatizar circuitos según la luminosidad, o la presencia o ausencia de luz. Esto permite no solo detectar las condiciones ambientales, sino que también separar o comunicar mediante luz dos circuitos separados eléctricamente.

Para este proyecto se va a usar un fotorresistor GL5528, que según su hoja de características es un LDR de gran sensibilidad y rápida respuesta. Este modelo, al igual que muchos otros, para un valor de luminosidad de 0 lux, presenta una resistencia cercana a 1MΩ, y aguanta valores de tensión de hasta 150V.

Si se observa la hoja de características del componente, se obtiene una gráfica que relaciona la resistencia con el valor en lux de la luminosidad. De esta gráfica se podrá obtener y aproximar el valor de luminosidad ambiente, y con ello deducir si el día es soleado, nublado o lluvioso.

Imagen 39: LDR - Relación Luminosidad/Resistencia



Se puede ver en la gráfica que la relación entre luminosidad y resistencia no es del todo precisa, ya que para valores de 1 lux puede variar entre 40 y 100kΩ. Pero esta precisión es más que suficiente para el objetivo de este sensor dentro del proyecto.

Dentro de la gráfica el valor que más acotado está es para 100lux, que se puede aproximar a 2k5 ohmios, y va a ser con este valor con el que se haga los cálculos. Haciendo uso de la regla de tres:

$$\frac{100\ lux}{x\ lux} = \frac{y\ \Omega}{2500\ \Omega} \rightarrow lux = \frac{250000}{\Omega} \rightarrow \Omega = \frac{250000}{lux}$$

Para poder acotar los valores de tensión dependiendo del día que haga, se va a obtener de internet una tabla que relacione la luminosidad con las condiciones atmosféricas.

Convirtiendo esos valores en resistencia del LDR en cada una de las características:

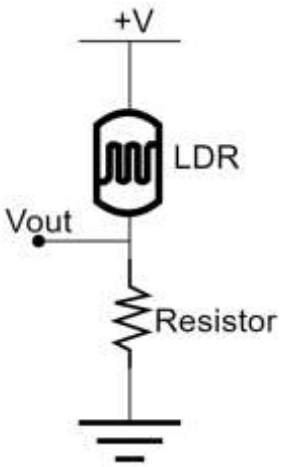
Tabla 25: LDR - Relación dia/luminosidad

	ILUMINANCIA	RESISTENCIA
LUZ DIURNA BRILLANTE	2100 lux	2 Ω
DIA CLARO EN EL OCASO	400 lux	625 Ω
DIA NUBLADO	10000 lux	25 Ω
NUBLADO EN EL OCASO	40 lux	6250 Ω
CLARO DE LUNA	<1 lux	>250 kΩ

Circuito

Para acondicionar la señal se va a optar por realizar un divisor de tensión, conectando la fotorresistencia a 5 voltios y la resistencia que se colocará en serie a tierra tal y como viene en la imagen a continuación.

Imagen 40: LDR - Circuito



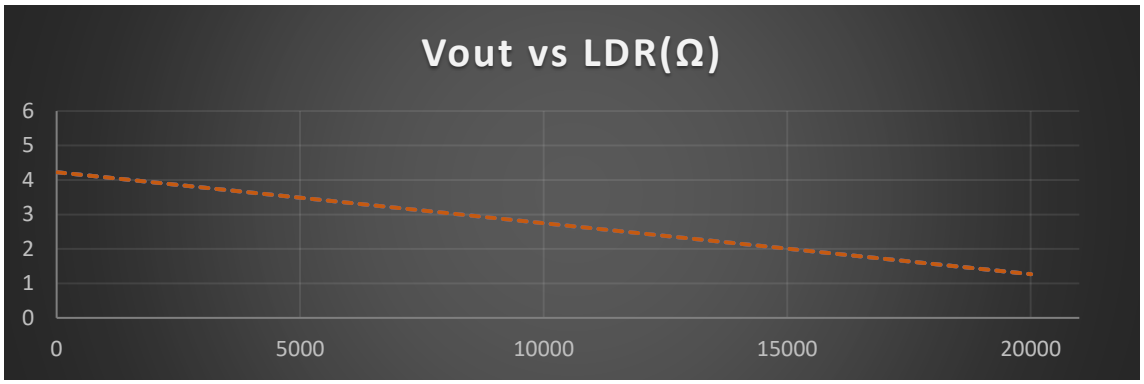
De esta manera, será posible calcular mediante el valor de Vout la resistencia del LDR, siendo los ohmios del Resistor un valor conocido:

$$V_{OUT} = \frac{Vcc * Resistor}{Resistor + LDR} \rightarrow LDR = \frac{Vcc * Resistor}{V_{OUT}} - Resistor$$

Al haber gran cantidad de valores que puede tomar el fotorresistor, se va a elegir un valor del Resistor cercano al rango de valores que se va a medir, de tal forma que para valores cercanos los valores leídos sean cercanamente lineales, mientras que para valores alejados (mucha oscuridad o mucha claridad) apenas se obtendrá variación en la tensión.

Se va a elegir el rango útil de valores entre 1 y 20kΩ, por lo que si el valor del Resistor es de la mitad (10kΩ), los valores leídos en la salida son:

Gráfica 4: LDR - Relación Tensión de lectura/resistencia



Programa

Para realizar la función de la medida del LDR, se va a implementar la función que relaciona el valor de la LDR con la tensión leída. Una vez se obtienen los ohmios del fotorresistor, mediante una regla de tres se relacionará la resistencia con la intensidad lumínica.

Esos valores de lumen se almacenarán y se enviarán, y será el propio monitor quien calcule las condiciones atmosféricas según la intensidad leída y los parámetros recogidos por los otros sensores.

Pruebas de funcionamiento

Para comprobar que la primera aproximación que se ha realizado se ajusta a las condiciones ambientales, se van a realizar varias pruebas a lo largo de una semana.

Tras analizar los valores se ha observado que dentro de unos márgenes de error se puede calcular con bastante precisión dependiendo de la hora del día si los valores corresponden a un día nublado o soleado. Siendo capaces de detectar sin ningún error la hora de amanecer y anochecer.

Tabla 26: LDR - Valores leídos

	RESISTENCIA LEÍDA	LUX EQUIVALENTES
LUZ DIURNA BRILLANTE	130 Ω	1900 lux
DIA CLARO EN EL OCASO	250 Ω	1000 lux
DIA NUBLADO	400 Ω	600 lux
NUBLADO EN EL OCASO	780 Ω	320 lux
CLARO DE LUNA	1.07 MΩ	0.2 lux

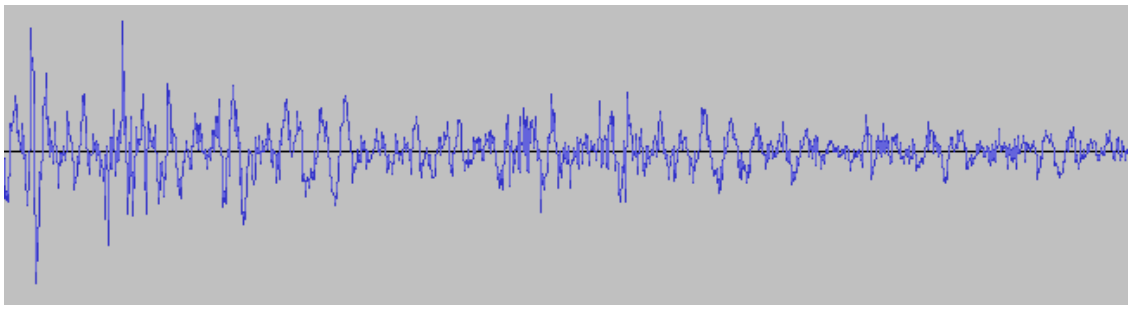
Como prueba final se ha comprobado la definición de un Lux. Sabiendo que un lux es la intensidad de luz que proporciona una vela a un metro de distancia, se han realizado medidas en una habitación completamente a oscuras, obteniendo valores entre 0,9 y 1 lux.

6.11. Sensor Acústico – MAX9814

Para detectar el ruido ambiente y la contaminación acústica tanto durante el día como durante la noche, se hará uso de un pequeño micrófono que se conectará a una de las entradas analógicas del Arduino.

Para entender el funcionamiento de este módulo se debería tener un ligero conocimiento sobre el sonido. El sonido hace referencia a la propagación de ondas mecánicas mediante un fluido o medio elástico. Estas ondas son formadas por la vibración de un cuerpo, y se transportan por el medio. El oído humano en su mayoría es capaz de percibir frecuencias entre 20Hz y 20kHz, por lo que este módulo al ser capaz de percibir dentro de este rango será de gran utilidad para detectar frecuencias audibles por el humano.

Imagen 41: Sonido



Al tratarse de una onda, se puede entender que tiene un estado de equilibrio, donde el sonido sería 0, y esas ondas producidas oscilarían respecto a ese punto medio. Este fenómeno se puede ver en el módulo a usar, ya que en un espacio sin ruido la tensión de salida tendrá un valor de tensión Vcc/2, y todas las vibraciones harán oscilar la salida entre los 0 y la tensión de alimentación (Vcc).

Conexión

Para conectar el módulo al Arduino y configurarlo a las necesidades del proyecto, se consultará la hoja de características del componente. En estas hojas se podrá encontrar tanto el cableado como la alimentación necesaria para el correcto funcionamiento.

Al poder alimentarse con 5V, se usará uno de los pines digitales del Arduino, con el fin de poder apagar y encender el módulo cuando se requiera.

Si se observa la tabla de pines del datasheet, dará una pequeña explicación de la funcionalidad de cada una de las conexiones.

Imagen 42: Sonido - Pines

Pin Description

PIN TDFN	NAME	FUNCTION
1	CT	Timing Capacitor Connection. Connect a capacitor to CT to control the Attack and Release times of the AGC.
2	$\overline{SHDN}$	Active-Low Shutdown Control
3	CG	Amplifier DC Offset Adjust. Connect a 2.2µF capacitor to GND to ensure zero offset at the output.
4, 11	N.C.	No Connection. Connect to GND.
5	V_{DD}	Power Supply. Bypass to GND with a 1µF capacitor.
6	MICOUT	Amplifier Output
7	GND	Ground
8	MICIN	Microphone Noninverting Input
9	A/R	Tri-Level Attack and Release Ratio Select. Controls the ratio of attack time to release time for the AGC circuit. A/R = GND: Attack/Release Ratio is 1:500 A/R = V_{DD}: Attack/Release Ratio is 1:2000 A/R = Unconnected: Attack/Release Ratio is 1:4000
10	GAIN	Tri-Level Amplifier Gain Control. GAIN = V_{DD}, gain set to 40dB. GAIN = GND, gain set to 50dB. GAIN = Unconnected, uncompressed gain set to 60dB.
12	BIAS	Amplifier Bias. Bypass to GND with a 0.47µF capacitor.
13	MICBIAS	Microphone Bias Output
14	TH	AGC Threshold Control. TH voltage sets gain control threshold. Connect TH to MICBIAS to disable the AGC.
—	EP	Exposed Pad. Connect the TDFN EP to GND.

Estos pines hacer referencia a todas las conexiones disponibles en el integrado MAX9814. Al usar un módulo con la mayoría de conexiones realizadas, únicamente se tendrá acceso a los pines de alimentación, lectura, ganancia y al ratio “Attack and Release”.

Como se ha comentado antes, se alimentará a 5V mediante un pin digital controlable. Al tratarse de un módulo para detectar sonido a larga distancia, se usará la mayor ganancia disponible, por lo que ese pin se dejará sin conectar. Para el “Attack and Release Ratio”, también se optará por dejarlo sin conectar para tener un parámetro de 1:4000.

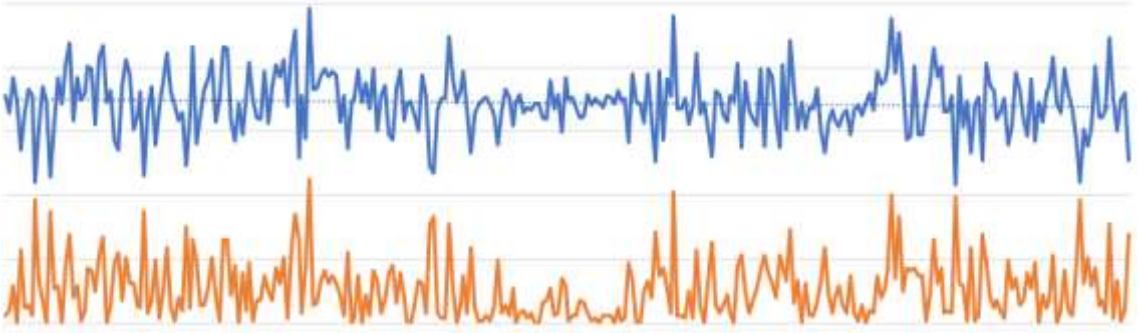
Programación

Para la realización del código se tendrá en cuenta lo anteriormente nombrado. Además, para facilitar la transmisión y guardado de los datos, se manejarán valores positivos. Para ello, se realizará un código que convierta el valor en absoluto. Para ello se ajustará la escala para que los resultados den entre 0 y 100, siendo 0 nada de ruido, y 100 el valor para el cual el micrófono se satura.

Para la prueba se recogerán valores cada 10 milisegundos, el código completo está disponible en el anexo.

Los datos recogidos se guardarán en un Excel y la gráfica obtenida sería la siguiente:

Gráfica 5: Prueba de sonido



Siendo el grafico azul los valores obtenidos por el micrófono, y el naranja los valores convertidos, se puede apreciar que el sonido en realidad oscila respecto a un valor medio (la línea de puntos). En cambios los datos convertidos serán únicamente positivos y facilitarán el cálculo y estudio.

Para el programa definitivo, se ajustará los valores en un rango de 0 a 100 respecto a los valores reales de dB. Haciendo varias pruebas se obtiene que para la ganancia elegida se satura a 80dB, por lo que el 100% corresponderá con 80dB. La segunda medida que se realizará será a ruido ambiente. En esta prueba, a 35dB se obtiene un 10% de la salida. Obtenidos esos dos puntos, e idealizando el sensor, obtendríamos la recta que relaciona los dB con la tensión.

$$y = mx + n$$

Siendo m la pendiente:

$$m = \frac{\Delta y}{\Delta x} = \frac{80 - 35}{100 - 10} = \frac{45}{90} = \frac{1}{2}$$

Y n el punto donde la recta corta al eje Y, (x=0 , y=30).

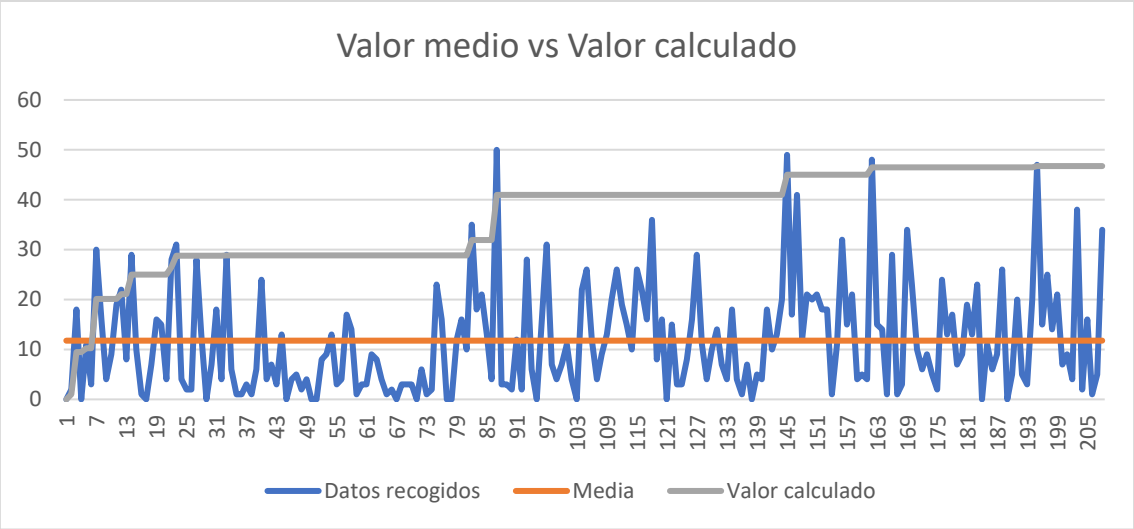
Se obtiene la recta:

$$y = {}^{1}/_{2}\, x + 30$$

Para realizar una toma de datos que se ajuste más a la sensación acústica y no sea puramente matemático, se ha decidido tomar recoger únicamente los valores que sean superiores a la media de los datos anteriormente recogidos. Esta decisión se ha tomado, ya que haciendo la media de todos los valores recogidos el resultado quedaba bastante por debajo de la percepción humana a lo que realmente había.

Se va a realizar en Excel un pequeño modelo con los valores obtenidos en la muestra, y se comprobará que efectivamente el valor obtenido es más coherente con la percepción.

Gráfica 6: Sonido - Valor calculado



En esta gráfica se puede observar que mientras el valor medio es simplemente un valor matemático, al valor calculado tiene en cuenta sobre todo los picos de valores, amortiguándolos un poco.

6.12. Antena de Radiofrecuencia

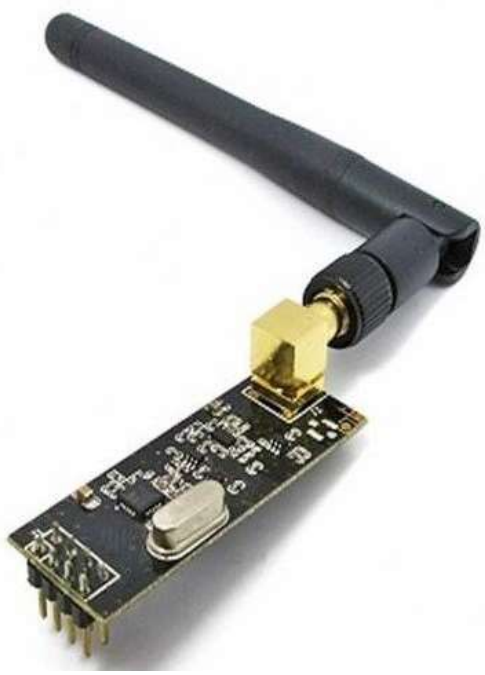
Durante el proyecto se va a realizar envío de comunicación no cableada, ya que la comunicación entre los dos módulos sería inviable si uno se encuentra en el exterior y el otro a unos metros y en el interior. Es por esto que se va a usar la radiofrecuencia como el modo de transmisión de información mediante ondas de aire.

La radiofrecuencia es un término que se refiere a la parte menos energética dentro del espectro electromagnético y corresponde a los valores situados entre los 3Hz y los 300GHz. Estas ondas electromagnéticas puedes ser generadas por una antena tras aplicarle una corriente alterna a la frecuencia que se desea transmitir.

Para este proyecto se va a hacer uso de una antena de radio que nos permita enviar y recibir información mediante su protocolo de comunicación correspondiente.

La antena a usar será el modelo nRF24L01, que permitirá transmitir la información con un alcance cercano a los 800m, dependiendo de las condiciones ambientales. Además, la comunicación con este módulo se puede realizar mediante SPI, el cual es compatible con el microcontrolador que se va a usar.

Imagen 43: Antena RF



Conexión y alimentación

Del datasheet se pueden obtener todos los parámetros de alimentación y conexión. Para el caso de la conexión, al usarse protocolo SPI será necesario el uso de los pines digitales 11, 12 y 13 del Arduino, además de dos pines más: un pin digital para CE y otro para CSN. Esos pines harán la función de inicio de comunicación siendo uno para avisar los casos en el que el Arduino sea el master y el otro para cuando sea la antena la que realice la función de master.

Imagen 44: Antena RF - pinout

Para el tema de la alimentación, la antena funciona a 3.3V. Esto es un problema para realizar una conexión directa, ya que el pin de 3.3V del Arduino no tiene gran suministro de corriente, por lo que puede que no produzca potencia suficiente para alimentar el módulo. Es por esto que se va a usar un regulador de tensión ASM1117, con el cual será posible alimentar con los 5V del Arduino (el cual dispone de mayor potencia de salida) y alimentar la antena con 3.3V.

El Regulador ASM1117 dispone de 3 pines:

- GND, el cual tiene que ser común a ambos lados del regulador
- IN, que tiene que tener un valor entre 4.5 y 7V
- OUT, en la salida se obtendrá una tensión constante de 3.3V

Imagen 45: Antena RF – Regulador AMS1117

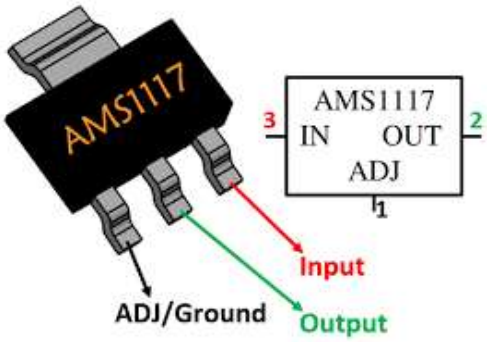


Imagen 46: Antena RF - Circuito Regulador

Librería RF24

Con el fin de evitar complicaciones con los protocolos y configuración del módulo, se va a hacer uso de una librería compatible con todas las antenas de radio de 2.4GHz.

Esta librería está disponible en el administrador de librerías propio del IDE, por lo que únicamente incluyéndolo al inicio del programa se agregarán las funciones propias de la librería al proyecto.

Entre otras disponibles, en este proyecto se van a usar unas funciones en concreto que ayudarán a inicializar, enviar y recibir información.

- Radio.begin( ): inicializa la antena, para ello:
    - Arranca el módulo con unos parámetros en concreto
    - Hace limpieza registros y los inicializa
    - Borra la información almacenada en las memorias FIFO tanto de transmisión como de recepción
    - Deshabilita el autoACK. El auto acknowledge es una función que disponen algunas antenas donde se puede detectar si la receptora ha obtenido los datos. En esta antena no es posible realizar esa comprobación por lo que será necesario deshabilitarlo
    - Por último, se reinicia el modulo

  La configuración previa puede ser modificada una vez inicializado el módulo.

- Radio.setRetrieves(,): contiene dos parámetros que indica cuantas veces se va a repetir la trama enviada y el tiempo de espera entre trama y trama.
  Para este proyecto al no ser le velocidad una prioridad, se ajustará para que realice 15 repeticiones y cada 400ms, el máximo valor para cada uno de los parámetros.
- Radio.setPALevel( ): fija la potencia de la antena, a mayor potencia mayor alcance.
  Se va afijar la máxima potencia, a fin de evitar perdida de información, además el consumo no será un problema, ya que únicamente usará esa potencia máxima durante un corto momento
- Radio.setDataRate( ): fija la velocidad de transmisión de datos, a menor velocidad mayor alcance y menor probabilidad de perdida de datos.
  Tal y como se ha comentado, la velocidad no es primordial, por lo que se ajustará a la mínima velocidad: 250kbps
- Radio.openWritingPipe( ): fija el canal de transmisión, se pueden configurar distintos canales para realizar conversaciones cruzadas
  Al disponer de únicamente dos módulos y un único sentido de envió, se fijará un único canal de transmisión
- Radio.startListening( ): configura la antena en modo escucha
- Radio.stopListening( ): configura la antena en modo escritura
- Radio.write(,): esta función se encargará de enviar el paquete de datos mediante RF. La función dispone de dos parámetros, el primero será el dato a enviar, y el segundo el tamaño de dicho paquete. Para ajustar el tamaño del archivo se usará la función sizeof() incluida por defecto en el Arduino

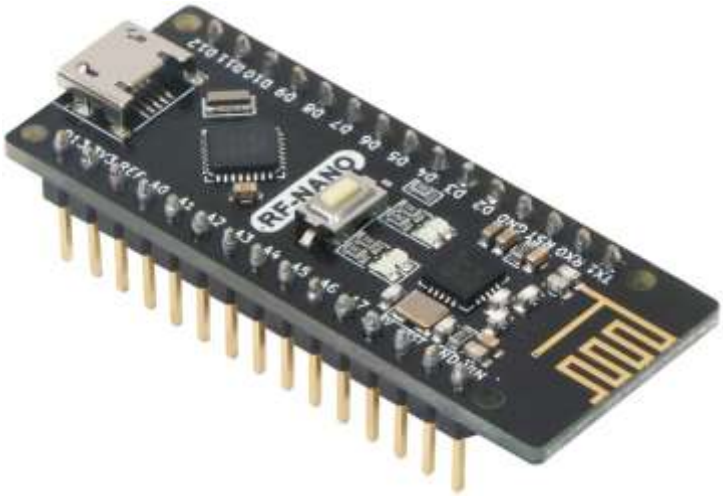
- Radio.available(): esta función no requiere de parámetros, y devolverá un 1 cuando se haya recibido un archivo de datos y sea posible leer el dato.
  Al recibir un nuevo archivo, se almacena en la memoria temporal FIFO de la antena, y se queda a la espera de ser leída desde el controlador. Una vez que se ha leído esa información se borra.
- Radio.read(,): permite leer la información recibida por RF. Contiene dos parámetros al igual que la función de escritura (write), donde el primer parámetro será la variable donde se va a almacenar los datos y el segundo parámetro el tamaño de dicha variable. Es importante que las variables de envío y recepción tengan la misma estructura para evitar problemas a la hora de guardar esos datos.

Arduino RF-nano

Este Arduino combina la parte computacional y la antena en un único controlador, permitiendo reducir tamaño y evitando módulos extra.

Se va a hacer uso de este de micro en el receptor, a fin de realizar la recepción de información mediante un único µC capaz de recibir analizar y posteriormente enviar de forma cableada la información al controlador encargado de almacenar dicha información.

Imagen 47: Arduino RF-nano



6.13. Potencia

Una de las características más importantes que dispondrá la estación, es que será capaz de funcionar sin la necesidad de estar conectada a la corriente eléctrica externa. Para ello, dispondrá de unas placas solares, batería, y todos los componentes y circuitos necesarios para su correcto funcionamiento.

La idea de esta parte del proyecto será la implementación de un sistema de alimentación a base de energía renovable, que permita convertir la estación meteorológica en un módulo completamente independiente, y con capacidad de ser portable y autosuficiente.

La parte de potencia se puede separar en 3 grupos:

- Placas solares
- Batería
- Circuitos de unión

6.13.1. Placa Solar

Para las placas solares se han elegido uno de los modelos más comunes para proyectos que requieran poca potencia, y resistencia ante clima y manipulación, las YD-107X61.

Estas placas serán de silicio monocristalino, y a diferencia de otros modelos, están formadas por un único cristal de silicio. Este cristal a la hora de su fabricación es controlado para que su crecimiento sea unidireccional, consiguiendo un buen alineamiento y evitando que se formen otros conjuntos de cristales.

Las YD-107X61, además de venir reforzadas para su manipulación y montaje, suministran 1W de potencia en condiciones óptimas. Es por esto que se convierten es una opción de lo más asequible para el proyecto.

Imagen 48: Placas solares

Para obtener un valor de potencia que se ajuste más a las necesidades del proyecto, se va a hacer uso de 10 placas conectadas en serie/paralelo para obtener 10V y 1A. Para conseguir esos 10W se colocarán las placas en formato 5p2s (5 paralelo y 2 serie).

Se ha decidido usar la matriz de placas previamente nombrada, ya que posterior a las placas se usará un convertidor DC-DC con salida a 5V. Este convertidor será un regulador de tensión pensado para placas solares con una salida máxima de 5V y 2A, y una tensión de entrada que puede ir desde los 6 a los 35V. Conectando a la entrada de este regulador 2 placas en serie, se alcanzarán los 6V de tensión de entrada para menores intensidades de sol, y aunque la corriente se vea bastante limitada por la ausencia de sol, será capaz de cargar de forma lenta en días no soleados.

Imagen 49: Regulador Placas solares

Pero el estudio de las placas solares no acaba con la elección de los componentes, ya que su orientación será clave para aprovechar al máximo el sol.

Para calcular tanto la orientación como el ángulo, hará falta analizar el movimiento del sol durante todas las estaciones. Para la orientación, al ser el trayecto del sol de Este a Oeste, y Bizkaia encontrarse en el hemisferio norte, las placas tienen que estar orientadas hacia el sur. En cambio, para el ángulo, se hará un estudio tanto en verano como en invierno, ya que serían los momentos clave.

Para verano se cogerá el día más largo del año: 21 de junio. Sabiendo que el eje de la tierra está inclinado unos 23º respecto al plano que forma el movimiento de la tierra respecto al sol, y que Bizkaia se encuentra a 23º norte respecto al ecuador; utilizando una simple formula, se podrá calcular el ángulo de las placas para que estén lo más perpendiculares posible al sol.

$$En\ verano \rightarrow 90º - (43º - 23º) = 70º$$

Con esta fórmula se obtiene que, durante esta época, la altura del sol respecto al horizonte será de 70º en el momento de mayor elevación. Por lo que las placas perpendicularmente colocadas irán a 20º.

Para invierno se realizará el mismo calculo, utilizando el día más corto: 21 de diciembre. En este caso la formula a usar será:

$$En\ invierno \rightarrow 90º - (43º + 23º) = 24º$$

24º será la altura máxima que alcance ese día de invierno, las placas deberían llevar un ángulo de 66º para situarse perpendicularmente.

Teniendo en cuenta que durante el invierno la intensidad es menor y además durante menos horas, se elegirá una inclinación entre las dos estaciones, pero procurando que favorezca al invierno, por tener más dificultades de generación.

Teniendo en cuenta todo lo anterior, se colocarán las placas orientadas al sur e inclinadas 45º.

6.13.2. Batería

Una vez obtenemos una tensión constante de 5V, es el momento de almacenar esa energía, para ser capaces de hacer funcionar el circuito en ausencia de sol.

La batería es la parte más delicada en el estudio, ya que además de ser un compuesto químico inflamable y que requiere protección extra, es importante tener en cuenta todo lo conectado y su consumo para evitar que el circuito se apague.

Se va a hacer uso de unas baterías de ion de litio de la marca LiitoKala, exactamente el modelo HG2 18650. Estas baterías tienen una capacidad de 3000mAh a una tensión nominal de 3.7V, por lo que cada pila será de 10,8Wh. Al igual que el resto de baterías de este tipo, su tensión máxima será de 4.2V y su tensión de corte 2.5V.

Imagen 50: Batería

Para realizar la protección de la batería, se colocará un BMS (battery management system / Sistema de gestión de baterías). Este circuito será el encargado de evitar sobretensiones y sobre corrientes, y de igual manera, mantendrá la batería entres sus valores máximos y mínimos de tensión. El BMS tendrá capacidad para una pila en serie, ya que, en el caso de querer aumentar la capacidad de batería, se aumentarán pilas en paralelo, no afectando esto a la tensión en bornes del BMS.

Imagen 51: BMS

Se ha elegido este BMS, el cual tiene una protección superior de 4,25V, una inferior de 2,55V y una protección de sobretensión de 3 Amperios tanto para carga como para descarga.

Pruebas de descarga

Para conocer la capacidad real de la batería, se van a hacer una prueba de descarga. Para ello se realizará un circuito con la batería, una resistencia de descarga y un Mosfet como protección. Además, se utilizarán dos leds para indicar de forma lumínica el correcto funcionamiento, un interruptor manual para detener y arrancarlo, y una SD para almacenar los datos.

Imagen 52: Circuito descarga

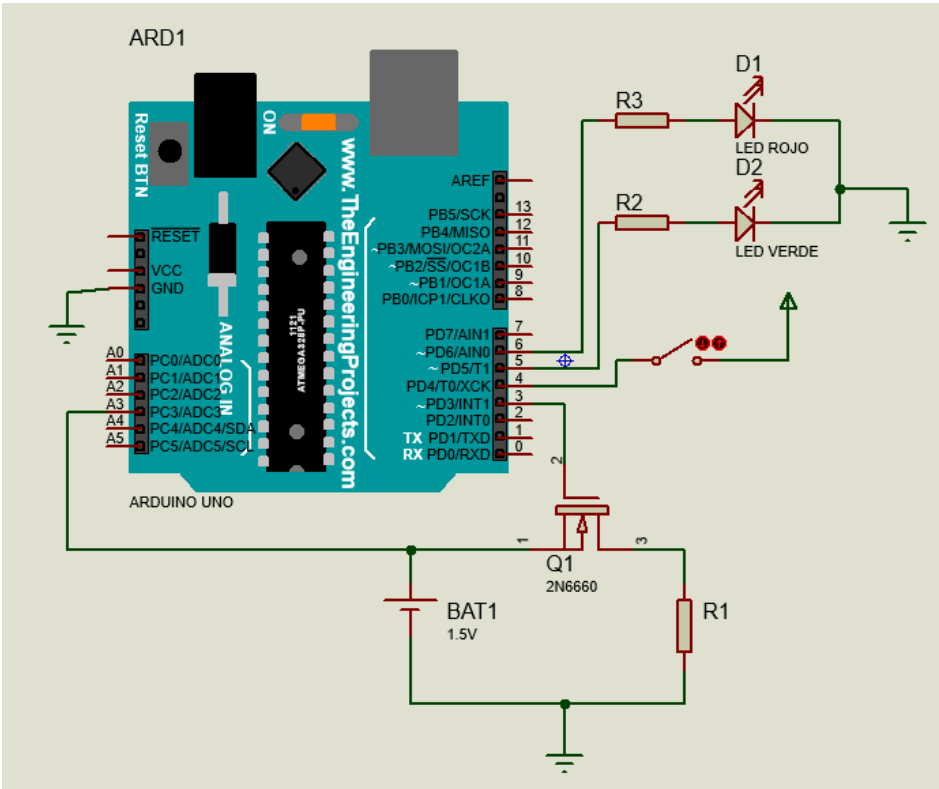
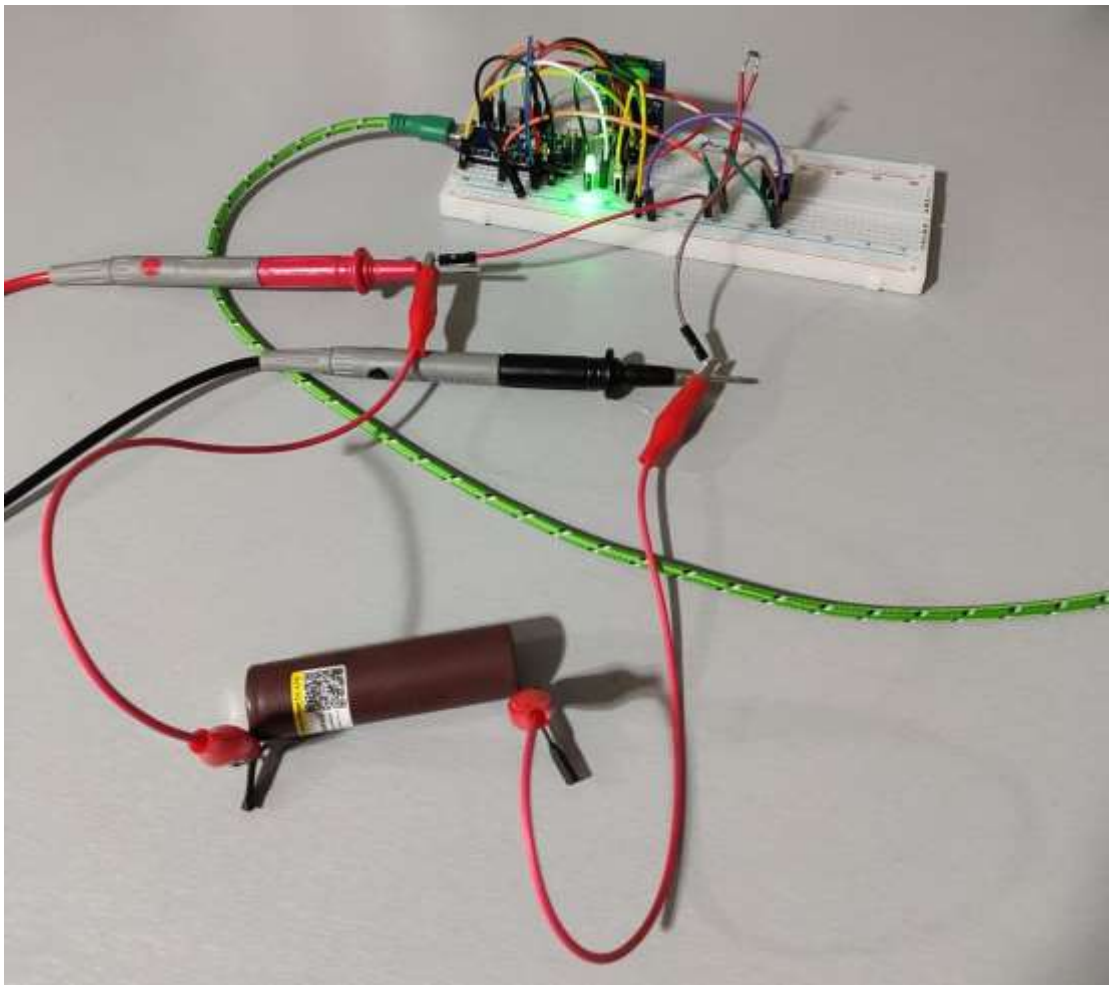


Imagen 53: Circuito descarga montado

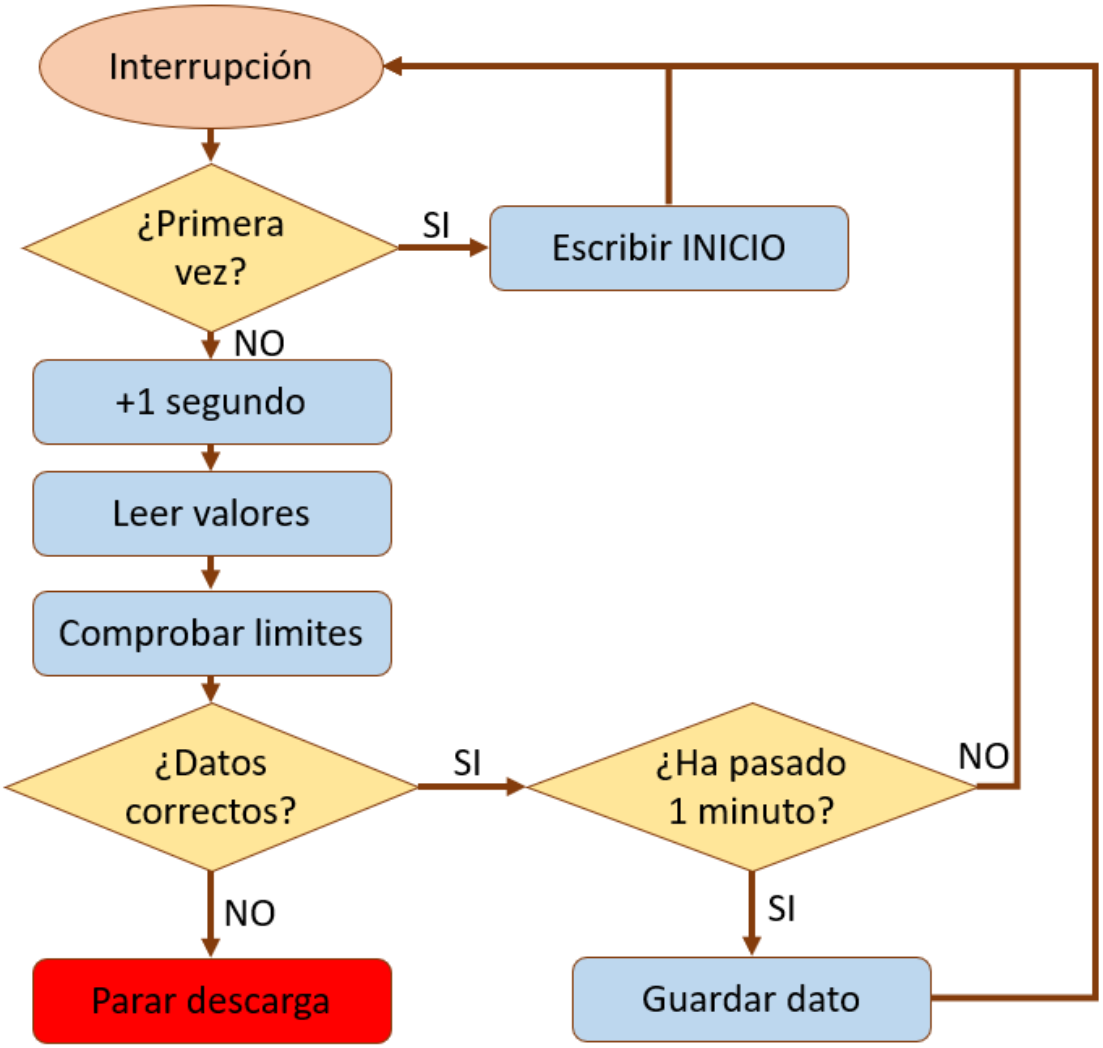


La prueba se ha realizado con una resistencia de 30Ω, consiguiendo una corriente de descarga de unos 120mA.

Para el programa se utilizó el timer interno para realizar una interrupción cada segundo, y de esta forma tener controlado en el tiempo la ejecución del programa.

Una vez se ha configurado la interrupción cada segundo, se realizará una función que será ejecutada cada vez que salte la interrupción.

Imagen 54: Diagrama de flujo prueba descarga



Como se desea obtener un dato cada minuto, cada interrupción se añadirá un segundo al programa y volverá a esperar hasta la próxima interrupción.

Una vez ha pasado el minuto completo, se realiza la toma de datos. Para ello, primero se mide la tensión en la celda y se hace la comprobación de seguridad para controlar que no se salga de los valores de tensión predefinidos. Si los datos son correctos y ha pasado el minuto se guardarán ese dato.

La descarga duró un total de 1336 minutos (22h y 15 minutos), y los datos recogidos en la SD, se abrieron con Excel para poder tratarlos.

Se hicieron varias columnas para ver la variación a lo largo del tiempo:

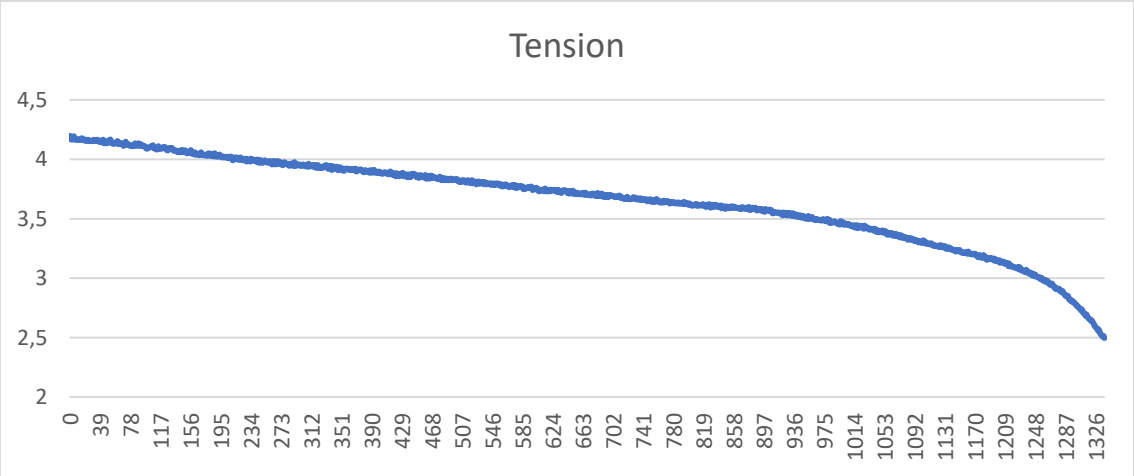
Tabla 27: Descarga batería

	A	B	C	D	E	F
1	Tension	Minuto	Corriente mA	Corriente acum mA	Potencia (mW)	Potencia Acum (mW)
2	4,197	0	139,9	0	587,1603	0
3	4,168	1	138,933333	2,315555556	579,0741333	9,651235556
4	4,168	2	138,933333	4,631111111	579,0741333	19,30247111
5	4,168	3	138,933333	6,946666667	579,0741333	28,95370667
6	4,168	4	138,933333	9,262222222	579,0741333	38,60494222
7	4,193	5	139,766667	11,59166667	586,0416333	48,37230278
8	4,168	6	138,933333	13,90722222	579,0741333	58,02353833
9	4,173	7	139,1	16,22555556	580,4643	67,69794333
10	4,163	8	138,766667	18,53833333	577,6856333	77,32603722
11	4,168	9	138,933333	20,85388889	579,0741333	86,97727278
12	4,163	10	138,766667	23,16666667	577,6856333	96,60536667
13	4,163	11	138,766667	25,47944444	577,6856333	106,2334606
14	4,163	12	138,766667	27,79222222	577,6856333	115,8615544
15	4,173	13	139,1	30,11055556	580,4643	125,5359594
16	4,168	14	138,933333	32,42611111	579,0741333	135,187195
17	4,178	15	139,266667	34,74722222	581,8561333	144,8847972

Una vez estructurados todos los datos, se procede a realizar varias graficas para poder observar de forma más visual los resultados obtenidos.

La gráfica de la tensión quedo:

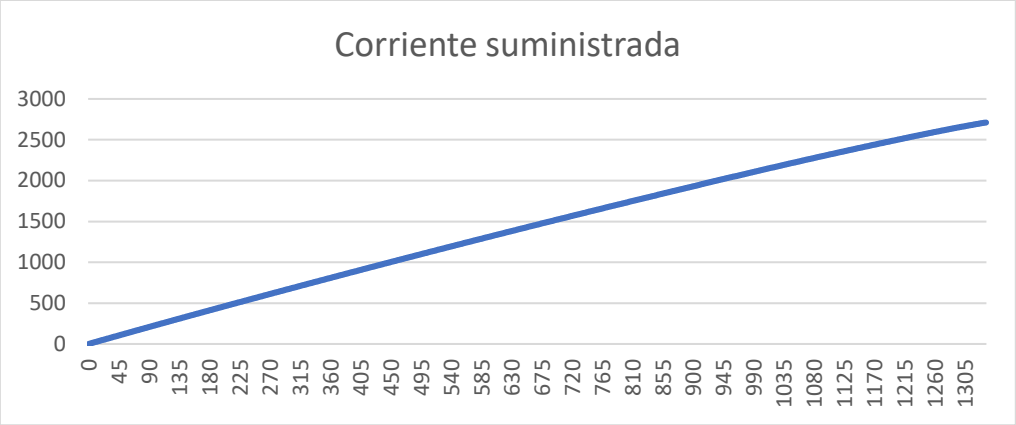
Gráfica 7: Prueba descarga - Tensión



Se puede apreciar como para un valor de 3V la tensión cae bruscamente, mientras que cercano a los 3.7V (la tensión nominal) se mantiene a una pendiente constante.

La gráfica de la corriente:

Gráfica 8: Prueba descarga - Corriente



En esta grafica se muestra la corriente acumulada suministrada por la celda. Se puede apreciar que, al mantener una resistencia de descarga constante, la corriente va a ser lineal, alcanzando un valor máximo cuando la batería suministre la capacidad total almacenada.

Los valores obtenidos durante la descarga son los siguientes:

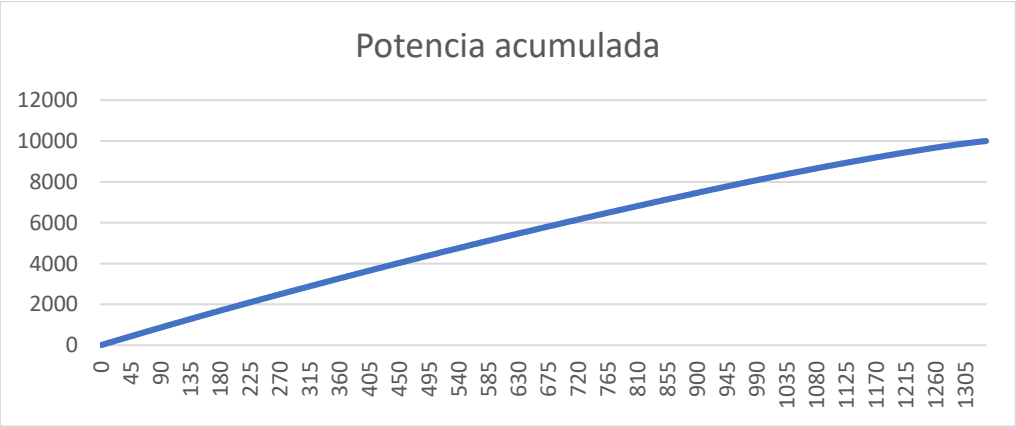
Tabla 28: Prueba descarga - Corriente

Corriente media (mA)	Tiempo total (min)	Tiempo en horas	Capacidad bateria (mAh)
121,7315645	1336,1	22,27	2710,759056

En esta tabla se muestra entre otros, los valores de la corriente media, que es de 121.73mA, muy cercana a la estimada al inicio de la prueba. Y el valor más importante de la tabla es la capacidad de la batería, que es de 2710mAh, y no 3000mAh, tal y como se muestra en las características.

Respecto a la potencia:

Gráfica 9: Prueba descarga - Potencia



En esta grafica se muestra la multiplicación de las dos graficas anteriores (tensión y corriente). Este es seguramente el valor de mayor utilidad de la prueba, ya que será con lo que se realicen los cálculos.

Para ver más claramente los valores se mostrarán en una tabla:

Tabla 29: Prueba descarga - Potencia

Potencia media	Tiempo total (min)	Tiempo en horas	Potencia suministrada (W)
448,9521054	1336,1	22,27	9,997415134

En la tabla se muestra que la potencia de cada batería es de cercana a 10W, por lo que sabiendo lo que va a consumir el circuito se puede calcular la duración de cada batería.

Como extra durante la prueba, se puede observar en el código que cuando la tensión ha caído a un valor, se recogen varios datos de forma rápida. Esto se hizo para observar la recuperación de la batería.

Gráfica 10: Prueba descarga - Recuperación

La recuperación de la batería en el proyecto que se va a realizar no va ser de gran utilidad, ya que este fenómeno se presenta cuando se corta la descarga de forma brusca. Aun no siendo un valor que no se vaya a tener en cuenta para los cálculos, se ha decidido dedicarle un momento, ya que hoy en día con el auge de los vehículos eléctricos, es algo a tener mucho en cuenta. Esto se muestra sobre todo durante la conducción de vehículos de carreras, ya que la agresividad durante la conducción produce picos de corriente que provocan caídas agresivas en la tensión que no son reales, ya que una vez cortada la descarga la tensión vuelve a su valor. Esto tiene gran importancia a la hora de controlar la tensión de corte de la batería, ya que una caída de tensión puede provocar un corte por protección, el cual puede afectar negativamente a la eficiencia de la batería.

Con todos estos datos se podrá calcular cuanta capacidad es necesaria, y por lo tanto cuantas baterías será necesario colocar en paralelo.

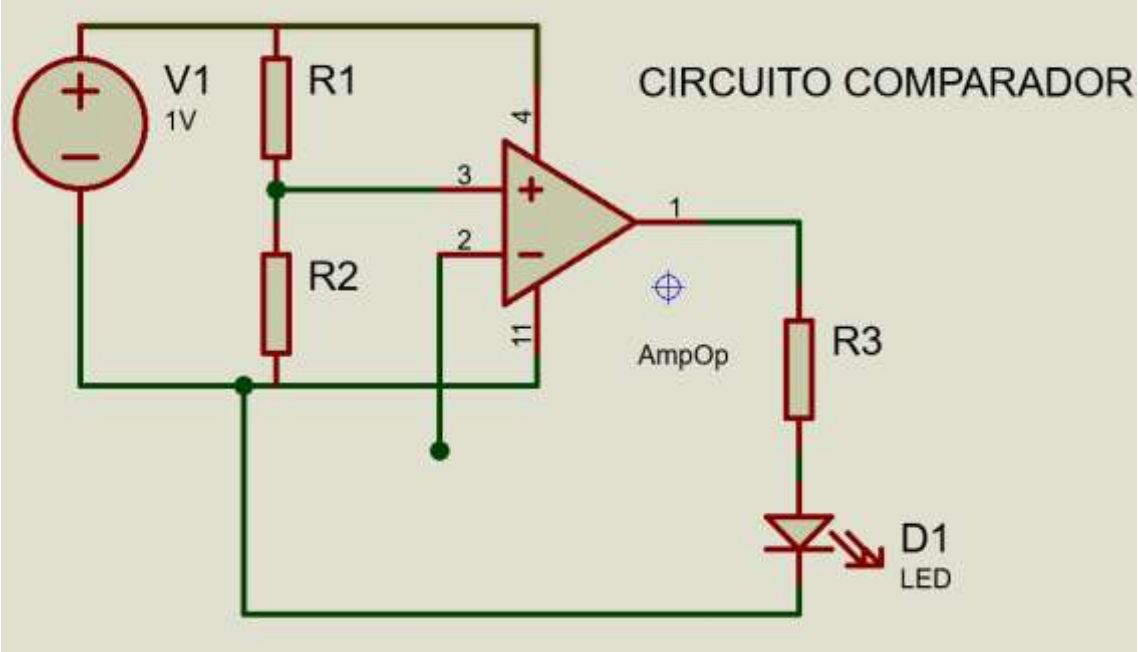
Si cada pila es capaz de suministrar cerca de 10W de potencia, en condiciones ideales, las placas solares cargarían la batería en 1h. En realidad, no todos los días se cumplirán están condiciones, por lo que en el peor de los casos se pondrá que tardará 2h. Si se cuenta con que se tienen unas 8h de luz útil diarias, con 4 pilas no se desaprovecharía la energía, ya que la batería se cargaría antes de empezar a anochecer. Con 4 pilas obtendríamos 40W, por lo que, si aproximamos el consumo con los datos de las hojas de características de todos los componentes, encendido se obtendría un consumo de 100mA a 5V (0.5Wh) cuando está encendido. En modo sleep, la mayoría de los módulos seguirán alimentados, pero el Arduino bajará considerablemente el consumo, gastando unos 50mA (0,25Wh). Como el circuito únicamente se despierta unos segundos cada 5 minutos, en el peor de los casos estará 6 minutos funcionando a la hora, 0.25*0.9+0.50*0.1=0.275W/h.

Cada pila de 10W durarían aproximadamente 36h, por lo que colocando 5 pilas suministrarían 9 días de autonomía.

Para poder notificar que el módulo exterior se está quedando sin batería se va a realizar un circuito mediante un amplificador operacional de tal forma que funcione como comparador. De esta manera cuando la tensión de la batería sea menor a un valor dado, encenderá una pequeña luz que indicará que el módulo puede llegar a apagarse si no es capaz de generar energía mediante las placas solares.

Para el circuito se hará uso de un amplificador operacional básico que sea capaz de funcionar a 5V y mediante la conexión mostrada en la imagen inferior se conseguirá encender una luz cuando la batería baje de los 3V.

Imagen 55: Circuito batería baja



Se ajustarán las resistencias para obtener 3V en la patilla no inversora, por lo que se usara para R1 una resistencia de 6k8 y en R2 una de 10K.

$$V = \frac{5 * 10k}{10k + 6.8k} = 2.97V$$

6.13.3. Circuitos de unión

Hasta ahora hemos hecho uso de unas placas solares para la obtencion de la energía; un circuito reductor de tension para obtener 5V de salida de las placas; unas baterias para almacenar y suministrar la energía; y un BMS para proteger esas baterias. Pero se requiere de un circuito extra, uno que sea capaz de unir lo anteriormente nombrado.

Este circuito será el encargado de transformar los 5V de las placas solares y cargar la batería con tension nominal de 3,7V. También será necesario que a la vezque carga las baterias suministre tensión al circuito. Además, será necesario que alimente el circuito cuando no se obtenga energía de las placas solares, y sea necesario obtenerla de la bateria.

Imagen 56: Regulador de tensión Buck-boost

Para cumplir con estor requisitos, se va a hacer uso de un convertidor Buck-Boost (reductor-elevador) que permita cargar las baterías, protegerlas, y hacer de nexo entre todos los componentes.

7. Programa

Durante el desarrollo del programa principal, se va a hacer uso de los fragmentos de código realizados mediante el testeo de los diferentes componentes, y se va a analizar la forma de interconexionar todos ellos.

Para realizar la conexión de todos los componentes, de tal forma que todos sean alimentados y tengan adjudicados al menos un pin de lectura, se debe conocer el pinout del Arduino a usar. Tal y como se comentó, se va a hacer uso del Arduino nano y dispone de 7 pines analógicos (2 de ellos para I2C), 13 pines digitales (4 de ellos para SPI) y 5 pines de alimentación/tierra.

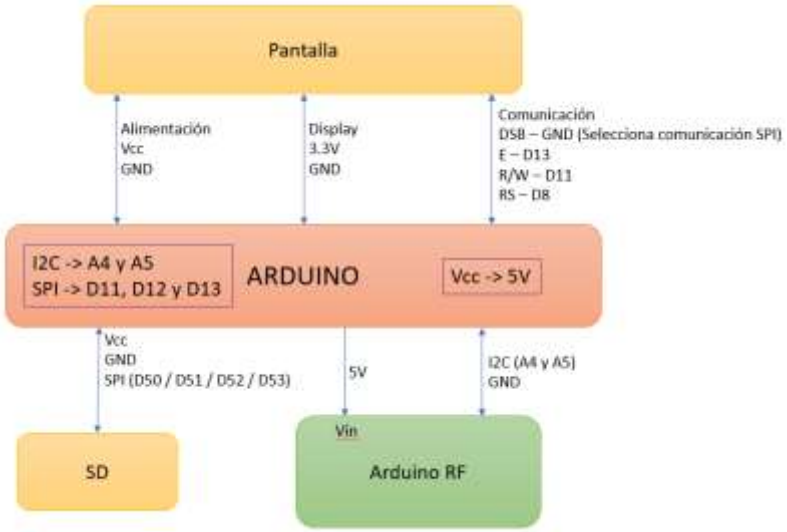
Agrupando todos los pines necesarios para alimentar a los sensores, se podrá realizar la alimentación tal y como viene en la imagen:

Imagen 57: Pinout Exterior

Esta será la estructura usada para la alimentación y lectura de los sensores, por lo que se realizará y se ajustará el código a partir del creado durante el testeo.

Para el caso del módulo interior, el cableado será ligeramente menor por la cantidad de componentes, pero es muy importante tener en cuenta que para la comunicación SPI se van a usar distintos pines a fin de evitar interferencias en la pantalla.

Imagen 58: Pinout Interior



7.1. Ahorro de energía

Al tratarse de una estación que necesariamente tiene que ser autosuficiente, y en todo momento recopilar lo que sucede en el exterior sin necesidad de alimentarse a una red externa, el ahorro de energía y la eficiencia energética es un tema a tener continuamente presente.

Es por esto que en este proyecto se va a hacer uso del Arduino nano, el controlador con menor consumo que proporciona las prestaciones necesarias para el funcionamiento correcto de la estación. Aun así, mantener el Arduino con todos sus módulos continuamente encendido no es nada eficiente, ya que se tomarán datos cada 5 minutos, y la ejecución del programa únicamente llevará unos segundos. Es por esto que entre todas las posibilidades que ofrece el Arduino para ahorro de energía se va a optar por apagar el módulo todo ese tiempo que no se esté ejecutando el programa.

En el modo Power-Down, tal y como se ve en la tabla, es el único modo que permite apagar tanto relojes como osciladores, consiguiendo un consumo 0.36mA.

Tabla 30: Diferentes modos Sleep

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/IO	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Para encender de nuevo el sistema se dispone de varias opciones, entre ellas la interrupción mediante pines de entrada. Será esta opción la que se use durante el proyecto, ya que para poder controlar correctamente el tiempo se va a encender el controlador mediante una interrupción creada en el Reloj. Este módulo, tal y como se ha estudiado anteriormente, dispone de la posibilidad de generar interrupciones o alarmas. Mediante la programación de estas alarmas podremos generar un flanco cambiante en la señal, la cual encenderá el sistema.

En el datasheet configuraremos el modo PowerDown. Para ello se irá al registro SMCR (Sleep Mode Control Register) y se habilitará el modo sleep (SE / bit 0).

Tabla 31: Sleep Mode

SMCR – Sleep Mode Control Register

The sleep mode control register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A continuación, en el selector de modo Sleep (SM) habrá que escribir la combinación de bits que apague el sistema en el modo deseado. En el caso de PowerDown será el 010.

Pero antes de apagar el sistema, habrá que configurar el modo de despertarlo. Para ello se configurará una interrupción la cual encenderá el sistema al cambiar el valor de una señal externa. Para ello se habilitará la interrupción mediante señal externa en el registro EIMSK (External Interrupt Mask Register).

Tabla 32: Mask Register

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A continuación, se configurará el modo de interrupción en el registro EICRA (External Interrupt Control Register A).

Tabla 33: Control Register

EICRA – External Interrupt Control Register A

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Como se va a usar el INT0 se configurarán los bits 1 y 0. Se desea que la interrupción se ejecute en el flanco descendiente, por lo que según la tabla habrá que escribir un 1 y un 0 en esos bits.

Tabla 34: Modos de interrupción

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Código

El código tendrá dos funciones distintas, una primera que configurará el apagado y lo ejecutará, y una segunda que realizará las acciones necesarias una vez se ejecuta la interrupción y se enciende el módulo.

Cabe destacar que al encender el módulo es importante deshabilitar tanto la interrupción como el modo sleep para evitar interrupciones o que el módulo se apague durante la ejecución del código.

Además, se va a hacer uso de la variable ENCENDIDO, la cual será de gran utilidad en el loop ( ) para tener constancia de que la interrupción se ha ejecutado.

7.2. Código Estación Meteorológica

En este apartado se va a desglosar y explicar el código realizado para el módulo exterior. Primero se explicará el funcionamiento en su conjunto, y posteriormente, se realizará un análisis más detallado de lo que realiza la estación meteorológica.

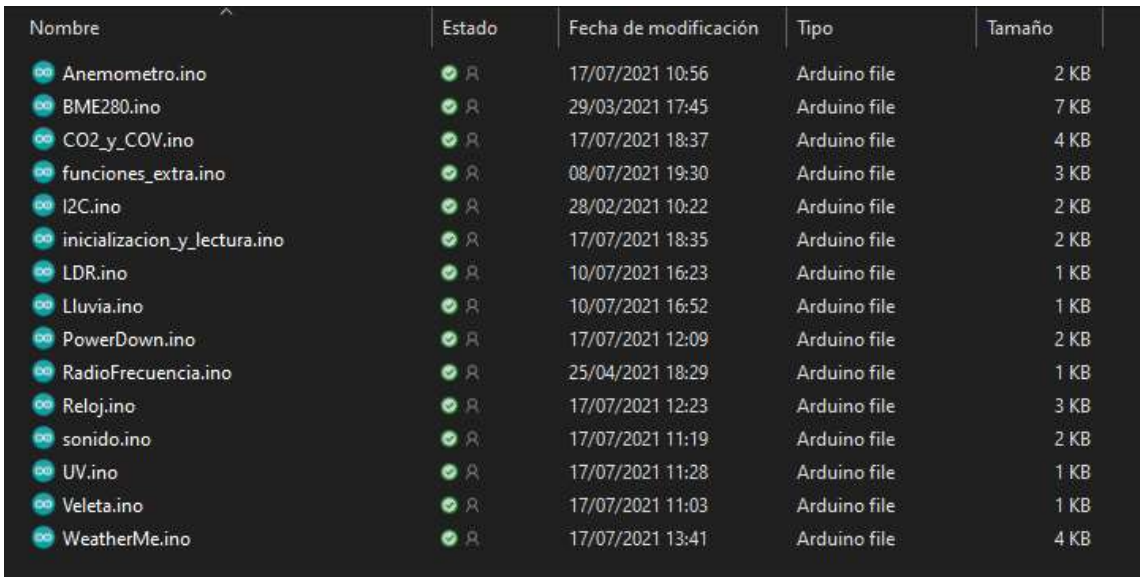
Tal y como se hizo mención en el diagrama de flujo inicial, el módulo exterior va a ejecutarse de forma cíclica marcado por interrupciones creadas por el reloj. Una vez ha saltado la interrupción, el microcontrolador arrancará y leerá los sensores, confirmando que los valores leídos son los correctos. Tras obtener los parámetros, los enviará y volverá a apagarse.

Para facilitar la explicación del código, se va a separar en apartados y en funciones las distintas acciones que tendrá que realizar el programa. Siendo el orden:

- Encenderse
- Encender sensores
- Leer parámetros
- Enviar parámetros
- Configurar alarma
- Apagarse

Esta separación no se hará únicamente para la explicación, el código se separará en distintos archivos conectados entre sí.

Imagen 59: Estructura código



Nombre	Estado	Fecha de modificación	Tipo	Tamaño
Anemometro.ino		17/07/2021 10:56	Arduino file	2 KB
BME280.ino		29/03/2021 17:45	Arduino file	7 KB
CO2_y_COV.ino		17/07/2021 18:37	Arduino file	4 KB
funciones_extra.ino		08/07/2021 19:30	Arduino file	3 KB
I2C.ino		28/02/2021 10:22	Arduino file	2 KB
inicializacion_y_lectura.ino		17/07/2021 18:35	Arduino file	2 KB
LDR.ino		10/07/2021 16:23	Arduino file	1 KB
Lluvia.ino		10/07/2021 16:52	Arduino file	1 KB
PowerDown.ino		17/07/2021 12:09	Arduino file	2 KB
RadioFrecuencia.ino		25/04/2021 18:29	Arduino file	1 KB
Reloj.ino		17/07/2021 12:23	Arduino file	3 KB
sonido.ino		17/07/2021 11:19	Arduino file	2 KB
UV.ino		17/07/2021 11:28	Arduino file	1 KB
Veleta.ino		17/07/2021 11:03	Arduino file	1 KB
WeatherMe.ino		17/07/2021 13:41	Arduino file	4 KB

En el Anexo está disponible el código completo usado, y en él se puede ver que al inicio del programa se dedicará un apartado para nombrar las variables globales, definir los valores (#define), añadir las librerías que se van a usar, y para ajustar los pines de alimentación y I/O.

7.2.1. Loop principal

El bucle principal del programa realizará las acciones que se ha nombrado anteriormente desde el encendido hasta el apagado. Durante este bucle se van a llamar una a una las funciones principales que se van a ir ejecutando en orden, y estas funciones llamadas realizarán sus respectivas acciones.

Este bucle únicamente se realizará una vez y tendrá la condición de que la variable Encendido esté a 1. Esta variable se colocará a 1 durante el encendido, y al apagarse se volverá a colocar a 0. De esta manera será posible asegurarse de que únicamente se ejecuta 1 vez el bucle y únicamente tras encenderse el módulo.

7.2.2. Encendido y Apagado

Tal y como se ha mencionado anteriormente, el encendido del µC se realizará tras recibir la señal desde el Reloj. Este reloj pondrá a 0 su pin de salida, y conectando esa salida al pin de entrada digital 2 será posible configurar una interrupción.

La función que arranca el módulo será una ISR (interrupt service routine / rutina de control de interrupciones) y se configurará antes del apagado.

La función de encendido, realizará las siguientes acciones en orden:

- Deshabilitará el modo sleep
- Se inhabilitará las interrupciones para evitar un reinicio durante la ejecución
- Encenderá una luz led que indicará que se ha encendido
- Colocará la variable Encendido a 1 para poder entrar al bucle

La función de apagado:

- Borrará los valores de las variables
- Apagará el led
- Colocará la variable a 0
- Habilitará las interrupciones
- Habilitará el modo sleep
- Se configurará el tipo de apagado
- Se ejecutará el apagado del módulo

7.2.3. Encender sensores

Durante el arranque de los sensores básicamente se alimentará los módulos que se han tenido desconectados durante el apagado y se realizarán las configuraciones necesarias en los módulos para su correcta lectura.

Durante el encendido de los componentes, no se alimentarán todos de golpe, ya que en un principio se alimentarán únicamente los que requieran configuración previa. Los módulos que solo sea necesario alimentar, se encenderán momentos antes de la lectura y a continuación se apagaran.

Los sensores que se van a encender son:

- BME280 – Sensor de Temperatura, Humedad y Presión
- CCS811 – Sensor CO2 y COV
- Antena RF

Para el sensor BME280 va a ser necesario una escritura previa en los registros de memoria configurando la lectura que se va a realizar. Pare ello se van a seguir los siguientes pasos:

1. Configurar los pines de comunicación
2. Esperar a que arranque, y en caso de ser necesario reiniciar
3. Esperar a que termina de medir
4. Se leen los registros de calibración
5. Se configura el sobremuestreo y se pone en modo Sleep

Para el sensor CCS811 los pasos a seguir son los siguientes:

1. Colocar el módulo en modo programa, ya que por defecto viene preparado para actualizar el software
2. Arrancar el sensor y configurarlo para una velocidad de lectura

La antena de radiofrecuencia para la inicialización requerirá:

1. Configurar velocidad de transmisión, potencia y tipo de onda
2. Seleccionar el canal de transmisión
3. Configurar la antena como transmisora

El resto de módulos se irán encendiendo y apagando sobre la marcha, ya que únicamente requieren de ser alimentados para funcionar al de pocos milisegundos.

7.2.4. Leer valores

Para la lectura de los parámetros se va a realizar una función que vaya leyendo uno a uno los parámetros.

Para seguir el mismo orden durante todo el programa, el orden de lectura va a ser:

- Sensor BME con los parámetros de temperatura, presión y humedad
- Velocidad y sentido del viento
- Intensidad de lluvia
- Intensidad de incidencia de luz UV
- Sonido de fondo
- Luminosidad leída con el fotorresistor
- Sensor CCS con las medidas de CO2 y COV

Para el desarrollo de estas funciones se ha hecho uso de los códigos previamente usados para cada uno de los componentes por separado, pero con algunas modificaciones de alimentación y de conexión a fin de evitar superposiciones entre los sensores.

7.2.5. Envío de parámetros

Para el envío de parámetros se hará uso de la antena RF. Esta antena ha sido encendida durante el encendido de componentes, por lo que, al llegar a este punto, la antena está correctamente configurada y preparada para hacer el envío.

La librería usada para la antena dispone de la función write, con la cual se puede enviar una estructura completa mediante radio frecuencia. La función write requiere de dos parámetros, el primer parámetro será el dato a enviar y el segundo el tamaño de este paquete de datos.

La variable que se va a enviar es un tipo Struct, por lo que bajo el nombre de una variable se puede acceder a diferentes subvariables.

Imagen 60: Formato struct

```
struct Struct{
  int FechaHora[6];   //dia de la semana, dia, mes, año, hora, minuto
  float Temperatura;
  float Presion;
  float Humedad;
  int Vel_viento;
  byte Sen_viento;
  int Lluvia;
  float UV;
  int Sonido;
  int Luminosidad;
  word CO2;
  word COV;
};

Struct WeatherMe;
```

La variable struct tiene una estructura interna que estamos definiendo al declarar la variable. Posteriormente, se crea la variable WeatherMe, que va a tener la estructura fijada previamente. De esta manera llamando a la variable WeatherMe.Temperatura, se accederá al float almacenado dentro de la estructura WeatherMe, dentro de la posición de temperatura.

Al enviar esta estructura como un único paquete de datos, el receptor deberá tener la misma estructura para no modificar los datos durante el guardado.

Está disponible en el anexo la función realizada para el envío.

7.2.6. Configurar alarma

Antes de proceder al apagado, se configurará la hora de próxima alarma sabiendo la hora actual.

Para ello se realizará una función llamada configurar reloj que realizará las acciones de configuración y de ajuste de la próxima alarma.

Durante la configuración de la alarma, se accederá a los registros de la alarma y se seleccionará para que la alarma se ejecute para un valor de minuto concreto.

Para el ajuste de la próxima alarma primero se limpiará el flag y a continuación se sumarán 5 minutos a la hora actual.

7.3. Código Modulo interior

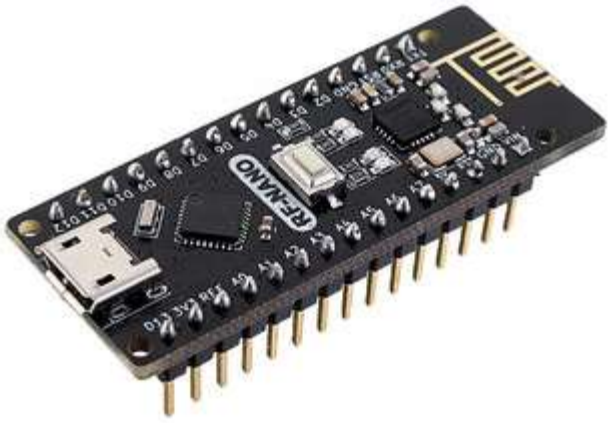
Tal y como se ha comentado con anterioridad, el módulo interior constará de dos controladores. El primero será el encargado de recoger y guardar de manera temporal la información recibida por Radiofrecuencia. El segundo, será el encargado de procesar esa información, guardarla y posteriormente mostrarla.

Estos dos controladores estarán continuamente conectados mediante comunicación serial, de esta manera, cada vez que un dato es obtenido, ambos controladores inician una comunicación y su posterior envío de información.

7.3.1. Receptor Radiofrecuencia

Para el microcontrolador encargado de obtener la información, se va a hacer uso de un Arduino nano, cuya característica principal es que tiene la antena integrada. De esta forma, el micro está optimizado para su correcto funcionamiento, y no será necesario el uso de un módulo externo.

Imagen 61: Arduino RF-nano



Al igual que los Arduinos convencionales, este modelo también dispone de comunicación I2C, y va a ser esta comunicación serie la que se usará para el envío de información.

Código principal

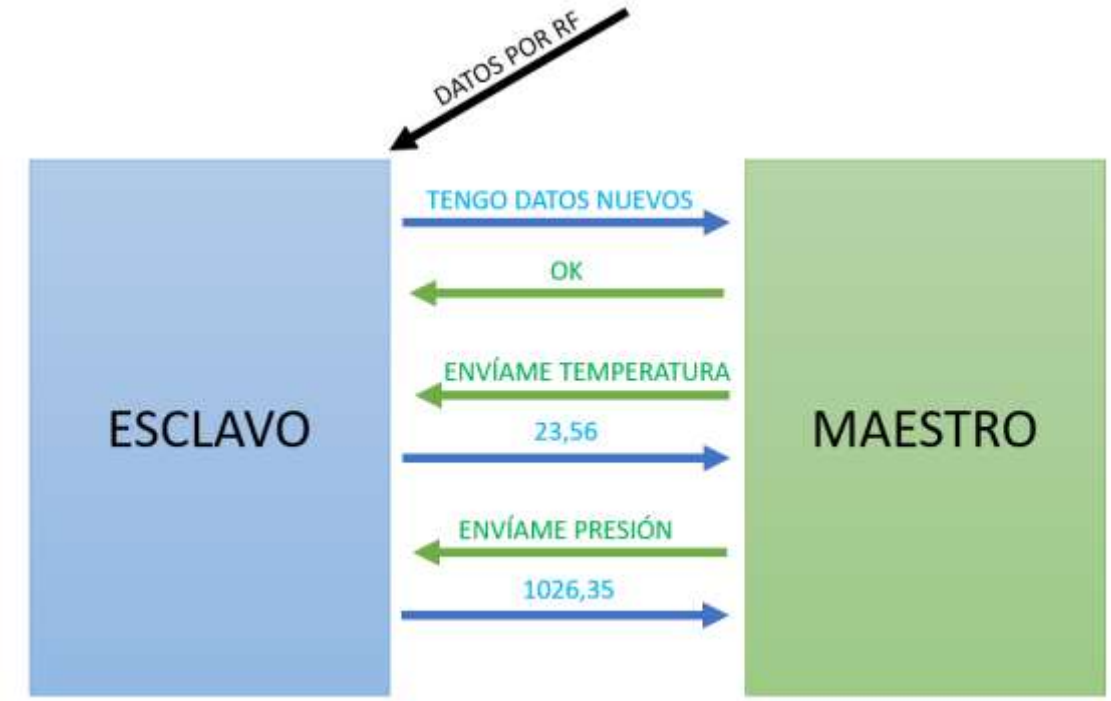
Para el funcionamiento del receptor se va a configurar tanto la antena como el hardware serie, por lo que para ello habrá que tener claro el flujo del programa.

El funcionamiento y procedimiento empleado para realizar la comunicación se puede resumir en dos sencillos pasos:

- Cuando se obtiene un nuevo dato -> El esclavo almacena los datos y avisa al maestro
- Cuando el maestro recibe el aviso -> Responde que está escuchando y solicita uno a uno los datos

El código completo del módulo está disponible en el Anexo. Y el proceso de ejecución que sigue se puede representar mediante este diagrama:

Imagen 62: Diagrama de flujo Esclavo/Maestro



Recepción por Radiofrecuencia

Para la recepción de los datos, se realizará un código similar al de envío, con la diferencia de que en el bucle principal se realizará continuamente la comprobación de la llegada de un nuevo dato.

Para ello se realizará una función: leer_radio(), que comprobará continuamente si hay nuevo dato disponible, y en caso de ser así se leerán los datos y se almacenarán en la variable *Estructura* creada. Código completo en el anexo.

Comunicación I2C

En este caso, al igual que se ha realizado en partes anteriores del proyecto, se va a hacer uso de la librería propia del Arduino (Wire.h). En el apartado Comunicación I2C en Arduino, se hizo mención a las funciones que se van a usar en este módulo.

Al módulo receptor se le dará el papel de esclavo, de tal forma que la pantalla realizará el papel de maestro y será el que controle la comunicación.

Para la comunicación entre los dos Arduinos, hay que tener presente las funciones *onReceive()* y *onRequest()*. Estas dos funciones crearán una interrupción en el programa, de tal forma que cuando se reciba o se solicite un dato saltará la función definida.

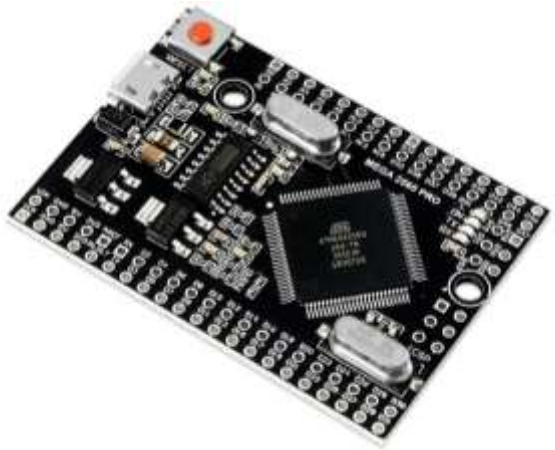
El código completo está disponible en el Anexo, y se muestra cada una de las funciones creadas, que se repetirán de forma cíclica cada vez que se obtenga un nuevo dato.

Cabe destacar que es el maestro quien solicita los datos uno a uno, de tal forma que manda un código concreto para cada variable, y es el esclavo quien interpreta el código y responde con el valor correspondiente.

7.3.2. Módulo pantalla y SD

Este módulo será el encargado de procesar todos los datos, guardarlos, y posteriormente mostrarlos. Al requerir de gran potencia de cómputo, será necesario hacer uso de un microcontrolador de más potencia, a fin de ser capaces de procesar con fluidez la información, y evitar bugs (error de software) a la hora de mostrarlos. Es por esto que se usará el Arduino Mega Pro, uno de los micros de mayor potencia dentro de la familia Arduino. Este controlador monta un procesador Atmega 2560, el cual, además de disponer de más potencia, dispone de mayor capacidad de memoria y cantidad de pines I/O.

Imagen 63: Arduino Mega Pro

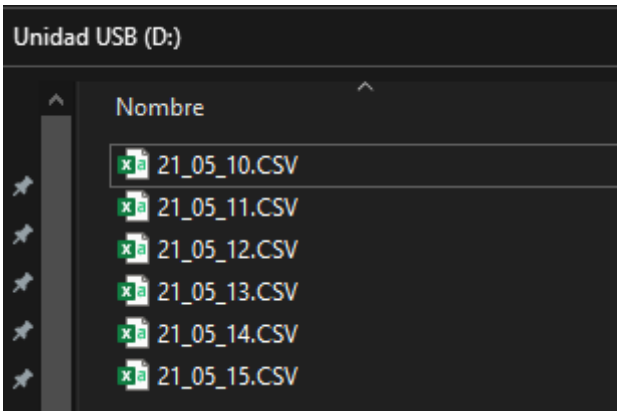


Tal y como se ha explicado en el apartado del Receptor de Radiofrecuencia, este Arduino hará el papel de maestro, y será quien controle la comunicación I2C con el receptor de RF. El código completo de esta comunicación viene explicado tanto en el apartado anterior como en el Anexo.

7.3.2.1. SD – Guardado de datos

Todos los datos recogidos se almacenarán en una tarjeta microSD, la cual puede ser extraída en cualquier momento del circuito y conectar al ordenador para recoger los datos obtenidos. Estos datos se guardarán en archivos separados diarios, que contendrán todos los parámetros separados por horas.

Imagen 64: Estructura SD



Tal y como se puede observar a la hora de conectar la tarjeta de memoria en el ordenador, cada día se crea un nuevo archivo *.CSV con el nombre año_mes_día.csv. En este archivo se guardarán todos los valores recogidos clasificados por la hora de lectura.

Escritura

Para el caso de los archivos diarios, se comenzará desde las 00:00 de ese día a recopilar datos, y se acabará a las 23.59 de ese mismo día. La recogida de datos se realizará cada 15 minutos, por lo que se recogerán 4 datos la hora, esto es, un total de 96 datos.

$$Datos\ diarios = 4\ datos/h * 24\ h = 96\ datos\ al\ día$$

Al separarse los datos por días, diariamente a las 00:00 se creará un nuevo archivo con el año mes y día actuales, y también creará la cabecera para tener perfectamente definida cada una de las columnas del archivo. Una vez creado el archivo junto a su estructura, se escribirán los primeros datos y se guardará el *.CSV a la espera de los nuevos datos.

Imagen 65: Estructura archivo

Dia	Hora	Minuto	Temperatura	Presion	Humedad	Velocidad del viento	Sentido del viento	Lluvia	Indice UV	Sonido	Luminosidad	CO2	COV
LUNES/10/11/21	13	30	19.32	1012.12	58.43	22	SUROESTE	NO LLUEVE	0.00	40	200	0	0
LUNES/10/11/21	13	45	19.56	1012.12	58.43	25	OESTE	NO LLUEVE	0.00	42	250	0	0
LUNES/10/11/21	14	0	19.38	1012.12	58.43	19	SUROESTE	NO LLUEVE	0.00	51	300	0	0
LUNES/10/11/21	14	15	19.42	1012.12	58.43	23	SUROESTE	NO LLUEVE	0.00	48	150	0	0
LUNES/10/11/21	14	30	19.51	1012.12	58.43	22	SUROESTE	NO LLUEVE	0.00	45	210	0	0

Para crear la estructura y la escritura correcta, se ha realizado un código que se ajuste a los requerimientos previamente fijados. El código se puede separar en 3 funciones principales que engloban todos los pasos.

- Iniciar y Detener la SD: estas funciones serán las encargadas de inicializar y detener la SD cuando se vaya a hacer uso de este módulo de memoria.
- Crear fichero: esta función será la encargada crear de forma diría los archivos de datos. Se encargará de realizar el archivo con el nuevo año_mes_dia y de escribir la cabecera.
- Guardar datos: esta tercera función será la encargada de escribir en los archivos correspondientes los nuevos datos. Está función se ejecutará de forma cíclica cada 15 minutos, y requiere de la previa ejecución de la función crear string, la cual recogerá los valores de todos los parámetros y los guardará en un único string (cadena de caracteres) delimitando los valores por comas.

7.3.2.2. Pantalla LCD

El módulo interior dispondrá de una pantalla donde se mostrarán de forma continua los valores actuales. La idea principal de este módulo será tener en todo momento acceso a los datos actuales sin necesidad de extraer la SD y leer los datos desde un ordenador o dispositivo externo.

Para cumplir este objetivo, se va a implementar un código en el Arduino Mega Pro, que sea capaz de procesar la información recibida y mostrarla.

La función que controlará la ejecución del programa estará en el loop principal y mostrará de forma cíclica los valores siempre y cuando no se estén recibiendo nuevos datos. Esta función realizará la comprobación de la recepción de nuevos datos y en caso de no ser así mostrará la información. Cuando se estén recibiendo nuevos datos, se los solicitará al receptor.

Aun siendo el tiempo de guardado cada 15 minutos, la pantalla tendrá un tiempo de refresco de 5 minutos, que es el tiempo que tarda la estación exterior en mandar los nuevos datos.

Para la muestra de los datos, se van a realizar 4 pantallas que mostrarán la distinta información:

- Salvapantallas: esta pantalla mostrará el nombre de la estación meteorológica. Esta pantalla se mostrará al estar recibiendo nuevos datos y al acabar la muestra de datos
- Pantalla uno: esta pantalla mostrará los valores de:
    - Temperatura
    - Presión
    - Humedad
    - Velocidad y sentido del viento
- Pantalla dos: esta mostrará:
    - Luminosidad
    - Sonido en dB
    - Índice de luz UV
    - La presencia o no de lluvia
- Pantalla tres: esta pantalla mostrará la contaminación atmosférica:
    - CO_2
    - COV

Para realizar la transición por las pantallas se va a realizar una función que vaya llamando a una a una las pantallas y muestre los correspondientes datos

Junto a estos datos se mostrará la fecha y hora de los datos: para ello se ha creado la función ahora, con la cual se muestra el día de la semana que es actualmente (Lunes, Martes…), el número del día del mes, el mes y el año. La fecha al igual que el resto de datos también se actualiza, pero al recibir un dato nuevo cada 5 minutos, la hora se va incrementando de 5 en 5 sin pasar por los valores intermedios.

8. Diseño

Una vez se ha probado el funcionamiento completo de la estación, llega el momento de realizar el diseño de cómo se va a realizar el montaje.

En este apartado se va a analizar las distintas opciones de montaje disponible y el modo correcto de ensamblar todos los módulos a fin de hacer un diseño discreto, funcional y compacto. Durante la parte de diseño se tendrá siempre presente el hecho de que la estación estará a la intemperie, teniendo que aguantar fuertes vientos y lluvia intensa.

Todos los componentes que se van a usar se podrían agrupar en tres grandes grupos:

- Los que requieren una ubicación y condiciones concretas
- Los que necesitan protección extra
- El resto de componentes

Teniendo en cuenta estos grupos, se comenzará diseñando todos los sensores que necesitan una ubicación concreta para desempeñar su función. Estos componentes son:

- Placas solares: necesitan luz directa
- Sensor UV y sensor luminosidad: necesitan luz directa y protección ante humedad
- Sensor de lluvia: estar ligeramente inclinado y en sitio elevado para detectar la lluvia
- Veleta y anemómetro: será necesario un lugar elevado con viento directo
- Sensor de humedad, presión y temperatura: estar abierto a la atmosfera, pero con protección ante lluvia y sol directo
- Sensor de CO2 y COV: mismas condiciones que el sensor de temperatura
- Arduino y Reloj: evitar sol directo y exceso de humedad

A continuación, se diseñará todas aquellas partes sensibles o más peligrosas de la estación, a fin de evitar riesgos. Este grupo lo forman todos los circuitos encargados de almacenar y transformar la energía. Este grupo en su conjunto tiene que ir completamente protegido ante humedad y temperatura, evitando posibles riesgos de sobrecalentamiento y daño por humedad.

Por último, se ubicarán todos aquellos módulos y sensores que no necesitan ni protección ni ubicación exacta, por lo que se diseñarán para que no estorben al resto de componentes. En este grupo están todos los componentes electrónicos encargados del acondicionamiento de la señal, que mientras tengan un mínimo de refrigeración para evitar calentamientos innecesarios y protección ante sol y lluvia directa no deberían dar mayores problemas.

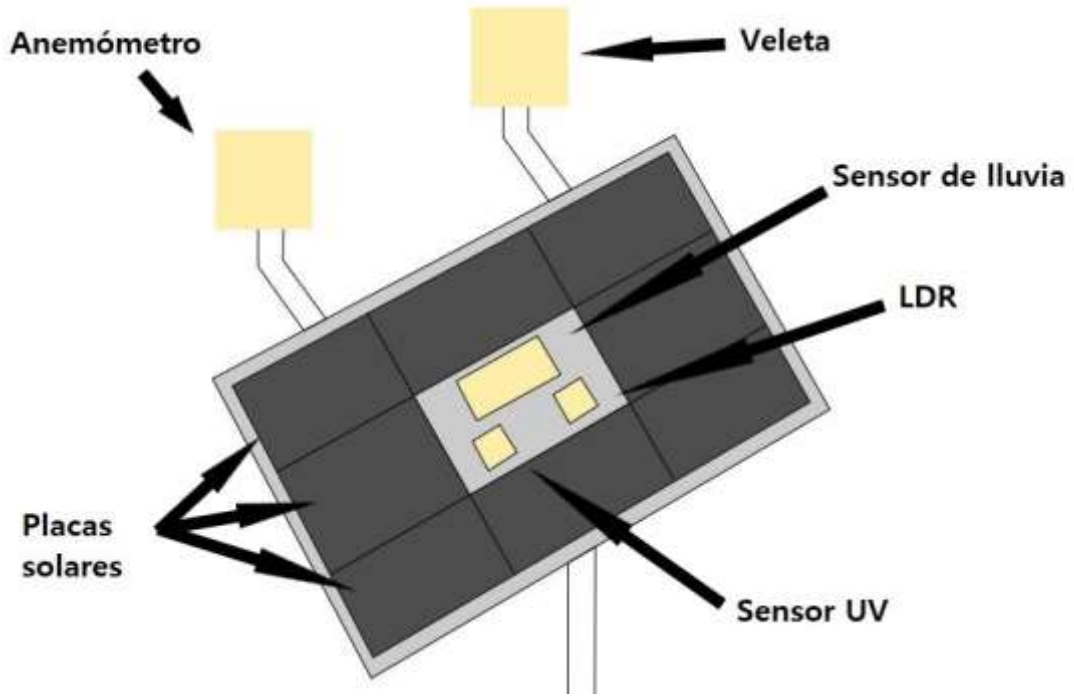
8.1. Parte superior

Se comenzará diseñando los componentes que necesitan luz directa y elevación, ya que necesitarán estar en la parte superior. La idea principal que se mantendrá durante esta parte de diseño será colocar tanto las placas solares como el sensor UV y el de luminosidad entre dos placas de metacrilato herméticamente unidas y con una inclinación de 45º, tal y como se decidió durante el estudio de la elevación del sol en distintas épocas del año. Sobre estas placas iría el sensor de lluvia, ya que requerirá elevación al igual que detectar la lluvia.

Teniendo en cuenta que las placas estarán inclinadas esos 45º mencionados, se va a realizar una estructura en forma de “Y” que sostendrá en su parte superior a los dos sensores de viento.

Se ha realizado un pequeño modelo con una primera idea de cómo se va a realizar el montaje de estos componentes.

Imagen 66: Diseño parte superior



Los tubos a utilizar para la sujeción serán de PVC, ya que además de ser duraderos a la intemperie, dispone de conectores tales como bifurcadores en T, y codos a 90 y 45 grados.

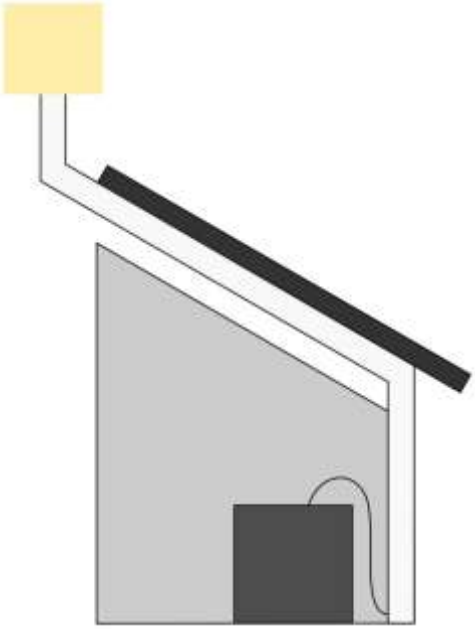
8.2. Parte inferior

Para esta parte se usará la placa superior como protección ante lluvia y sol, por lo que se realizará una pequeña caja en la parte inferior. La idea será realizar una marcada división dentro de esta caja, donde en un compartimento se almacenarán los componentes que necesiten protección extra, tal y como se ha comentado anteriormente.

Para aprovechar al máximo el espacio, la parte inferior será completamente horizontal, mientras que el techo formará un ángulo de 45º para ajustarse a la placa superior. Esto condicionará al montaje de los componentes dentro de la PCB, teniendo que ajustarse lo máximo posible a estos contornos.

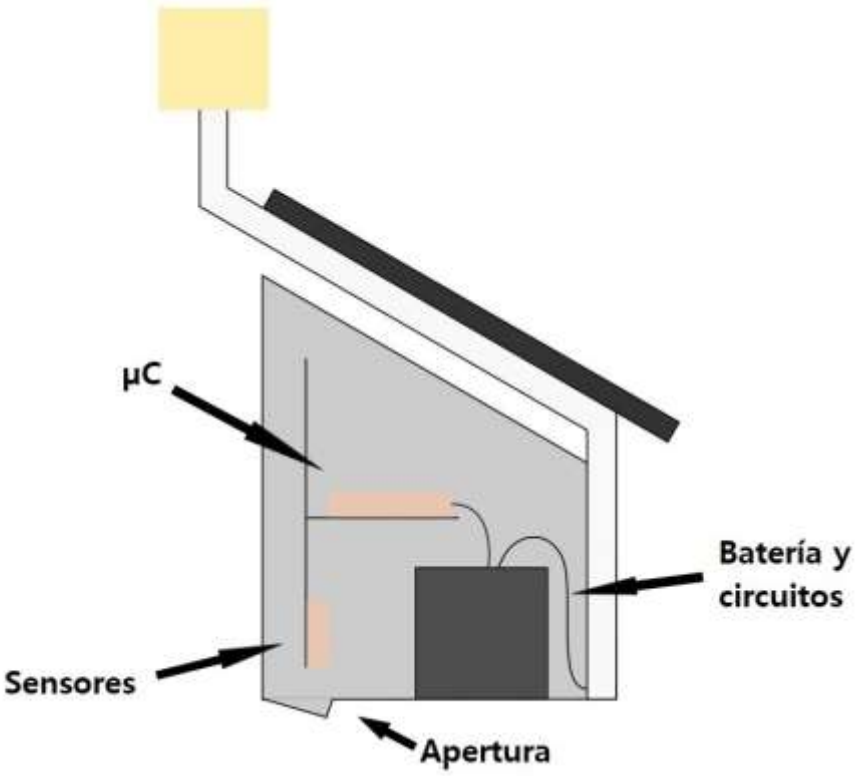
Se ha realizado una pequeña primera aproximación a las características a fin de empezar con el diseño de las placas y distribución de los componentes dentro de la caja inferior.

Imagen 67: Diseño parte inferior



Para el diseño de la PCB se va a aprovechar el espacio que se dispone dentro de la caja. Para reducir en la medida de lo posible el tamaño de la estación, se va a realizar una estructura 3D con forma de "T" que se ajustará al vértice superior; al espacio inferior cerca de la caja hermética; y la parte más cercana al tubo por donde bajará el cableado.

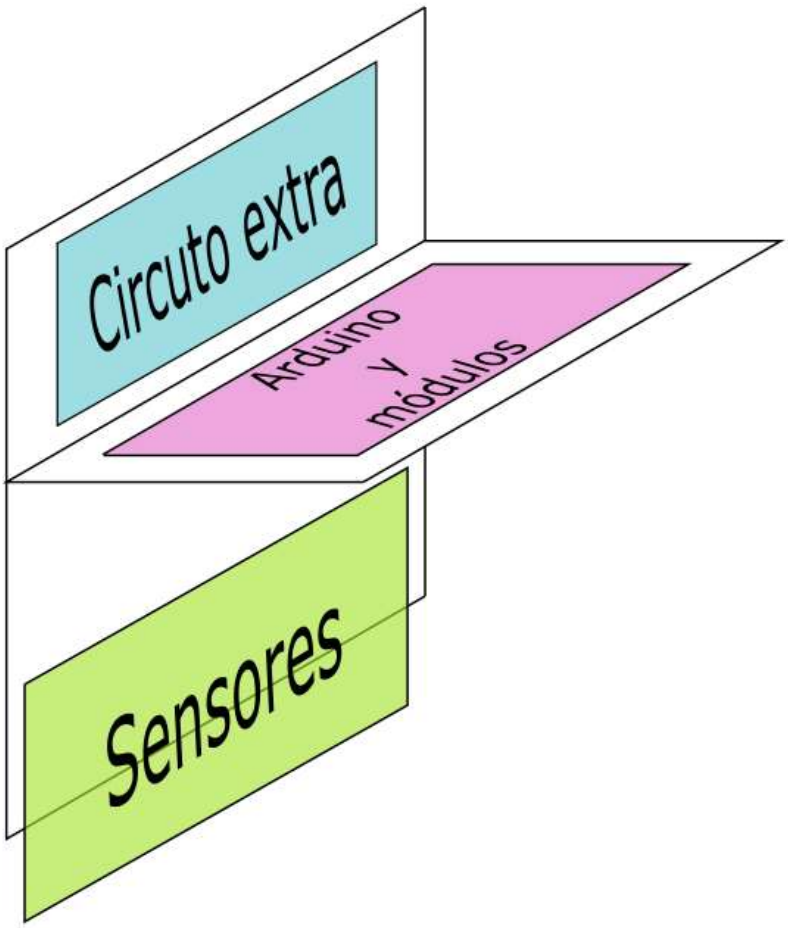
Imagen 68: Diseño interior



Mediante este diseño se conseguirá optimizar el espacio, así como colocar sensores en la parte inferior de la estación, cerca de una pequeña apertura protegida, en donde se podrá leer la humedad, temperatura… sin mayor problema.

Haciendo un zoom sobre las placas, se realizará una pequeña aproximación de la distribución de los componentes.

Imagen 69: Diseño PCB



Para el caso de la Antena RF, se colocará en la medida de lo posible en un lugar protegido y elevado, para que la señal no se vea afectada.

9. Montaje y fabricación

Una vez el diseño principal ha finalizado, es momento de comprobar si el diseño es viable y si es necesario alguna modificación. Para comenzar con el montaje se iniciará con las placas solares y todo el módulo superior. Y a continuación se realizarán los modelos 3D de la caja inferior y su posterior montaje.

9.1. Modelos 3D

En este apartado se van a realizar todos los modelos 3D requeridos para ensamblar correctamente los componentes. Para la realización de estos modelos se va a hacer uso del programa Catia V5, el cual es ampliamente usado en la industria.

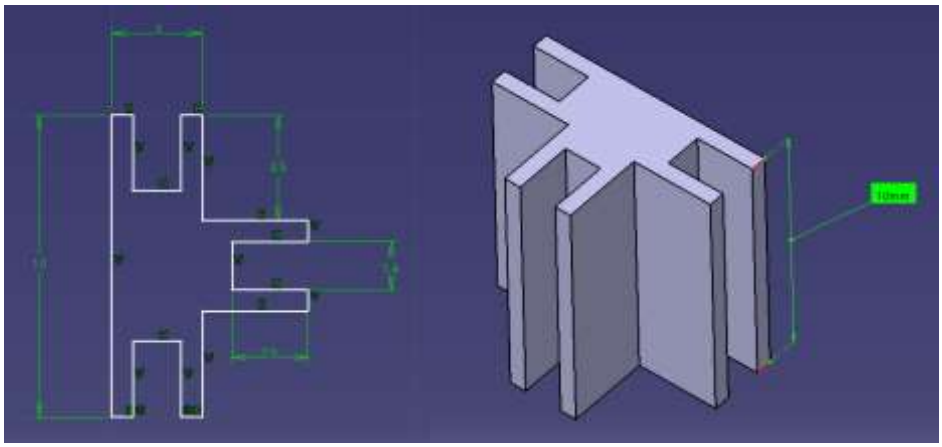
Estos modelos tomarán forma mediante su fabricación con una impresora 3D de PLA. El modelo de impresora que se usará durante la fabricación será Anycubic Kossel, la cual dispone de una precisión de 0.1/0.3 mm y un tamaño de impresión de 230mm de diámetro por 300 de alto. Entre otras características, esta impresora dispone de un único extrusor y cama calefactable que alcanza los 100ºC para evitar que se quede excesivamente pegada la pieza y se pueda romper durante su extracción.

Imagen 70: Impresora PLA

Enganches

Lo primero que se diseñará serán los enganches de las placas. Teniendo el grosor de las placas y usando el diseño en T, el modelo quedaría bastante sencillo.

Imagen 71: 3D - Enganches



Se realizarán dos piezas idénticas para evitar movimientos de las placas y evitar roturas.

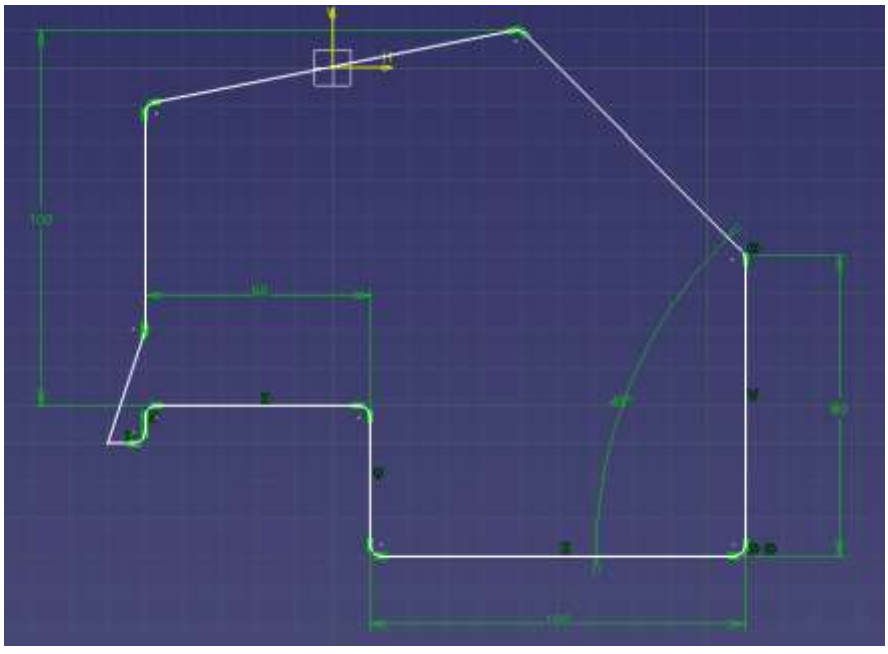
Caja exterior

Para el diseño de la caja se tendrán en cuanta las medidas de las placas que se van a usar, así como el tamaño del pack de baterías y todos sus componentes.

Las tres medidas que se van a tener en cuenta durante el diseño son

- Que el pack de baterías y sus componentes van a requerir un espacio de al menos 80x80mm. Necesita algo más de espacio para el cableado
- El Arduino y sus sensores necesitaran al menos 80X60
- Es necesario colocar pendiente en la parte superior para no acumular agua, el lado derecho necesitará una pendiente de 45º para ajustarse con las placas solares.

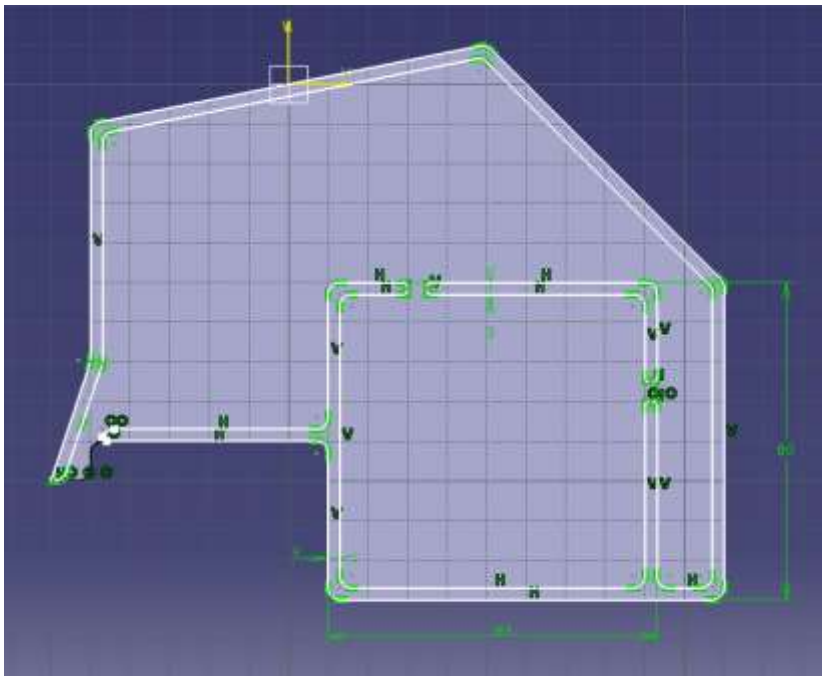
Imagen 72: 3D - Contorno caja



Para diseñar los compartimentos internos, en un principio se generará una pared de 3mm de grosor, y a continuación la caja de las baterias, que tal y como se ha mencionado necesita un espacio cercano a 80x80mm. A la derecha y en la parte superior de la caja de baterías se puede ver una pequeña ranura por donde accederán los cables.

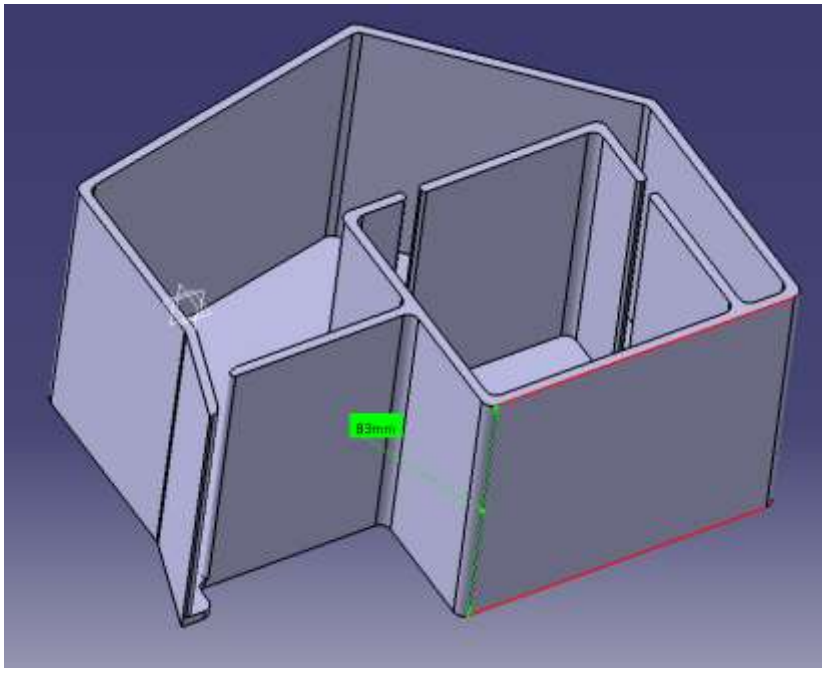
Por último, se colocará una apertura en la parte inferior izquierda para que acceda directamente a los sensores.

Imagen 73: 3D - Interior caja



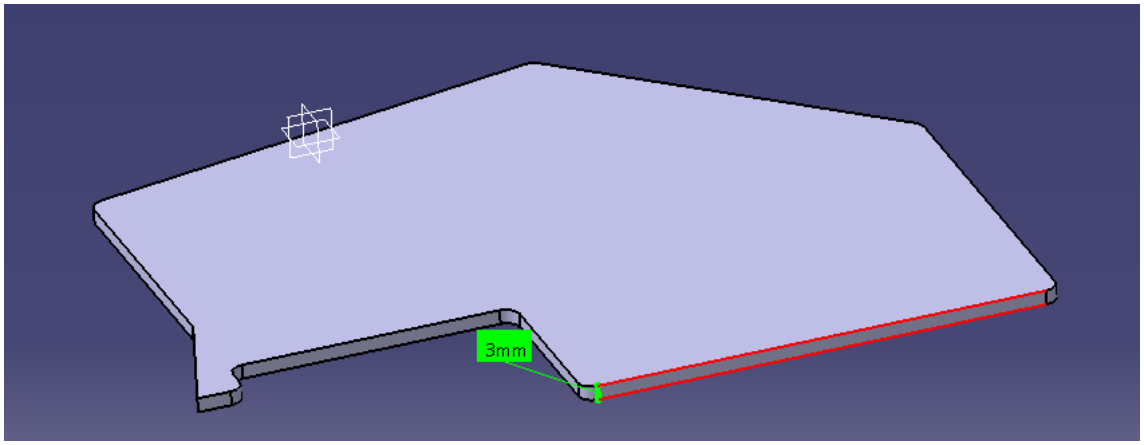
Se extruirá el contorno creado para darle volumen y alcance un grosor de 80mm el interior, siendo una de las tapas los 3mm extra.

Imagen 74: 3D - Volumen caja



Para crear la otra tapa, simplemente se usará el primer modelo con el contorno y se le dará un grosor de 3mm.

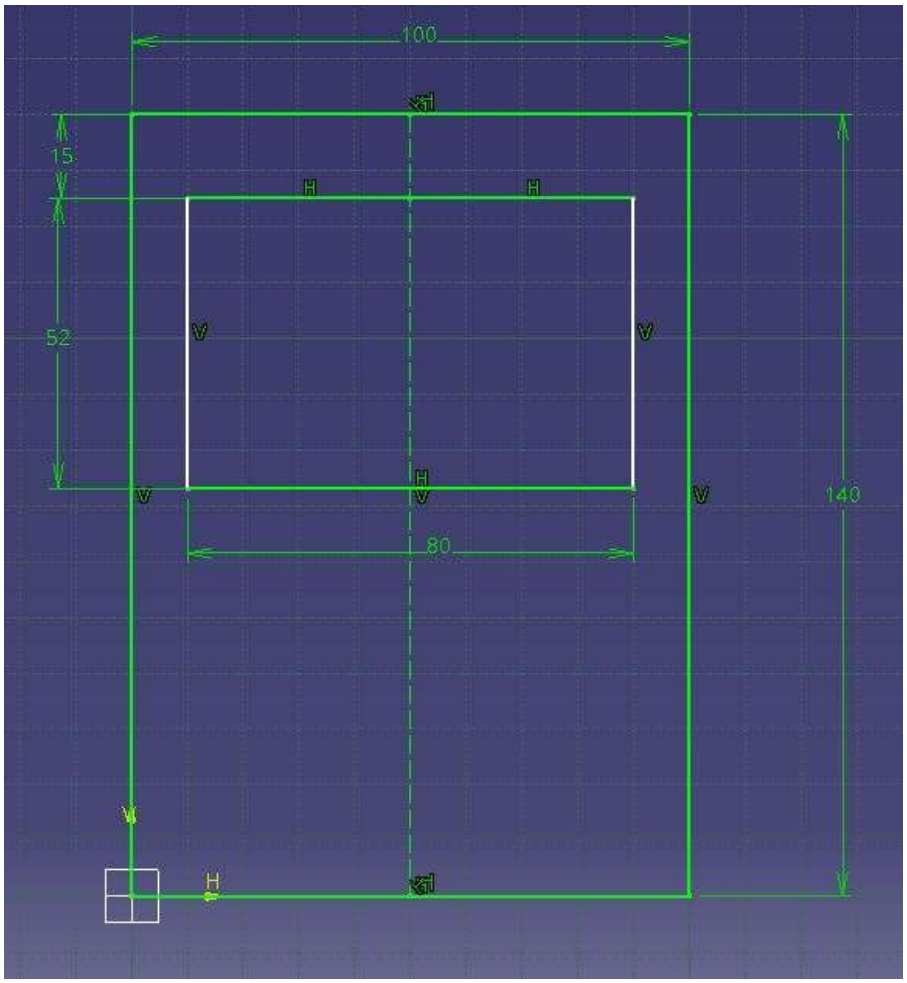
Imagen 75: 3D - Tapa



Caja interior

Para el modelo en 3D de la caja interior, se va a coger de referencia el tamaño de la pantalla para hacer la ranura, y posteriormente se diseñará una caja rectangular de anchura y largura suficientes para todo el cableado.

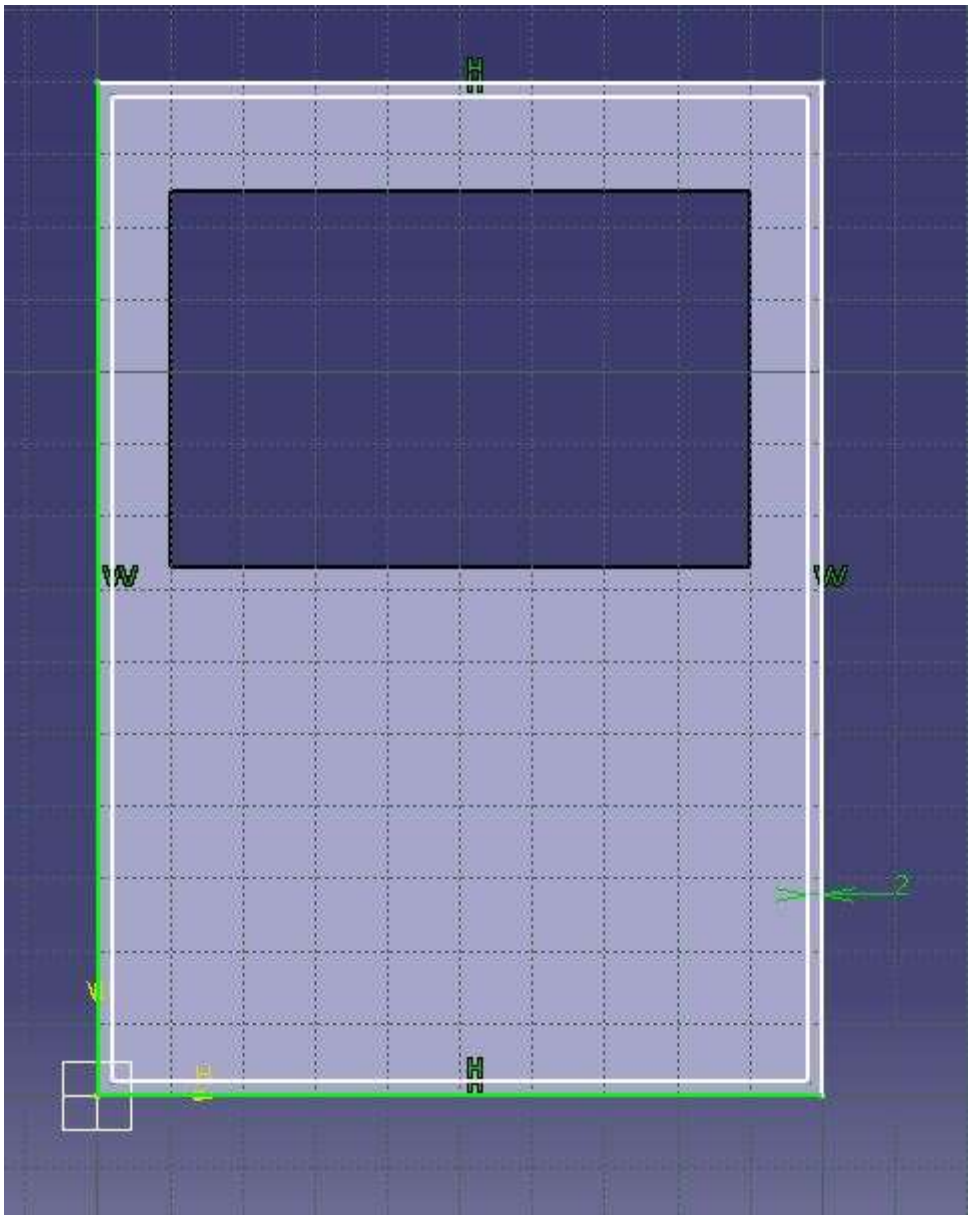
Imagen 76: 3D - Caja interior



Para poder mostrar por completo la pantalla se requiere de una ranura de 80*52mm, y se ha colocado centrada respecto al eje de la pieza y a altura suficiente para permitir colocar el cableado en la parte inferior.

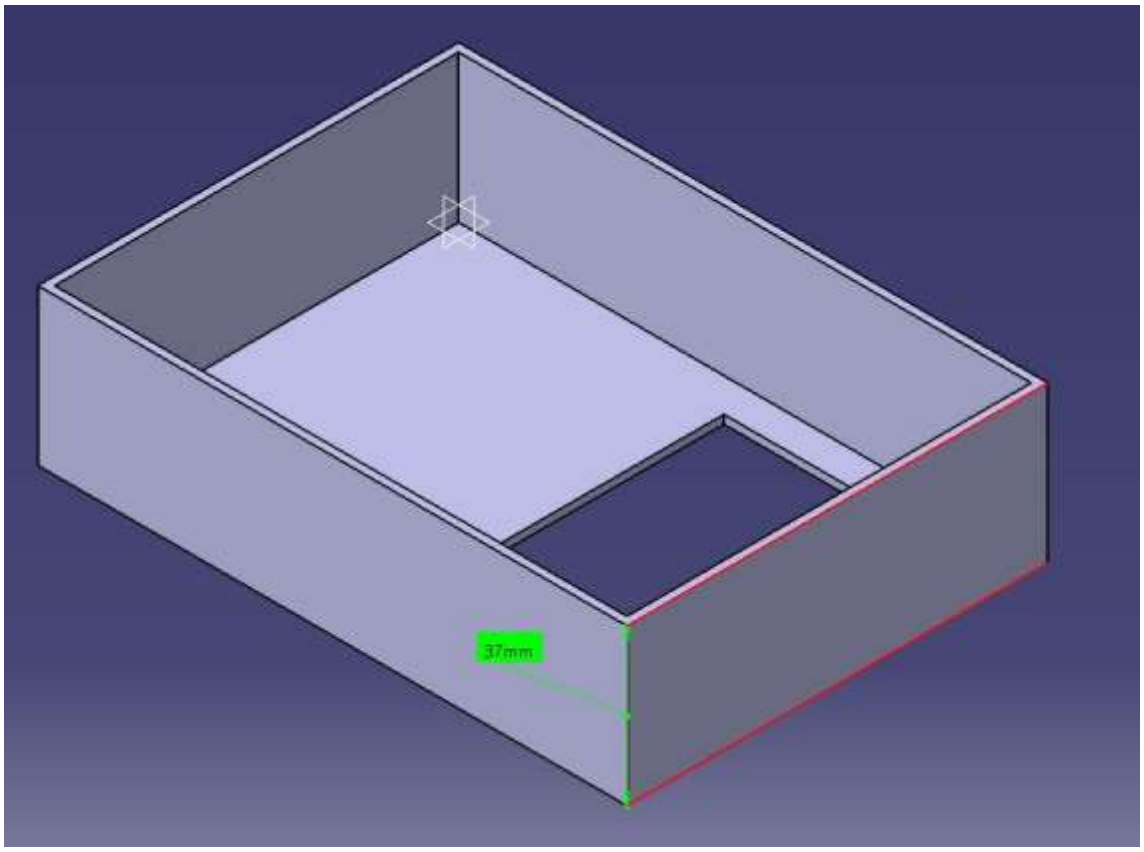
Para esta caja las paredes se van a diseñar para 2mm, por lo que el diseño quedaría:

Imagen 77: 3D - Paredes Caja interior



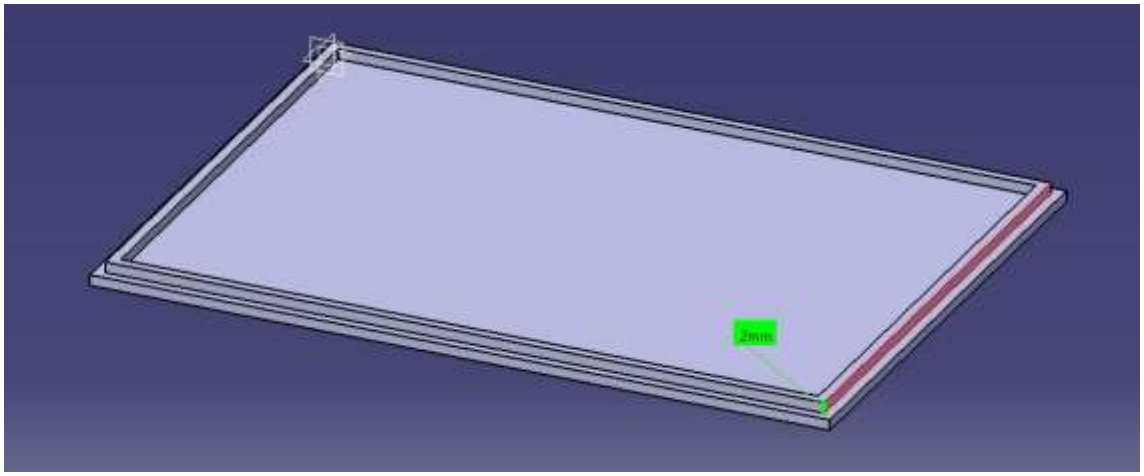
Extruyendo ambas formas, la primera con 2mm y la segunda con 35mm, se conseguiría un tamaño de piezas de 37, siendo 35mm el espacio útil en su interior.

Imagen 78: 3D - Volumen Caja interior



Para la tapa se creará un rectángulo de 100*140mm, que irá encajada en la parte final de la caja mediante un pequeño saliente

Imagen 79: 3D - Tapa Caja interior



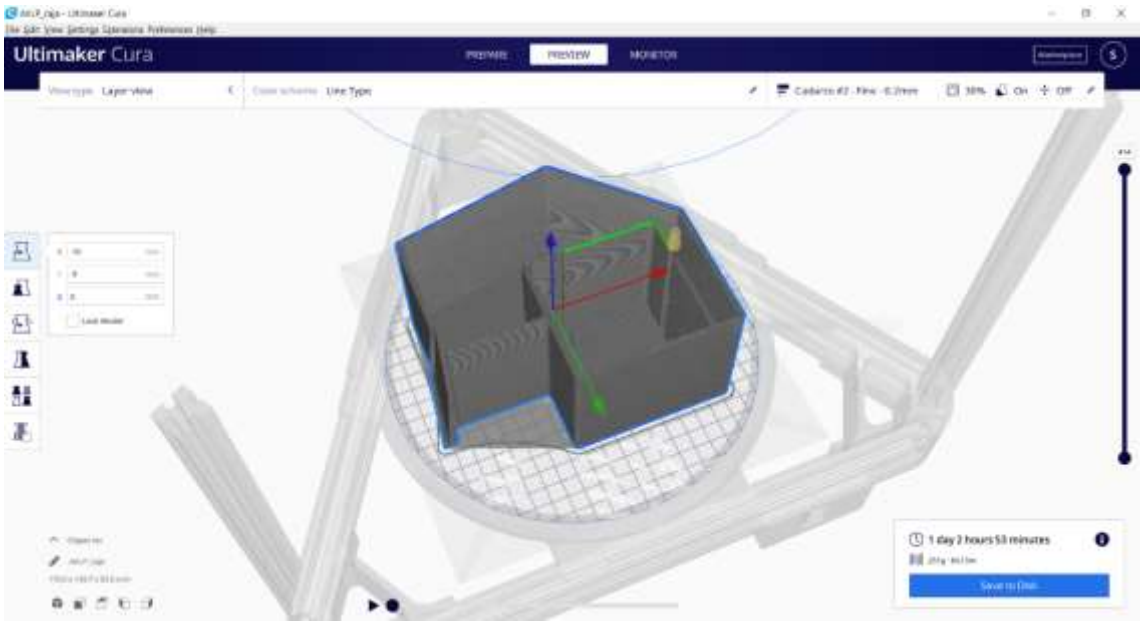
9.2. Fabricación

Para la fabricación de estos modelos se hará uso del programa Ultimate Cura, con el cual será posible cargar los modelos 3D a la impresora. Para ello, será necesario convertir los archivos a *.stl, un formato compatible para la impresora 3D. El formato stl es un formato de archivo de diseño asistido por computadora que define la geometría de los objetos 3D sin definir la información de texturas o color de la figura.

Una vez se ha instalado el programa, será necesario realizar unos ajustes básicos de temperatura, medidas de la impresora y modo de impresión, para asegurarse un buen funcionamiento del programa. Tras la carga del archivo en la impresora, queda definir la densidad de nuestra pieza final y darle a slice.

Mediante la ejecución de slice, la impresora hace sus cálculos de tiempo, cantidad de material y forma de ejecución, y con ello es capaz de mostrar una vista previa de la ejecución de la impresión.

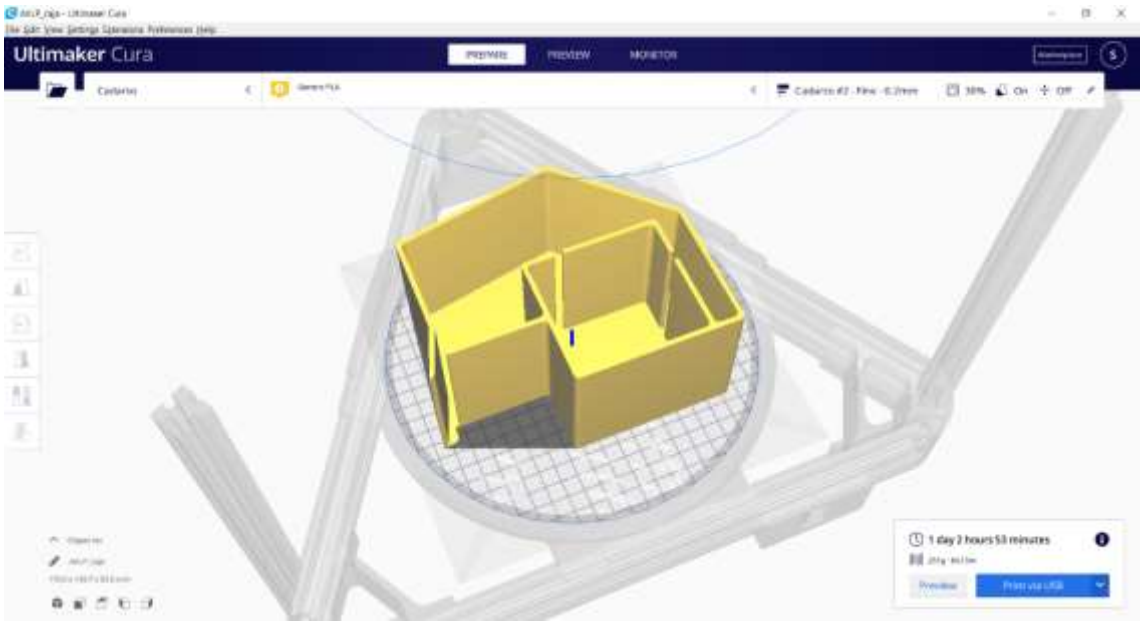
Imagen 80: Cura



En este modo, entre otras cosas, permite ver el recorrido que va a seguir la pieza y si requiere de material extra para poder realizar puentes o sobresalientes en la pieza.

Una vez se ha guardado en el disco interno (memoria) del ordenador el programa con toda la ejecución, será posible conectar la impresora y e imprimir.

Imagen 81: Cura impresión

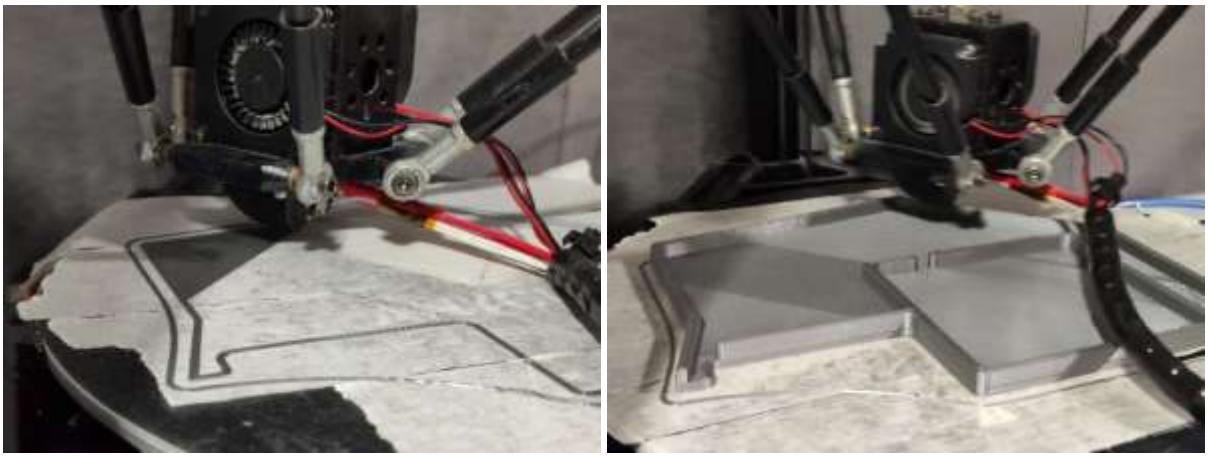


Una vez se inicia la impresión, tanto el extrusor como la cama (base) de la impresora empiezan a calentarse hasta los valores fijados durante la configuración. Y una vez se alcanza ese valor óptimo comienza la impresión.

Imagen 82: Cura preparación

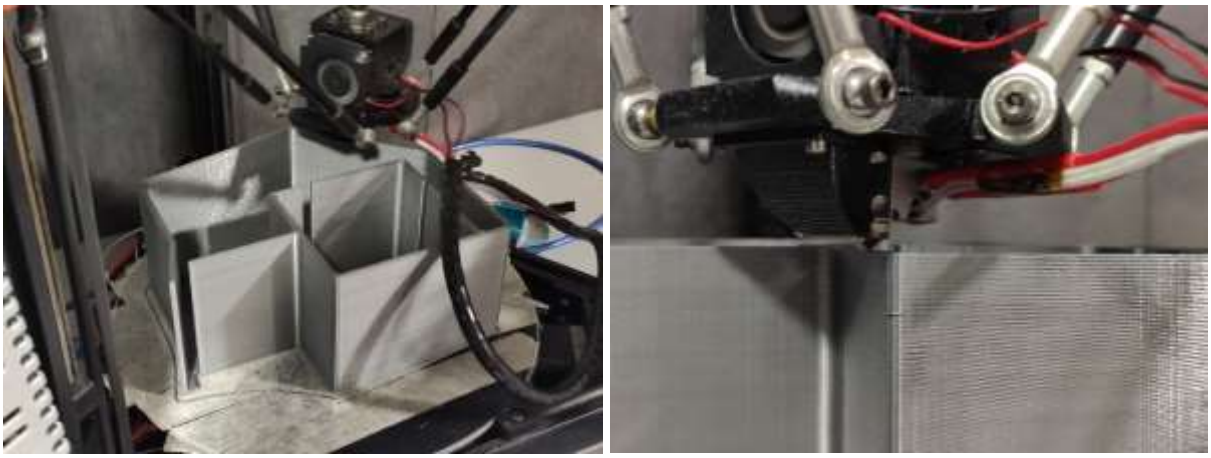
Tras calentarse, la impresora comienza la impresión desde la parte inferior, elevándose poco a poco con cada pasada y realizando la forma indicada.

Imagen 83: Comienzo Impresión



Una vez ha transcurrido el tiempo necesario la impresora ha realizado la pieza deseada.

Imagen 84: Finalizando Impresión



Para desmoldear cómodamente la pieza, se ha colocado previamente cinta de carrocero y un poco de laca para pelo. Con esto se consigue reducir la adherencia de la pieza a la cama y evitar posibles roturas al desmoldear.

A continuación, se procederá a imprimir la tapa siguiendo los mismos pasos realizados anteriormente con la tapa.

Para la impresión de la caja de interior se realizará el mismo procedimiento que lo ejecutado anteriormente.

Y una vez se tienen todas impresas se procederá al montaje de cada una de las partes.

9.3. Montaje

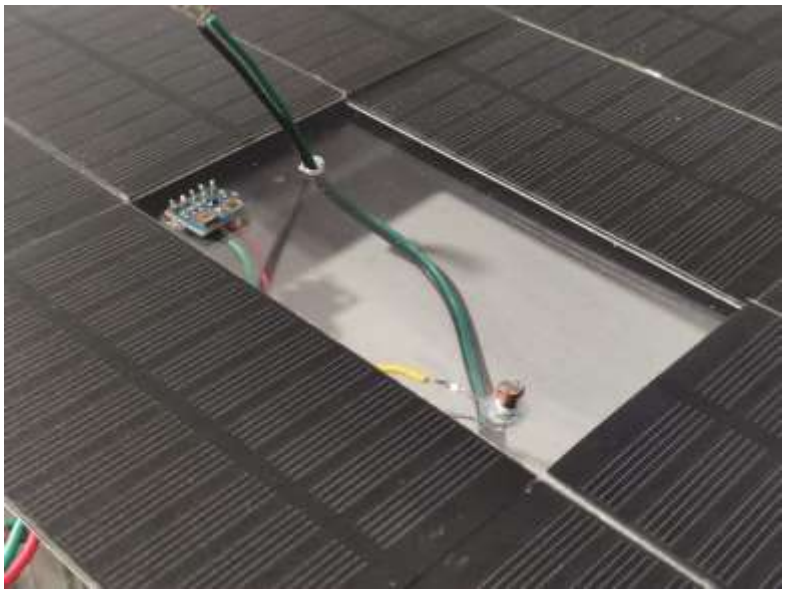
Módulo exterior

Se comenzará con el montaje de la parte superior. Tal y como se ha comentado anteriormente, las placas, y los sensores de luminosidad y UV irán entre dos placas de metacrilato herméticamente unidas apoyadas sobre una estructura de PVC con forma de "Y" que soportará a los sensores de viento.

Imagen 85: Montaje - Parte superior

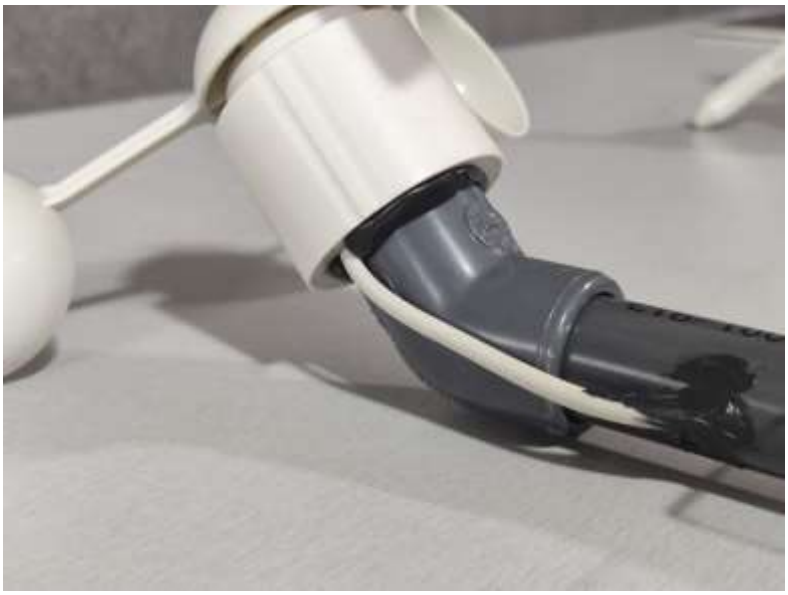
En esta primera imagen se puede ver la placa inferior con la estructura de PVC ya pegada y tanto los sensores como placas solares ya colocadas y pegadas.

Imagen 86: Montaje - Sensores superiores



En esta imagen se ve la colocación de los sensores de UV y el fotorresistor. Además de estos dos sensores, hay dos cables extra que van enganchados al sensor de lluvia, que se colocará una vez esté la placa de metacrilato superior colocada.

Imagen 87: Montaje - A prueba de agua



En la parte superior, pegados mediante silicona, se encuentran los sensores de viento correctamente sujetos y cerrados ante cualquier entrada de agua.

Imagen 88: Montaje - Cableado

Por último, en el tubo principal se han realizado agujeros adicionales para meter cables y puedan bajar por la columna principal hasta la caja inferior.

Ahora se soldarán todos los sensores y circuitos. Para ello se unirán las placas de PCB mediante las T impresas y se comenzará soldando desde el Arduino hasta terminar con todos los sensores necesarios.

A continuación, se muestran varias fotos del montaje realizado:

Imagen 89: Soldadura de componentes inicio

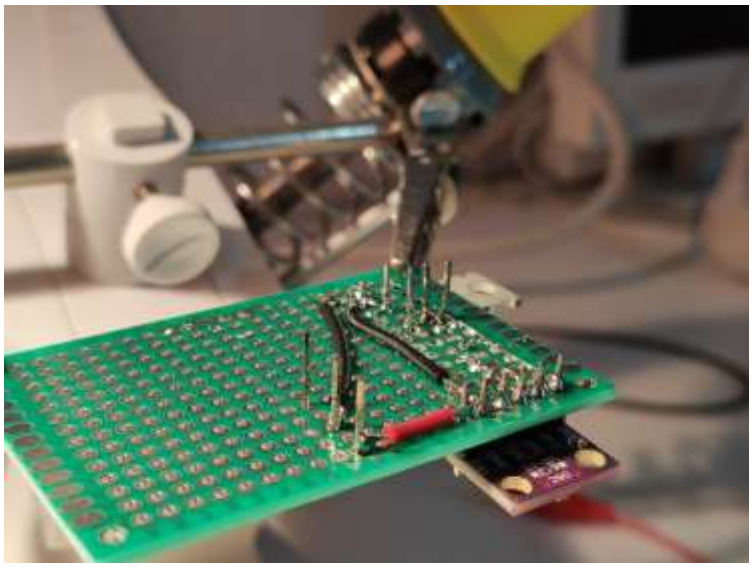
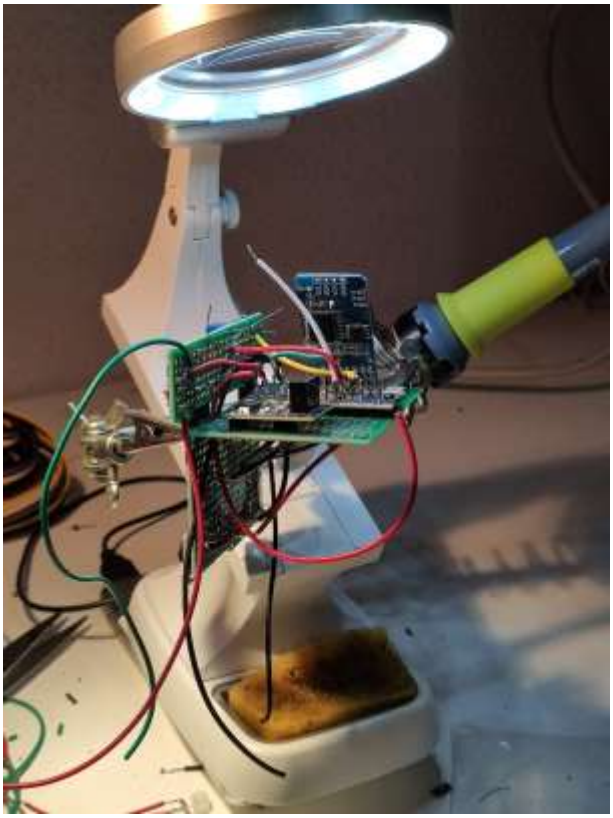


Imagen 90: Soldadura de componentes finalizando



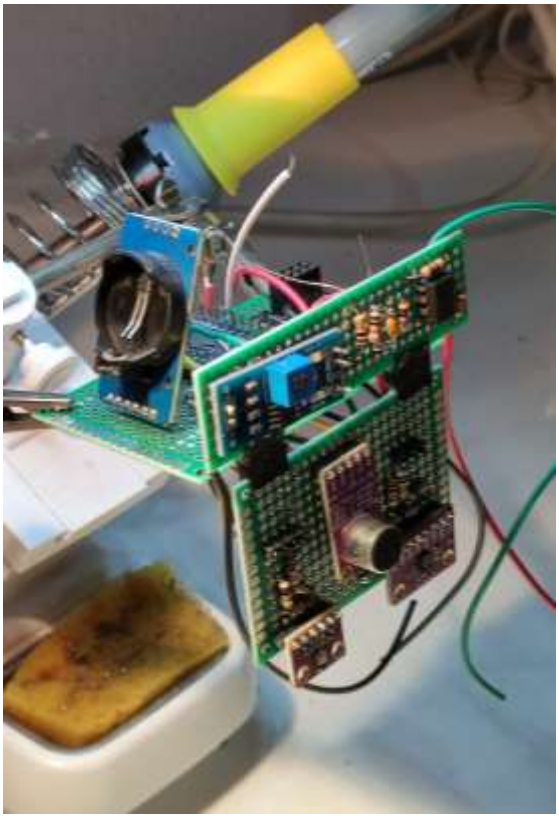
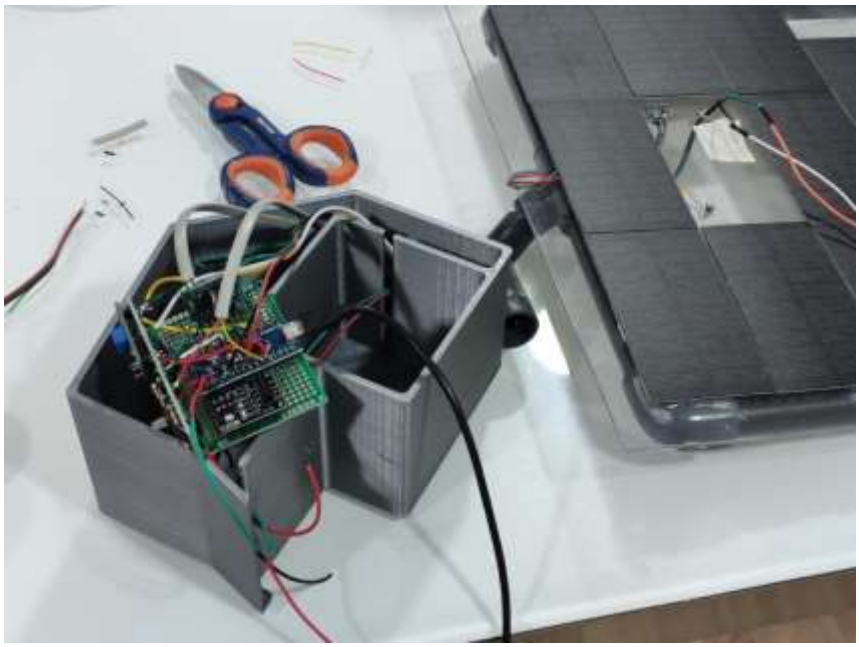


Imagen 91: Unión parte superior e inferior

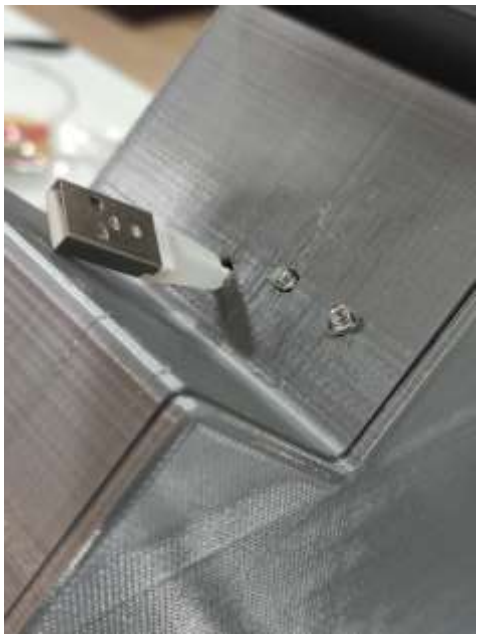


Durante el montaje de las distintas partes se ha ido cargando uno a uno los programas para comprobar el funcionamiento individualizado, y una vez montados todos se ha probado el código principal.

Tras comprobar en funcionamiento de la estación meteorológica completa, se procederá a ensamblar todo y prepararlo para la puesta en marcha en el exterior.

Imagen 92: Ensamblaje casi al completo

Imagen 93: Últimos detalles de montaje



Tal y como se puede ver en la imagen, se ha colocado un puerto externo para posibles mejoras del código sin necesidad de despegar todo.

Se ha decidido ubicar la estación en el tejado de la caja a fin de evitar lecturas incorrectas de viento.

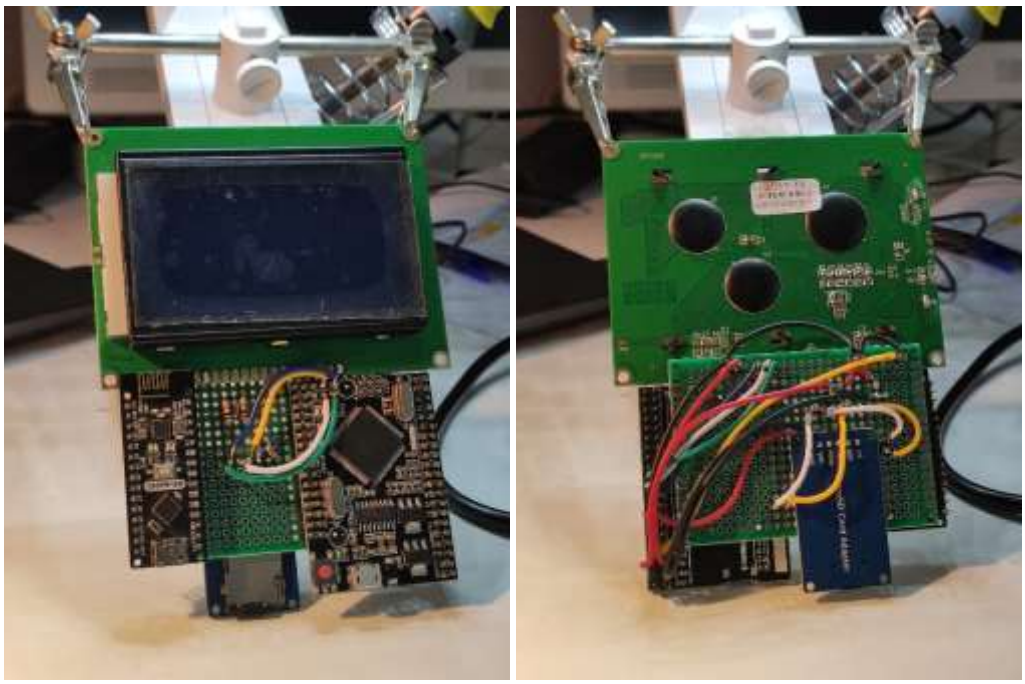
Imagen 94: Ubicación estación

Módulo interior

Para el montaje del módulo de interior, se soldará en una única placa todos los componentes a finde reducir el tamaño y poder colocarlo discretamente en algún lugar de la casa.

Se soldará según el esquema eléctrico diseñado y los resultados se pueden ver en las imágenes inferiores.

Imagen 95: Montaje módulo interior



Una vez ha terminado la impresión de la caja que ha durado unas 8 horas aproximadamente, se hace un agujero para la alimentación y se cierra con todos los componentes en su interior.

Imagen 96: Componentes dentro de la caja

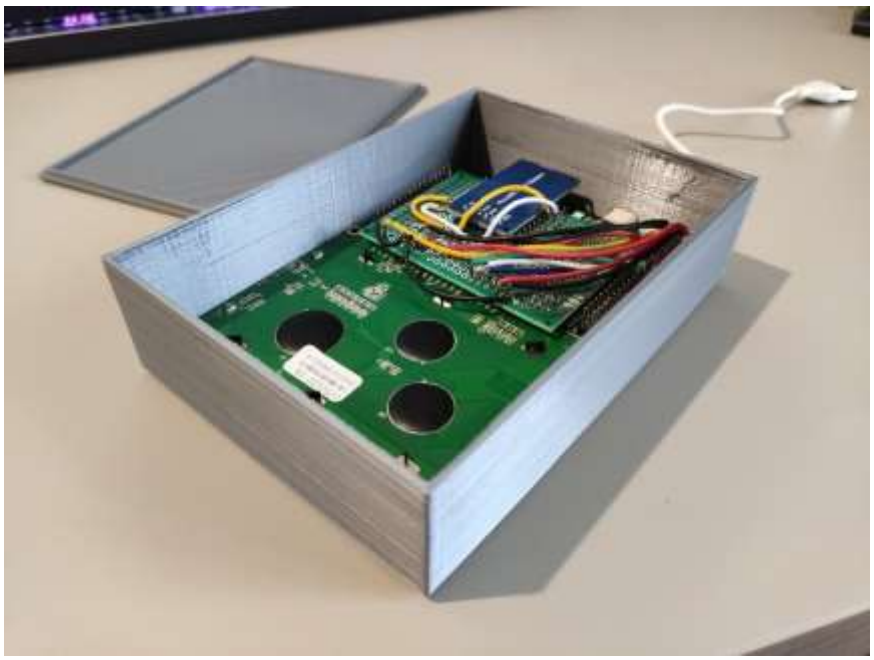


Imagen 97: Caja interior cerrada

10. Pruebas de funcionamiento

Tras ensamblar todas las piezas, llega el momento de probar su funcionamiento y comprobar que no hay errores durante la ejecución. Para ello se van a hacer dos pruebas distintas:

- Una primera prueba para comprobar que la pantalla muestra correctamente los valores recogidos en el exterior
- Y una segunda prueba donde se extraerá la tarjeta y se observará en el Excel los valores

10.1. Prueba monitor

Para esta prueba se va a observar que las pantallas se muestran correctamente, siendo coherentes los datos y no habiendo problemas durante la transición entre pantallas.

Imagen 98: Salvapantallas

Imagen 99: Pantalla 1

Imagen 100: Pantalla 2

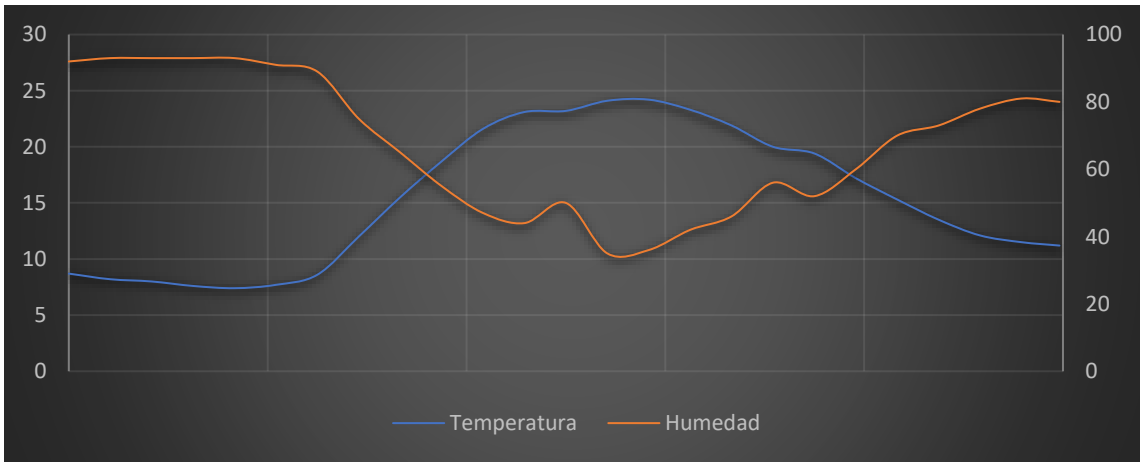
Imagen 101: Pantalla 3

10.2. Pruebas SD

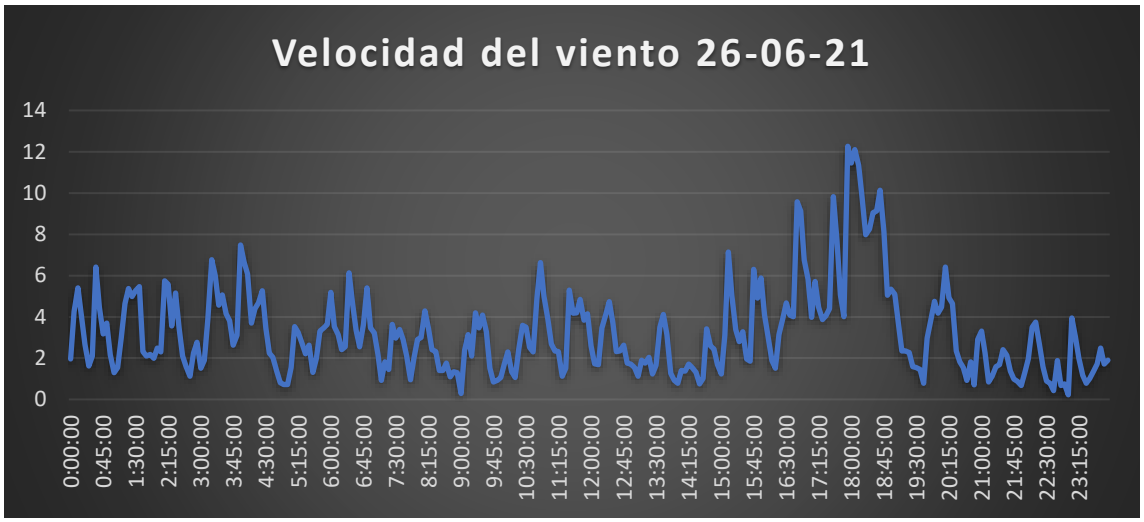
Para esta prueba se va a extraer la memoria SD del módulo interior y se van a realizar distintas gráficas con los parámetros recogidos durante el día.

Se muestra los valores de temperatura, y velocidad y sentido del viento:

Gráfica 11: Temperatura y humedad día completo



Gráfica 12: Velocidad del viento día completo



Gráfica 13: Sentido del viento día completo

11. Costes totales

En este apartado se va a hacer un pequeño calculo con el coste final del proyecto y si ha sido posible ajustarse al presupuesto inicial, y con ello conseguir un precio competente en el mercado.

Los precios de los componentes por separado y contando gastos de envío han sido:

- Arduino nano – 1.84€
- Modulo Reloj – 1.54€
- BME280 – 1.59€
- Anemómetro – 11.47€
- Veleta – 11.47€
- CCS811 – 5.28€
- Sensor UV – 2.71€
- Sensor Sonido – 1.03€
- Sensor lluvia – 1.22€
- LDR – 0.05€
- Antena RF – 3.68€
- Arduino Mega Pro – 6.32€
- Arduino RF-nano – 4.21€
- Pantalla LCD – 4.91€
- Adaptados microSD – 0.48€
- Circuito reductor a 5V – 3.55€
- Circuito Buck-boost – 2.42€
- Cableado total comprado – 2.49€
- Componentes electrónicos varios – 2.67€
- Placas solares – 13.50€
- BMS – 1.45€
- Baterías – 6.19€
- PCB de topos – 3.52€
- Placa metacrilato – 6.99€
- Tubo PVC y codos – 4.67€

Siendo estos todos los componentes comprados, se tiene un precio total de 105.25€, que sobrepasa ligeramente el presupuesto inicial, pero se ajusta al rango permitido según objetivos iniciales.

Además, hay que tener en cuenta que los modelos fabricados en 3D también suponen un gasto por la utilización de PLA. La impresora hace una aproximación al gasto de cada una de las piezas fabricadas, y para el conjunto de cajas más tapas, aproxima a un valor de 11€.

Con todo esto se obtiene un coste total de 116.25€ por cada modelo a fabricar.

En caso de realizar un montaje masivo de estas estaciones, se podría reducir en gran medida el coste final, ya que al comprar al por mayor se pueden reducir notablemente el coste de cada componente y los gastos de envío se repartirían.

12. Conclusiones

Hacer una estación meteorológica autónoma alimentada por placas solares fue una idea que se me ocurrió hace tiempo y que estaba con ganas de ponerla en marcha. Me ha parecido buena idea realizarlo para el Trabajo fin de Grado por la complejidad y variedad de ramas de la electrónica que se tratan. Durante todo el proyecto se ha usado la programación como nexo, pero además hay componentes analógicos y digitales, circuitos electrónicos, telecomunicación y comunicación cableada, además de parte de potencia y parte de fabricación, que siendo ingenieros técnicos es algo a tener muy en cuenta.

Durante el estudio de los componentes ha sido de gran ayuda todos los foros y tutoriales disponibles en la web y posiblemente es uno de los motivos por los que he seleccionado el Arduino. Los Arduinos, además de todas las características que se han nombrado durante el proyecto, es un microcontrolador a tener en cuenta a la hora de hacer estos proyectos (DIY / Do it yourself o Hágalo usted mismo), ya que la comunidad es uno de los pilares fundamentales del éxito de estos componentes, que día a día apoya el proyecto open-source (código abierto) donde la única finalidad es ayudarse unos a otros en el manejo de estos componentes. A lo largo de todo el trabajo se han usado estos foros de divulgación donde he aprendido a solucionar los problemas que me han podido surgir.

De las cosas que más me han gustado de este proyecto es la cantidad de programas que he aprendido a usar y ver cómo poco a poco iba mejorando en el manejo de éstos. Para la programación he usado el IDE de Arduino, que había utilizado con anterioridad en algún otro proyecto, pero la complejidad de éste me ha llevado a mejorar en la programación, obteniendo gran diferencia entre el código realizado en las primeras pruebas y el código final del proyecto.

Además del IDE se han usado programas de electrónica, como por ejemplo el Proteus, donde he podido realizar varios circuitos cableados además de hacer alguna simulación de funcionamiento antes de realizar el montaje. Ha sido de las primeras veces que he usado este programa, ya que en el grado cursado se ha utilizado Multisim y sinceramente éste me ha parecido más completo e intuitivo.

Para todo el tema de diagramas y modelos, se ha usado el programa Inkscape, un programa de código abierto que me recomendó mi tutor, con el que he conseguido de forma sencilla realizar modelos 2D y 3D de las propuestas de diseño.

Una de las partes que más me ha costado desarrollar ha sido el uso de protocolos de comunicación serie. A lo largo del trabajo, se ha hecho uso de forma continua de estos protocolos y la verdad es que ha sido todo un reto conseguir dominarlos casi al completo. Gran parte del tiempo lo he dedicado al estudio y a las pruebas con los distintos sensores, hasta el punto de realizar comunicaciones completas entre distintos módulos. Antes de iniciar el proyecto tenía unas nociones básicas del funcionamiento de estos protocolos de comunicación, ya que en alguna asignatura se han mencionado y me ha servido de ayuda para tener un punto de apoyo sobre el que poder empezar a estudiarlos.

Posiblemente la parte del proyecto en la que mayor cantidad de problemas he tenido ha sido durante la fabricación: a lo largo del diseño se han plasmado todas las ideas que he tenido de cómo realizar el montaje, pero una vez que se comienza a fabricar y montar, llegan los problemas: diseño de la estación, ubicación de los componentes, estructura e inclinación de todo el conjunto para que todos los sensores cumplan sus objetivos (veleta, anemómetro, optimización de las placas solares…).

Durante el diseño y fabricación de las piezas 3D, se han tenido problemas de todo tipo. Sobre todo al final se han realizado bastantes modificaciones en diseño y fabricación a fin de conseguir que todo funcione. Durante la parte de diseño por ordenador, se ha usado el programa Catia V5, un programa que he usado en contadas ocasiones, y que para este proyecto he tenido que exprimir a fin de reducir espacio para el tamaño de impresora y conseguir modelos factibles. Pero el mayor problema lo he tenido a la hora de la fabricación.

Para la fabricación he usado una impresora de PLA que nunca antes había usado, y que además de aprender a usarla, he tenido que familiarizarme con el programa de impresión. El programa de impresión Cura me ha resultado bastante intuitivo, pero en más de una ocasión he tenido problemas para hacerla funcionar correctamente. Además de todos los problemas de fabricación que he tenido, donde piezas han salido incorrectas o incompletas, he tenido que calibrar varias veces la impresora, teniendo que modificar ligeramente el código (el µC que lo compone es un Arduino Mega) y teniendo que nivelar la bandeja y railes para evitar choques del extrusor.

Para mí, solucionar estos problemas con la impresora ha sido motivador, ya que la impresión en 3D es el futuro.

Para el montaje final, el tema del reducido espacio y conseguir una estación hermética para posibles lluvias, me ha llevado a muchos rompederos de cabeza, ya que no conseguía realizar ningún diseño compacto que pudiese soportarlo. Para inspirarme he acudido a varias páginas de proyectos al aire libre y llegue a la conclusión de que colocar todos los componentes bajo las placas solares era la opción correcta.

Una vez acabado el proyecto estoy bastante orgulloso con el resultado obtenido, ya que, aunque han sido cerca de 7 meses de trabajo, he conseguido hacerlo funcionar contra todo pronóstico. Varias veces a lo largo de la ejecución me he replanteado si el trabajo se estaba convirtiendo en demasiado ambicioso, desde un principio tenía un montón de ideas, y algunas de ellas he tenido que ir rechazando por falta de conocimiento y tiempo para poder desarrollarlas. La verdad es que hay un montón de ideas que todavía tengo en mente y posiblemente las acabe implantando en el proyecto, pero, aunque no haya tenido tiempo para realizarlo todo, estoy muy satisfecho por haber alcanzado los objetivos iniciales y estas ideas las desarrollaré como ampliación a un modelo que a día de hoy es completamente funcional.

La experiencia ha sido muy satisfactoria.

13. Bibliografía

Arduino y sensores

[1] https://www.naylampmechatronics.com/

[2] https://www.arduino.cc/

[3] https://www.arduino.cc/reference/en/

[4] https://forum.arduino.cc/

[5] https://controlautomaticoeducacion.com/

[6] https://arduwiki.perut.org/wiki

[7] https://www.luisllamas.es/

[8] https://www.digilogic.es/

Medio Ambiente

[9] https://www.miteco.gob.es/

[10]https://ec.europa.eu/

Inclinación placas solares

[11] https://astro.unl.edu/nativeapps/

14. Anexo

14.1. Código

14.1.1. Fundamento teórico

SPI

```
#include <SPI.h>

#define SS 5    //pin selector esclavo

void setup() {

  pinMode(SS, OUTPUT);      //pin de salida
  digitalWrite(SS, HIGH);   //mantenemos a alto

  SPI.begin();    //se inicia el HW del bus

}

byte comunicacion_SPI(byte enviar){

  //se inicia la comunicacion ajustando la configuracion de comunicacion
  SPI.beginTransaction(SPISettings(15000000, MSBFIRST, SPI_MODE0));
  //se escribe un 0 en el pin del esclavo (inicio comunicacion)
  digitalWrite(SS, LOW);
  //se envia en byte y se guarda la respuesta
  byte recibido = SPI.transfer(enviar);
  //se escribe un 1 en el pin del esclavo (detener comunicacion)
  digitalWrite(SS, HIGH);
  //se termina la comunicacion y se deja el bus libre
  SPI.endTransaction();
  //se devuelve el byte leido
  return recibido;
}
```

14.1.2. Código Componentes

14.1.2.1. Temperatura, Presión y Humedad

Escribir registro

```
void escribir_registro_BME(byte registro,byte valor){

  registro += 0x80;     //se ajusta el registro al comando escritura

  SPI.beginTransaction(SPISettings(velocidad_SPI, MSBFIRST, SPI_MODE0));
  digitalWrite(SS, LOW);          //se pone a 0 el pin de seleccion
  SPI.transfer(registro);         //se envia el registro que se desea escribir
  SPI.transfer(valor);            //se envia el byte de datos
  digitalWrite(SS, HIGH);         //se deselecciona el esclavo
  SPI.endTransaction();
  return 0;
}
```

Leer registro

```
byte leer_registro_BME(byte registro,int n_bytes){

  SPI.beginTransaction(SPISettings(velocidad_SPI, MSBFIRST, SPI_MODE0));
  digitalWrite(SS, LOW);                  //se selecciona el esclavo
  SPI.transfer(registro);                 //primer registro que se va a leer
  for (int i=0;i<n_bytes;i++) {
        lectura[i] = SPI.transfer(0);     //cada lectura obtendrá un byte
    }
  digitalWrite(SS, HIGH);                 //se deselecciona el esclavo
  SPI.endTransaction();
  return lectura[0];                      //se devuelve la primera lectura
}
```

Iniciar BME

```
void iniciar_BME(void){

  escribir_registro_BME(reg_RESET,0xB6);
  delay(10);

  while(leer_registro_BME(reg_ID,1)!=0x60){    //se espera hasta que el modulo arranque
    escribir_registro_BME(reg_RESET,0xB6);
    delay(10);
  }

  leer_registro_BME(reg_STATUS,1);
  bool valor = lectura[0]>>3 & 1;
  while(valor==true){                //se espera hasta que termina de medir
    leer_registro_BME(reg_STATUS,1);
    valor = lectura[0]>>3 & 1;
  };
  delay(10);

  leer_dig();          //se leen los registros de calibracion

  //Se configura el oversampling a 8, realizará 8 mediciones de cada valor
  escribir_registro_BME(reg_HUM_config,0x04);
  escribir_registro_BME(reg_ctrl_meas,0x90);      //Sleep mode
  escribir_registro_BME(reg_config,0x0C);         //standby 0.5ms, filter 8 y 4 wires
  delay(100);

  return 0;
}
```

Leer Dig

```
void leer_dig(void){

  leer_registro_BME(0x88,2);     T1 = dos_a_uno(true);
  leer_registro_BME(0x8A,2);     T2 = dos_a_uno(true);
  leer_registro_BME(0x8C,2);     T3 = dos_a_uno(true);

  leer_registro_BME(0x8E,2);     P1 = dos_a_uno(true);
  leer_registro_BME(0x90,2);     P2 = dos_a_uno(true);
  leer_registro_BME(0x92,2);     P3 = dos_a_uno(true);
  leer_registro_BME(0x94,2);     P4 = dos_a_uno(true);
  leer_registro_BME(0x96,2);     P5 = dos_a_uno(true);
  leer_registro_BME(0x98,2);     P6 = dos_a_uno(true);
  leer_registro_BME(0x9A,2);     P7 = dos_a_uno(true);
  leer_registro_BME(0x9C,2);     P8 = dos_a_uno(true);
  leer_registro_BME(0x9E,2);     P9 = dos_a_uno(true);

  H1 = leer_registro_BME(0xA1,1);
  leer_registro_BME(0xE1,2);     H2 = dos_a_uno(true);
  H3 = leer_registro_BME(0xE3,1);

  leer_registro_BME(0xE4,3);
  H4 = ((int16_t)lectura[0] << 4 | lectura[1] & 0x0F);
  H5 = ((int16_t)lectura[2] << 4 | lectura[1] & 0xF0);

  H6 = leer_registro_BME(0xE7,1);

  return 0;
}
```

Función Extra

```
word dos_a_uno(bool invertido){     //invertido: MSB está el [1] del array, se ha leido despues
  word unido = 0;

  if(invertido==true){
    unido += lectura[1];                  //se guarda el byte MSB
    unido <<= 8;                          //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[0];                  //se guarda el byte LSB
  }else{
    unido += lectura[0];                  //se guarda el byte MSB
    unido <<= 8;                          //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[1];                  //se guarda el byte LSB
  }
  return unido;
}
```

Leer Temperatura

```
float leer_temp(void){
  int64_t adc_T = 0;                //se crea variable de 64 bits
  int64_t var1 = 0,                 //se crea variables temporales para operaciones
          var2 = 0;

  leer_registro_BME(0xFA, 3);         //se leen los dos registros de temperatura empezando por el MSB (FAh)
  adc_T = tres_a_uno(false);

  if(adc_T==0x800000) return 0;       //si el registro tiene este valor no esta habilitada la lectura

  adc_T >>= 4;                        //se desplaza 4 a la derecha para aliminar los 0s del ultimo byte

  //Se realizan las operaciones indicadas en el datasheet
  var1 = ((((adc_T >> 3) - ((int64_t)T1 << 1))) *(int64_t)T2) >>11;
  var2 = (((((adc_T >> 4) - (int64_t)T1) *((adc_T >> 4) - (int64_t)T1)) >>12) * (int64_t)T3) >>14;

  t_fine = var1 + var2;

  long T = ((t_fine * 5 + 128) >> 8);
  return T / 100.;                    //se devuelve el valor de la temperatura
}
```

Leer Presión

```
float leer_pres(void){
  int64_t adc_P = 0;                //se crea variable de 64 bits
  int64_t var1 = 0,
          var2 = 0,
          P = 0;              //se crea variables temporales para operaciones

  leer_registro_BME(0xF7, 3);         //se leen los dos registros de presion empezando por el MSB (F7h)
  adc_P = tres_a_uno(false);

  if(adc_P==0x800000) return 0;       //si el registro tiene este valor no esta habilitada la lectura

  adc_P >>= 4;                        //se desplaza 4 a la derecha para aliminar los 0s del ultimo byte

  //Se realizan las operaciones indicadas en el datasheet
  var1 = (int64_t)t_fine - 128000;
  var2 = var1 * var1 * (int64_t)P6;
  var2 = var2 + ((var1 * (int64_t)P5) << 17);
  var2 = var2 + (((int64_t)P4) << 35);
  var1 = ((var1 * var1 * (int64_t)P3) >> 8) + ((var1 * (int64_t)P2) << 12);
  var1 =(((((int64_t)1) << 47) + var1)) * ((int64_t)P1) >> 33;

  if (var1 == 0) {
    return 0;          // para retirar el valor obtenido y ahora no dividir entre 0
  }

  P = 1048576 - adc_P;
  P = (((P << 31) - var2) * 3125) / var1;
  var1 = (((int64_t)P9) * (P >> 13) * (P >> 13)) >> 25;
  var2 = (((int64_t)P8) * P) >> 19;

  P = ((P + var1 + var2) >> 8) + (((int64_t)P7) << 4);
  return (float)P / 25600.;
}
```

Leer Humedad

```
float leer_hum(void){
  int32_t adc_H = 0;                //se crea variable de 32 bits
  int32_t var;              //se crea variables temporales para operaciones

  leer_registro_BME(0xFD, 2);         //se leen los dos registros de presion empezando por el MSB (FDh)
  adc_H = dos_a_uno(false);

  if(adc_H==0x8000) return 0;       //si el registro tiene este valor no esta habilitada la lectura


  //Se realizan las operaciones indicadas en el datasheet
  var = (t_fine - ((int32_t)76800));

  var = (((((adc_H << 14) - (((int32_t)H4) << 20) - (((int32_t)H5) * var)) + ((int32_t)16384)) >> 15) *
          (((((((var * ((int32_t)H6)) >> 10) * (((var * ((int32_t)H3)) >> 11) + ((int32_t)32768))) >> 10) +
          ((int32_t)2097152)) * ((int32_t)H2) + 8192) >> 14));

  var = (var - (((((var >> 15) * (var >> 15)) >> 7) * ((int32_t)H1)) >> 4));

  var = (var < 0 ? 0 : var);
  var = (var > 419430400 ? 419430400 : var);
  float h = (var >> 12);
  return h / 1024.0;
}
```

14.1.2.2. Anemómetro

```
bool leer_v_viento(void){

  int muestra = 0;       //numero de muestras tomadas
  word millis_ant = 0,   //se guarda el valor de millis anterior
       millis_act = 0,   //valor de millis actual
       anem_time = 0;    //tiempo de vuelta

  digitalWrite(anem_ON, HIGH);  //se alimenta el anemometro
  delay(100);

  while(muestra<4){   //se toman 4 muestras (una vuelta completa)

    while(digitalRead(anem_IN) != 0);     //se espera a que esté en bajo
    while(digitalRead(anem_IN) != 1);     //se espera a que esté en alto

    if(digitalRead(anem_IN) == 1){        //cuando detecta flanco entra
      millis_ant = millis();           //se recoge valor de millis

      while(digitalRead(anem_IN) == 1);  //se espera hasta el flanco de bajada

      if(digitalRead(anem_IN) == 0){     //se comprueba que esta a 0

        millis_act = millis();

        if(millis_act>millis_ant){     //se comprueba que no ha habido overflow
          anem_time += millis_act - millis_ant;

        }else {Serial.println("E3"); return 1;  }   //error overflow
      }else {Serial.println("E2"); return 1;  }   //error no ha vuelto a 0
    }else {Serial.println("E1"); return 1;  }   //error no se ha detectado flanco de subida

   muestra++;
  }

  anem_speed=((2*3.14*anemometer_R*1000)/(anem_time)*3.6)*factor;  //Km/h

  digitalWrite(anem_ON, LOW);   //se deja de alimentar
  delay(10);

  return 0;
}
```

14.1.2.3. Veleta

```
void loop() {
  veleta = analogRead(veletaPin);
  if(veleta > 100){
    if(veleta > 200){
      if(veleta > 300){
        if(veleta > 500){
          if(veleta > 700){
            if(veleta > 800){
              if(veleta > 900){
                Serial.println("Oeste");
              }else Serial.println("Noroeste");
            }else Serial.println("Norte");
          }else Serial.println("Suroeste");
        }else Serial.println("Noreste");
      }else Serial.println("Sur");
    }else Serial.println("Sureste");
  }else Serial.println("Este");
  delay(1000);
}
```

14.1.2.4. Sensor de lluvia

Función

```
byte intensidad_lluvia(void){

  byte intensidad = 0;                  //byte de respuesta

  digitalWrite(lluvia_ON,HIGH);         //se enciende el módulo
  delay(5);                             //se espera un poco a que arranque

  int lluvia = analogRead(lluvia_IN);   //se lee la entrada
  if(lluvia<700){
    if(lluvia<300){
      intensidad = 3;                   //si es inferior a 300: lluvia intensa
    }else intensidad = 2;               //si 300<=x<700: lluvia moderada
  }else intensidad = 1;                 //si es superior a 700: no llueve

  digitalWrite(lluvia_ON,LOW);          //se apaga el módulo
  return intensidad;                    //se devuelve el valor de la franja
}
```

Modo texto

```
String cuanto_llueve(byte intensidad){

  switch(intensidad){
    case 1: return("NO LLUEVE"); break;
    case 2: return("LLUVIA MODERADA"); break;
    case 3: return("LLUVIA INTENSA"); break;
  }
  return "0";
}
```

14.1.2.5. UltraVioleta

```
float leer_UV(void){
  digitalWrite(UV_ON, HIGH);   //se enciende el sensor
  delay(3);
  float tension = analogRead(entrada) * 5.036/1023; //se calcula equiv de V
  float valor = (tension-1) * (15./(2.9-1));
  digitalWrite(UV_ON, LOW);   //se apaga el sensor
}
```

14.1.2.6. Módulo Reloj

Escribir datos

```
bool escribir_reloj(byte direccion, byte registro,byte valor){
  bool ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(direccion);   //se inicia comunicacion
    Wire.write(registro);                //se envia registro a escribir
    Wire.write(valor);                   //se envia el valor a escribir
    ACK = Wire.endTransmission();        //se guarda en ACK si el esclavo
    delay(10);                            //ha obtenido bien los datos
  }
  return ACK;                            //se devuelve que la comunicacion ha sido exitosa
}
```

Leer reloj

```
byte leer_reloj(byte direccion, byte registro){
  bool ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(direccion);   //se inicia comunicacion
    Wire.write(registro);                //se envia registro a leer
    ACK = Wire.endTransmission();        //se guarda en ACK si el esclavo
    delay(10);                            //ha obtenido bien los datos
  }

  Wire.requestFrom(direccion,1);         //se le solicita al esclavo un byte
  while(Wire.available()!=1);            //se espera hasta obtener el dato
  byte valor = Wire.read();              //se guarda el dato
  return valor;                          //se devuelve el dato
}
```

Configurar alarma

```
bool configurar_alarma_reloj(byte direccion){
  bool ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(direccion);   //se inicia comunicacion
    Wire.write(0x0B);                    //primer registro de la alarma
    Wire.write(0x00);                    //se coloca A2M2 a 0, registro(0Bh)
    Wire.write(0x80);                    //se coloca A2M3 a 1, registro(0Ch)
    Wire.write(0x80);                    //se coloca A2M4 a 1, registro(0Dh)
    Wire.write(0x1E);                    //se coloca A2IE a 1, registro(0Eh)
    ACK = Wire.endTransmission();        //se guarda en ACK si el esclavo
    delay(10);                              //ha obtenido bien los datos
  }
  return ACK;
}
```

Limpiar flag

```
bool limpiar_flag_reloj(byte direccion){
  bool ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(direccion);   //se inicia comunicacion
    Wire.write(0x0F);                    //Status Register
    Wire.write(0x88);                    //limpiar flag
    ACK = Wire.endTransmission();        //se guarda en ACK si el esclavo
    delay(10);                             //ha obtenido bien los datos
  }
  return ACK;
}
```

Ajustar próxima alarma

```
void ajustar_prox_alarma(void){

  byte minuto = leer_reloj(Reloj_address,0x01); //se lee el min actual

  byte minuto_MSB = (minuto & 0xF0);      //se guardan los 4 bits mas significativos
  byte minuto_LSB = minuto & 0x0F;        //se guardan los 4 bits menos significativos

  if(minuto_LSB >= 0x05){     //si el minuto es mayor o igual a 5
    minuto_LSB = 0;           //se pone a 0
    minuto_MSB += 0x10;       //se suman 10 minutos
    if(minuto_MSB >= 0x60){   //si ha llegado a 60 minutos
      minuto_MSB = 0;         //se pone a 0 los minutos
    }
  }else minuto_LSB = 0x05;    //si no ha llegado a 5 se pone un 5

  minuto = minuto_MSB;          //se guardan los MSB
  minuto += minuto_LSB & 0x0F;  //se guardan los LSB

  escribir_reloj(Reloj_address,0x0B,minuto);  //se envia la prox alarma

  return 0;
}
```

14.1.2.7. Sensor CO2 y COV

Iniciar Sensor

```
bool iniciar_co2(){
  bool ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(co2_address); //se inicia comunicacion
    Wire.write(0xF4);                    //APP-START pasa de modo boot a aplicacion
    ACK = Wire.endTransmission();        //se guarda ACK
    delay(30);   //tarda 20ms en el peor de los casos

  ACK = 1;                              //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(co2_address); //se inicia comunicacion
    Wire.write(0x01);                    //Measure Mode register
    Wire.write(arrancar);                //se configura el arranque a la velocidad fijada
    ACK = Wire.endTransmission();        //se guarda en ACK si el esclavo
    delay(10);                                  //ha obtenido bien los datos
  }
  return ACK;
}
```

Nuevo dato disponible

```
bool co2_data_ready(){
  bool dato = 0;
  while(dato == 0){
    Wire.beginTransmission(co2_address);  //se inicia comunicacion
    Wire.write(0X00);                      //registro de estado
    Wire.endTransmission();
    delay(10);

    Wire.requestFrom(co2_address,1);     //se le solicita al esclavo 1 byte
    while(Wire.available()!=1);
    byte valor = Wire.read();            //se obtiene el byte completo
    dato = valor>>3 & 1;                 //se desplaza 3 posiciones a la izquierda
  }                                            //y se guarda el ultimo bit
  return dato;
}
```

Leer CO2 y COV

```
void leer_co2_cov(){
  bool ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                       //Mientras ACK sea 1
    Wire.beginTransmission(co2_address);  //se inicia comunicacion
    Wire.write(0x02);                    //Results data register
    ACK = Wire.endTransmission();        //se guarda en ACK si el esclavo
    delay(10);                               //ha obtenido bien los datos
  }

  Wire.requestFrom(co2_address,4);    //se le solicita al esclavo 4 bytes
  while(Wire.available()!=4);       //se espera hasta obtener los 4 bytes

  //se recoge el valor de CO2
  byte valor = Wire.read();         //se coge el primer byte
  co2 = valor<<8;                   //se desplaza 8 bits a la izk antes de guardar
  valor = Wire.read();              //se coge el segundo byte
  co2 += valor;                     //se guarda junto al primer byte recogido

  //se recoge el valor de COV
  valor = Wire.read();              //se coge el primer byte
  cov = valor<<8;                   //se desplaza 8 bits a la izk antes de guardar
  valor = Wire.read();              //se coge el segundo byte
  cov += valor;                     //se guarda junto al primer byte recogido

  return 0;
}
```

14.1.2.8. Sensor Luminosidad – LDR

```
int leer_LDR(void){
  float lectura = 0;      //variable de dato leido
  int lux = 0;            //variable en lumen

  digitalWrite(LDR_ON,HIGH);  //se alimenta el sensor
  delay(100);

  lectura = analogRead(LDR_IN)*Vcc/1023;      //se lee el divisor de tensión
  lux = int(250/(((Vcc*resistencia)/lectura)-resistencia));

  digitalWrite(LDR_ON,LOW);             //se apaga el sensor

  return lux;
}
```

14.1.2.9. Sensor Acústico

Prueba

```
valor = analogRead(micro_IN);
Serial.print(valor);
Serial.print("\t");
valor -= 265;
valor = int(valor/2.65);
valor = abs(valor);
Serial.println(valor);
delay(10);
```

Definitivo

```
void loop() {
  int lectura = 0,      //variable de dato leido
      convertido = 0;   //variable una vez convertida

  digitalWrite(micro_ON,HIGH);  //se enciende el sensor
  delay(100);

  for(int i=0; i<1000; i++){        //se recogen 1000 datos para realizar la medida
    lectura = analogRead(micro_IN); //se recoge el nuevo dato
    lectura -= 265;
    lectura = int(lectura/2.65);
    lectura= abs(lectura);          //se guarda el valor convertido 0:100
    if(lectura>=convertido){        //si la lectura es mayor al convertido
      convertido += lectura;        //se guarda la lectura
      convertido /= 2;              //se divide entre dos el valor convertido
    }                                       //antes de la siguiente comparacion
    delay(2);   //2ms entre cada lectura
  }
  convertido = int(0.5*convertido)+30;    //se usa la ecuacion de la recta calculada
  Serial.println(convertido);             //mostrar valor en dB
  digitalWrite(micro_ON,LOW);             //se apaga el sensor
  delay(1000);
}
```

14.1.2.10. Potencia

Configurar timer

```
// Configuracion del TIMER1

TCCR1A = 0;                         // Limpiar Registro A
TCCR1B = 0;                         // Limpiar Registro B
OCR1A = 0x3D09;                     // Valor fin de cuenta para 1 seg -> 15625 Hz -> Hexa 0X3D09
TCCR1B |= (1 << WGM12);             // Activar modo CTC
TCCR1B |= (1<<CS10)|(1 << CS12);    // Preescalador 1:1024 -> CS12 = 1 e CS10 = 1
TIMSK1 |= (1 << OCIE1A);            // Habilitar interrupción por comparación con registro A
```

Prueba de descarga de batería

```
ISR(TIMER1_COMPA_vect)          //Interrupcion con Timer 1
{
  if(seg==0&&minuto==0&&digitalRead(manual_pin)==1&&SD.begin(9)){
    logFile = SD.open("datalog.txt", FILE_WRITE);   //se abre SD
    while(!logFile);                                //se espera hasta que abra la SD
    logFile.println("INICIO");                      //se escribe inicio en el txt datalog
    logFile.close();                                //se cierra la SD
    digitalWrite(mosfet_pin,HIGH);    // Empezar descarga   //se alimenta el mosfet para cerrar el circuito
    digitalWrite(pin_verde,HIGH);     //Encendemos led      //se enciende el pin verde para indicar que está descargando
  }
  if(digitalRead(manual_pin)==1&&FIN==0)       //si el interruptor manual esta dado y no se han dado las condiciones de fin
  {
    seg++;                                      //se añade un segundo
    tension=analogRead(lectura_pin)/1024.*5.033;      //se lee la tensión
    if((tension-tension_ant<=-0.05||tension<=2.5))    //si cumple las condiciones de parada
    {
      logFile = SD.open("datalog.txt", FILE_WRITE);   //se abre el txt
      while(!logFile);                                //se espera a que abra
      logFile.print(tension,3);                       //se escribe el valor de tension
      logFile.print("\t");
      logFile.print(minuto);                          //se escribe el minuto
      logFile.print("\t");
      logFile.println(seg);                           //se escribe los segundos
      logFile.println("FIN");                         //se escribe FIN
      logFile.close();                                //se cierra el txt
      digitalWrite(mosfet_pin,LOW);                   //se abre el circuito y se para la descarga
      FIN=1;                                          //ha llegado al fin

       for(int i=0; i<100; i++){                          //durante 100 veces
         logFile = SD.open("datalog.txt", FILE_WRITE); //se abre el txt
         while(!logFile);
         tension=analogRead(lectura_pin)/1024.*5.033;  //se lee el valor de tension
         logFile.println(tension,3);                   //se escribe el valor de tension
         logFile.close();
       }
       digitalWrite(pin_verde,LOW);    //led rojo indica parada
       digitalWrite(pin_rojo,HIGH);    //se apaga el verde
    }
    if(seg>59)   //se han pasado 60 seg = 1 minuto
    {
      seg = 0;   //se reinician los segundos
      minuto++;  //se aumenta los minutos
      logFile = SD.open("datalog.txt", FILE_WRITE);  //se abre el txt
      while(!logFile);
      logFile.print(tension,3);    //se escribe el valor de tension
      logFile.print("\t");
      logFile.println(minuto);     //se escribe el minuto de ese valor
      logFile.close();
    }
    tension_ant=tension;     //se guarda en tension anterior el valor de tension
  }else digitalWrite(mosfet_pin,LOW);    // Para la descarga de forma manual
}
```

14.1.3. Código Programa Principal

Librerías y setup

```
/*  Weatherme - Proyecto Fin de Grado
 *
 *   Omar Fernandez Gallego
 *   Curso 2020/2021
 *   Grado en Ingeniería Electrónica Industrial y Automática - EMT
 *
 */

//Librerias
#include <Wire.h>
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <avr/sleep.h>

//RADIO
#define CE  10
#define CSN 9
RF24 radio(CE, CSN);
uint64_t pipe = 0xE8E8F0F0E1LL;

//Reloj
#define Reloj_address 0x68
#define Reloj 0x01            //este registro es el del minuto

//BME280
#define SS_BME 5  //pin selector esclavo
#define MOSI 11   //master OUT
#define MISO 12   //master IN
#define SCK 13    //pin reloj
#define velocidad_SPI 5000000 //velocidad comunicacion SPI
#define reg_HUM_config 0xF2   //registro control muestreo Humedad
#define reg_STATUS 0xF3       //registro de estado
#define reg_ctrl_meas 0xF4    //registro control muestreo y modo funcionamiento
#define reg_config 0xF5       //registro de configuracion
#define reg_RESET 0xE0        //registro de reseteo
#define reg_ID 0xD0           //registro de identificacion
//variables de calibración
uint16_t T1, P1;
int16_t T2, T3, P2, P3, P4, P5, P6, P7, P8, P9, H2, H4, H5;
uint8_t H1;
int8_t H3, H6;
int32_t t_fine = 0;

//Anemometro
#define anemometro_K 0.672 //radio del anemometro
#define factor 1.2  //factor de corrección
byte anem_IN = 7;     //pin entrada
    //se alimenta con 3.3V

//Veleta
byte veleta_IN = 6;
     //se alimenta a 5V (con un pin no tenia suficiente potencia)

//variables PowerDown
bool ENCENDIDO = 0;

//CO2 y COV
#define co2_address 0x5A
#define arrancar 0x40       //Recoge datos cada 250ms
#define ccs_wake_pin 5      //pin digital para arrancar modulo

//UV
byte UV_EN = 4,   //pin digital para encender (ENABLE)
     UV_IN = 0;   //pin analogico de lectura

//Sonido
byte micro_IN = 2,  //entrada analogica
     micro_ON = 6;  //pin digital de alimentacion

//LDR
#define resistencia 10
byte LDR_IN = 1,  //entrada analogica
     LDR_ON = 8;  //pin digital de alimentacion

//Lluvia
byte lluvia_IN = 3,  //entrada analogica
     lluvia_ON = 8;  //pin digital de alimentacion
```

```
//PowerDown
byte pin_encendido = 0; //luz indicadora de encendido
byte pin_bateria = 1; //luz indicadora de bateria baja

//variables globales
byte lectura [8];
#define Vcc 4.9

struct Struct{
  int FechaHora[6];   //dia de la semana, dia, mes, año, hora, minuto
  float Temperatura;
  float Presion;
  float Humedad;
  int Vel_viento;
  byte Sen_viento;
  int Lluvia;
  float UV;
  int Sonido;
  int Luminosidad;
  word CO2;
  word COV;
};

Struct WeatherMe;

void setup() {

  Wire.begin();          //se inicia el HW del bus
  SPI.begin();           //se inicia el HW del bus

  leer_reloj();   //se obtiene la hora actual
  ajustar_pcos_alarma();  //se ajusta en 5 mins
  pinMode(2,INPUT); //interrupción
  pinMode(sden_IN,INPUT); //evitar cortocircuitos
  powerDown();    //apagar
}
```

Loop principal

```
void loop() {

  if(ENCENDIDO==true){
    inicializar_sensores();
    delay(100);
    leer_parametros();
    delay(100);
    enviar_datos();
    delay(100);
    configurar_reloj();
    powerDown();
  }
}
```

14.1.3.1. Arrancar

```
ISR(INT0_vect){

  SMCR |= (0<<SE);    //se deshabilita el modo sleep
  EIMSK |= (0<<INT0);    //se inhabilita interrupciones externas en el INT0

  pinMode(pin_encendido,OUTPUT);
  digitalWrite(pin_encendido,HIGH);   //se enciende la luz indicadora de encendido

  pinMode(pin_bateria,OUTPUT);
  digitalWrite(pin_bateria,HIGH);   //se alimenta el AmpOp de la bateria

  ENCENDIDO = true;     //se coloca la variable a 1
}
```

14.1.3.2. Inicializar sensores

```
void inicializar_sensores(void){

  iniciar_BME();  //se inicia la configuración
  digitalWrite(UV_EN, LOW); //se pone a 0 para deshabilitar el sensor
  while(iniciar_CCS()!=0);  //se inicializa
  iniciar_RF(); //se configura
}
```

Iniciar BME

```
void iniciar_BME(void){

  pinMode(SCK, OUTPUT);
  pinMode(MOSI, OUTPUT);
  pinMode(MISO, INPUT);
  pinMode(SS_BME, OUTPUT);      //pin de seleccion
  digitalWrite(SS_BME, HIGH);   //mantenemos a alto

  escribir_registro_BME(reg_RESET,0xB6);
  delay(10);

  while(leer_registro_BME(reg_ID,1)!=0x60){   //se espera hasta que el modulo arranque
    escribir_registro_BME(reg_RESET,0xB6);
    delay(10);
  }

  leer_registro_BME(reg_STATUS,1);
  bool valor = lectura[0]>>3 & 1;
  while(valor==true){               //se espera hasta que termina de medir
    leer_registro_BME(reg_STATUS,1);
    valor = lectura[0]>>3 & 1;
  };
  delay(10);

  leer_dig();         //se leen los registros de calibracion

  //Se configura el oversampling a 8, realizará 8 mediciones de cada valor
  escribir_registro_BME(reg_HUM_config,0x04);
  escribir_registro_BME(reg_ctrl_meas,0xB0);     //Sleep mode
  escribir_registro_BME(reg_config,0x0C);        //standby 0.5ms, filter 8 y 4 wires
  delay(100);

  return 0;
}
```

Iniciar CCS

```
bool iniciar_CCS(){
  bool ACK = 1;                         //Se coloca el ACK a 1
  while(ACK == 1){                      //Mientras ACK sea 1
    Wire.beginTransmission(co2_address); //se inicia comunicacion
    Wire.write(0xF4);                   //APP-START pasa de modo boot a aplicacion
    ACK = Wire.endTransmission();       //se guarda ACK
    delay(30);   //tarda 20ms en el peor de los casos
  }
  ACK = 1;                          //Se coloca el ACK a 1
  while(ACK == 1){                  //Mientras ACK sea 1
    Wire.beginTransmission(co2_address); //se inicia comunicacion
    Wire.write(0x01);                   //Measure Mode register
    Wire.write(arrancar);               //se configura el arranque a la velocidad fijada
    ACK = Wire.endTransmission();       //se guarda en ACK si el esclavo
    delay(10);                              //ha obtenido bien los datos
  }
  return ACK;
}
```

Iniciar RF

```
void iniciar_RF(void){

  radio.begin();
  radio.setRetries(15, 15);
  radio.setPALevel(RF24_PA_MAX);
  radio.setDataRate(RF24_250KBPS);
  radio.openWritingPipe(pipe);
  radio.stopListening();

  return 0;
}
```

14.1.3.3. Leer sensores

```
void leer_parametros(void){

  leer_reloj();
  leer_BME();
  while(leer_v_viento()!=0);
  while(WeatherMe.Sen_viento==0){
    WeatherMe.Sen_viento=Sentido_viento();
  }
  intensidad_lluvia();
  leer_UV();
  leer_micro();
  leer_LDR();
  leer_CCS();
}
```

Leer Reloj

```
void leer_reloj(void){
  leer_i2c(Reloj_address,Reloj,6);

  WeatherMe.FechaHora[5] = Hex_a_Dec(lectura[0]);   //Minuto
  WeatherMe.FechaHora[4] = Hex_a_Dec(lectura[1]);   //Hora
  WeatherMe.FechaHora[0] = Hex_a_Dec(lectura[2]);   //Dia de la semana
  WeatherMe.FechaHora[1] = Hex_a_Dec(lectura[3]);   //Dia del mes
  WeatherMe.FechaHora[2] = Hex_a_Dec(lectura[4]);   //Mes
  WeatherMe.FechaHora[3] = Hex_a_Dec(lectura[5]);   //Año

  return 0;
}
```

Leer BME

```
void leer_BME(void){

  escribir_registro_BME(reg_ctrl_meas,0x92);    //modo normal
  delay(10);

  while(WeatherMe.Temperatura==0||WeatherMe.Temperatura<=-100){
    WeatherMe.Temperatura = leer_temp();     //Temperatura
  }
  while(WeatherMe.Presion==0){
    WeatherMe.Presion = leer_pres();         //Presion
  }
  while(WeatherMe.Humedad==0){
    WeatherMe.Humedad = leer_hum();          //Humedad
  }

  escribir_registro_BME(reg_ctrl_meas,0x90);    //vuelve a modo sleep

  return 0;
}
```

Leer Veleta

```
byte Sentido_viento(void){

  pinMode(veleta_IN,INPUT);
  delay(10);

  int veleta = analogRead(veleta_IN); //se lee el valor

  if(veleta > 100){
    if(veleta > 200){
      if(veleta > 300){
        if(veleta > 500){
          if(veleta > 700){
            if(veleta > 800){
              if(veleta > 900){
                return 3; //Este
              }else return 4; //Sureste
            }else return 5; //Sur
          }else return 2; //Noreste
        }else return 6; //Suroeste
      }else return 1; //Norte
    }else return 0; //Noroeste
  }else return 7; //Oeste

  return 0;  //error
}
```

Leer Anemómetro

```
bool leer_v_viento(void){

  pinMode(anem_IN,INPUT);

  int muestra = 0;      //numero de muestras tomadas
  word millis_ant = 0,  //se guarda el valor de millis anterior
       millis_act = 0,  //valor de millis actual
       anem_time = 0;   //tiempo de vuelta

  while(muestra<4){   //se toman 4 muestras (una vuelta completa)

    while(digitalRead(anem_IN) != 0);    //se espera a que este en bajo
    while(digitalRead(anem_IN) != 1);    //se espera a que esté en alto

    if(digitalRead(anem_IN) == 1){       //cuando detecta flanco entre
      millis_ant = millis();          //se recoge valor de millis

      while(digitalRead(anem_IN) == 1);  //se espera hasta el flanco de bajada

      if(digitalRead(anem_IN) == 0){     //se comprueba que esta a 0

        millis_act = millis();

        if(millis_act>millis_ant){     //se comprueba que no ha habido overflow
          anem_time += millis_act - millis_ant;

        }else return 1;  //error overflow
      }else return 1;    //error no ha vuelto a 0
    }else return 1;      //error no se ha detectado flanco de subida

   muestra++;
  }

  WeatherMe.Vel_viento=int((((2*3.14*anemometer_R*1000)/(anem_time)*3.6)*factor));  //Km/h
  delay(100);

  return 0;
}
```

Leer Lluvia

```
void intensidad_lluvia(void){

  pinMode(lluvia_ON,OUTPUT);
  pinMode(lluvia_IN,INPUT);

  byte intensidad = 0;                  //byte de respuesta

  digitalWrite(lluvia_ON,HIGH);         //se enciende el módulo
  delay(10);                             //se espera un poco a que arranque

  int lluvia = analogRead(lluvia_IN);   //se lee la entrada
  if(lluvia<700){
    if(lluvia<300){
      intensidad = 3;                   //si es inferior a 300: lluvia intensa
    }else intensidad = 2;               //si 300<=x<700: lluvia moderada
  }else intensidad = 1;                 //si es superior a 700: no llueve

  digitalWrite(lluvia_ON,LOW);          //se apaga el módulo
  delay(10);

  WeatherMe.Lluvia=intensidad;

  return 0;
}
```

Leer UV

```
void  leer_UV (void){

  pinMode (UV_EN , OUTPUT ); //como salida

  float  UV = 0;

  digitalWrite  (UV_EN , HIGH );   //se habilita el sensor
  delay (30);
  float  tension  = analogRead  (UV_IN ) * Vcc/1023 ; //se calcula equiv de V
  UV = (tension -1) * (15./(2.9-1)); //se realizan los cálculos
  if(UV<=0) UV = 0;        //demasiado oscuro y solo detecta ruido

  WeatherMe .UV = int (UV); //se guarda el valor enter

  digitalWrite  (UV_EN , LOW);    //se apaga el sensor
  return  0;
}
```

Leer Micro

```
bool leer_micro(void){

  pinMode(micro_ON,OUTPUT);

  int lectura = 0,      //variable de dato leido
      convertido = 0;   //variable una vez convertida

  digitalWrite(micro_ON,HIGH);  //se enciende el sensor
  delay(100);

  for(int i=0; i<500; i++){       //se recogen 500 datos para realizar la medida
    lectura = analogRead(micro_IN); //se recoge el nuevo dato
    lectura -= 265;
    lectura = int(lectura/2.65);
    lectura= abs(lectura);          //se guarda el valor convertido 0:100
    if(lectura>convertido){         //si la lectura es mayor al convertido
      convertido += lectura;        //se guarda la lectura
      convertido /= 2;              //se divide entre dos el valor convertido
    }                                 //antes de la siguiente comparacion
    delay(2);   //2ms entre cada lectura
  }
  convertido = int(0.5*convertido)+30;    //se usa la ecuacion de la recta calculada
  WeatherMe.Sonido=convertido;
  digitalWrite(micro_ON,LOW);             //se apaga el sensor

  return 0;
}
```

Leer LDR

```
void leer_LDR(void){

  pinMode(LDR_ON,OUTPUT);

  float lectura = 0;      //variable de dato leido
  int lux = 0;            //variable en lumen

  digitalWrite(LDR_ON,HIGH);  //se alimenta el sensor
  delay(100);

  lectura = analogRead(LDR_IN)*Vcc/1023;     //se lee el divisor de tensión
  lux = int(250/((Vcc*resistencia)/lectura)-resistencia));

  digitalWrite(LDR_ON,LOW);             //se apaga el sensor

  WeatherMe.Luminosidad=lux;

  return 0;
}
```

Leer CCS

```
bool  leer_CCS (void){

  digitalWrite  (ccs_wake_pin  , LOW);

  while (iniciar_CCS  ()!=0);
  delay (2000);

  while (lcca_co2_cov  ()!=0);

  digitalWrite  (ccs_wake_pin  , HIGH);

}
```

14.1.3.4. Enviar datos

```
void enviar_datos(void){

  radio.write(&WeatherMe, sizeof(WeatherMe));
  delay(100);

  return 0;
}
```

14.1.3.5. Configurar Reloj

```
void configurar_reloj(void){
  configurar_alarma_reloj();
  ajustar_prox_alarma();
}
```

Configurar alarma

```
bool configurar_alarma_reloj(void){

  bool ACK = 1;                                 //Se coloca el ACK a 1
  while(ACK == 1){                              //Mientras ACK sea 1
    Wire.beginTransmission(Reloj_address);      //se inicia comunicacion
    Wire.write(0x0B);                           //primer registro de la alarma
    Wire.write(0x00);                           //se coloca A2M2 a 1, registro(0Bh)
    Wire.write(0x80);                           //se coloca A2M3 a 1, registro(0Ch)
    Wire.write(0x80);                           //se coloca A2M4 a 1, registro(0Dh)
    Wire.write(0x1E);                           //se coloca A2IE a 1, registro(0Eh)
    ACK = Wire.endTransmission();               //se guarda en ACK si el esclavo
    delay(10);                                      //ha obtenido bien los datos
  }
  return ACK;
}
```

Ajustar próxima alarma

```
void ajustar_prox_alarma(void){

  limpiar_flag_reloj();

  byte minuto = leer_i2c(Reloj_address,0x01,1); //se lee el min actual

  byte minuto_MSB = (minuto & 0xF0);      //se guardan los 4 bits mas significativos
  byte minuto_LSB = minuto & 0x0F;        //se guardan los 4 bits menos significativos

  if(minuto_LSB >= 0x05){     //si el minuto es mayor o igual a 5
    minuto_LSB = 0;           //se pone a 0
    minuto_MSB += 0x10;       //se suman 10 minutos
    if(minuto_MSB >= 0x60){   //si ha llegado a 60 minutos
      minuto_MSB = 0;         //se pone a 0 los minutos
    }
  }else minuto_LSB = 0x05;    //si no ha llegado a 5 se pone un 5

  minuto = minuto_MSB;          //se guardan los MSB
  minuto += minuto_LSB & 0x0F;  //se guardan los LSB

  escribir_i2c(Reloj_address,0x0B,minuto);  //se envia la prox alarma

  return 0;
}
```

Limpiar flag

```
bool limpiar_flag_reloj(void){

  bool ACK = 1;                                 //Se coloca el ACK a 1
  while(ACK == 1){                              //Mientras ACK sea 1
    Wire.beginTransmission(Reloj_address);      //se inicia comunicacion
    Wire.write(0x0F);                           //Status Register
    Wire.write(0x88);                           //limpiar flag
    ACK = Wire.endTransmission();               //se guarda en ACK si el esclavo
    delay(10);                                      //ha obtenido bien los datos
  }
  return ACK;
}
```

14.1.3.6. PowerDown – Apagado

```
void powerDown(void){

  borrar_valores(); //coloca todas las variables a 0

  pinMode(pin_encendido,OUTPUT);
  pinMode(pin_bateria,OUTPUT);
  digitalWrite(0,LOW);    //se apaga el led indicador de encendido
  digitalWrite(pin_bateria,LOW);   //se alimenta el AmpOp de la bateria
  ENCENDIDO = false;
  delay(5000);  //se espera un poco a que termine de ejecutar las acciones anteriores
  EIMSK |= (1<<INT0);         //habilitar interrupciones externas en el INT0
  EICRA |= (1<<ISC01) | (0<<ISC00);   //interrupcion en el flanco de bajada
  SMCR |= (1<<SE);  //se habilita el modo sleep
  SMCR |= (0<<SM2) | (1<<SM1) | (0<<SM0); //se configura el PowerDown en el selector de modo
  sleep_cpu();    //se ejecuta

}
```

14.1.3.7. Funciones Extra

Borrar valores

```
void borrar_valores(void){

  for(int i=0; i<6;i++){
      WeatherMe.FechaHora[i] = 0;
    }
  WeatherMe.Temperatura = 0;
  WeatherMe.Presion = 0;
  WeatherMe.Humedad = 0;
  WeatherMe.Vel_viento = 0;
  WeatherMe.Sen_viento = 0;
  WeatherMe.Lluvia = 0;
  WeatherMe.UV = 0;
  WeatherMe.Sonido = 0;
  WeatherMe.Luminosidad = 0;
  WeatherMe.CO2 = 0;
  WeatherMe.COV = 0;
}
```

De hexadecimal a decimal

```
int Hex_a_Dec(byte Hex){   //funcion que convierte valor Hexadecimal a Decimal
  byte Dec = 0;
  Dec = Hex & 0x0F;
  Dec += (Hex >> 4) *10;

  return Dec;
}
```

Dos a uno

```
word dos_a_uno(bool invertido){     //invertido: MSB está el [1] del array, se ha leido despues
  word unido = 0;

  if(invertido==true){
    unido += lectura[1];                //se guarda el byte MSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[0];                //se guarda el byte LSB
  }else{
    unido += lectura[0];                //se guarda el byte MSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[1];                //se guarda el byte LSB
  }
  return unido;
}
```

Tres a uno

```
long tres_a_uno(bool invertido){     //invertido: MSB está el [3] del array, se ha leido despues
  long unido = 0x00000;

  if(invertido==true){
    unido += lectura[2];                //se guarda el byte MSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[1];                //se guarda el byte LSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[0];                //se guarda el byte XLSB

  }else{
    unido += lectura[0];                //se guarda el byte MSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[1];                //se guarda el byte LSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[2];                //se guarda el byte XLSB
  }
  return unido;
}
```

14.1.4. Código Modulo Interior

Configuración y bucle esclavo

```
#include <Wire.h>
#include <RF24.h>
#include <nRF24L01.h>

//RADIO
#define CE  10
#define CSN 9
RF24 radio(CE, CSN);
uint64_t pipe = 0xE8E8F0F0E1LL;

//I2C
#define master 71
#define slave 72
int16_t dato = 0;

struct Struct{
  int FechaHora[6];   //dia de la semana, dia, mes, año, hora, minuto
  float Temperatura;
  float Presion;
  float Humedad;
  int Vel_viento;
  byte Sen_viento;
  int Lluvia;
  float UV;
  int Sonido;
  int Luminosidad;
  word CO2;
  word COV;
};

Struct WeatherMe;

void setup() {
  Wire.begin(slave);
  Wire.onRequest(requestEvent);
  Wire.onReceive(receiveEvent);
  iniciar_radio();
}

void loop() {

  while(leer_radio()==0);
  esta_despierto();

}
```

Configuración y bucle maestro

```
#include <SD.h>
#include <SPI.h>
#include <Wire.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <U8g2lib.h>

//I2C
#define master 71
#define slave 72

bool recibiendo = false;
byte lectura[2] = {0};

//Display
U8G2_ST7920_128X64_F_SW_SPI u8g2(U8G2_R0, /* clock=*/ 13, /* data=*/ 11, /* CS=*/ 8, /* reset=*/ 6);
#define display_ON_pin 12   //pin para encender la pantalla del display
byte n_pantalla = 0;          //pin para seleccionar entre las pantallas

//SD
const byte SD_ON_pin = 5;      //pin para arrancar la SD
const byte SD_CE_pin = 53;   //pin selector esclavo SD
String string_datos = "";   //string para almacenar y leer los datos de la SD
String nombre_fichero = ""; //nombre del fichero CSV
```

```
struct Struct{
  int FechaHora[6];   //dia de la semana, dia, mes, año, hora, minuto
  float Temperatura;
  float Presion;
  float Humedad;
  int Vel_viento;
  byte Sen_viento;
  byte Lluvia;
  float UV;
  int Sonido;
  int Luminosidad;
  word CO2;
  word COV;
};

Struct WeatherMe;

void setup() {
  iniciar_display();
  iniciar_SD();
  Wire.begin(master);           // se une al bus como maestro
  Wire.onRequest(requestEvent); //funcion de evento de solicitud
}

void loop() {

  if(recibiendo){
    salvapantallas();
    for(byte i=0x01; i<=0x11; i++){
      solicitar(i);
      convertir_y_guardar(i);
      delay(20);
    }
      if(WeatherMe.FechaHora[1]==0&&WeatherMe.FechaHora[0]==0){ //nuevo dia
        crear_string();  //se crea el archivo nuevo
      }
    recibiendo = false;
  }else{
    Serial.println("NO");
    //word millis_ant=millis();
    pantalla(n_pantalla);
    delay(5000);
    n_pantalla++;
    if(n_pantalla>3) n_pantalla=0;
  }
}
```

14.1.4.1. Recepción Radiofrecuencia

Configuración Radio

```
void iniciar_radio(void){
  radio.begin();
  radio.setRetries(15,15);          //nº de repeticiones y tiempo entre repeticiones
  radio.setPALevel(RF24_PA_MAX);    //potencia de la señal
  radio.setDataRate(RF24_250KBPS);  //frecuencia
  radio.openReadingPipe(1,pipe);    //se inicia como lectura
  radio.startListening();           //empezar escucha
}
```

Escucha y guardado

```
bool leer_radio(void){

  if (radio.available()){   //si hay nuevo dato
    radio.read(&WeatherMe, sizeof(WeatherMe));
    return 1;
    }
  return 0;
}
```

14.1.4.2. Comunicación I2C

Inicio de la comunicación (esclavo)

```
bool esta_despierto(void){

  Wire.requestFrom(master, 1);    // Solicitar 1 byte al maestro

  while (Wire.available()!=1);     //se espera hasta que el esclavo tire de la linea
  if(Wire.read()=='O'){
    if(Wire.read()=='K') return 1; //si responde OK está escuchando
  }
  return 0;   //la respuesta no ha sido correcta
}
```

Respuesta del maestro

```
void requestEvent() {
  Wire.write("OK");  //contesta que está escuchando
  delay(200);
  recibiendo=true;
}
```

Maestro solicita datos

```
void solicitar(char parametro){

  int i = 0;

  Wire.beginTransmission(slave); // Comunicarse con esclavo
  Wire.write(parametro);          //dato que se solicita
  Wire.endTransmission();

  Wire.requestFrom(slave, 2);    // Solicitar 2 bytes al esclavo

  while(Wire.available()){      //mientras haya datos
    lectura[i] = Wire.read();   //se guardan los datos
    i++;
  }
  return 0;
}
```

Esclavo recibe parámetro y prepara datos

```
void receiveEvent(){
  char valor = Wire.read();
    if( valor == 0x01 ) dato = int(WeatherMe.FechaHora[0]);      //Dia de la semana
    if( valor == 0x02 ) dato = int(WeatherMe.FechaHora[1]);      //Dia del mes
    if( valor == 0x03 ) dato = int(WeatherMe.FechaHora[2]);      //Mes
    if( valor == 0x04 ) dato = int(WeatherMe.FechaHora[3]);      //Año
    if( valor == 0x05 ) dato = int(WeatherMe.FechaHora[4]);      //Hora
    if( valor == 0x06 ) dato = int(WeatherMe.FechaHora[5]);      //Minuto
    if( valor == 0x07 ) dato = int(WeatherMe.Temperatura*100.); //Temperatura
    if( valor == 0x08 ) dato = int(WeatherMe.Presion*10.);       //Presion
    if( valor == 0x09 ) dato = int(WeatherMe.Humedad*100.);      //Humedad
    if( valor == 0x0A ) dato = int(WeatherMe.Vel_viento);   //Velocidad del Viento
    if( valor == 0x0B ) dato = int(WeatherMe.Sen_viento);   //Sentido del viento
    if( valor == 0x0C ) dato = int(WeatherMe.Lluvia);       //Lluvia
    if( valor == 0x0D ) dato = int(WeatherMe.UV*100.);          //Indice UV
    if( valor == 0x0E ) dato = int(WeatherMe.Sonido);       //Sonido
    if( valor == 0x0F ) dato = int(WeatherMe.Luminosidad); //Luminosidad
    if( valor == 0x10 ) dato = int(WeatherMe.CO2);          //CO2
    if( valor == 0x11 ) dato = int(WeatherMe.COV);          //COV
}
```

Esclavo envía dato solicitado

```
void requestEvent() {
  byte uno = (dato >> 8) & 0xFF;     //byte mas significativo
  byte dos = dato & 0xFF;            //byte menos significativo
  Wire.write(uno);
  Wire.write(dos);
}
```

Maestro une y reconstruye dato

```
int convertir_y_guardar(byte parametro){

  int unido = 0;

    unido += lectura[0];                //se guarda el byte MSB
    unido <<= 8;                        //se desplaza a la izq para dejar hueco para el siguiente byte
    unido += lectura[1];                //se guarda el byte LSB

  switch(parametro){
    case 0x01: WeatherMe.FechaHora[0] = unido;      //Dia de la semana
    case 0x02: WeatherMe.FechaHora[1] = unido;      //Dia del mes
    case 0x03: WeatherMe.FechaHora[2] = unido;      //Mes
    case 0x04: WeatherMe.FechaHora[3] = unido;      //Año
    case 0x05: WeatherMe.FechaHora[4] = unido;      //Hora
    case 0x06: WeatherMe.FechaHora[5] = unido;      //Minuto
    case 0x07: WeatherMe.Temperatura = float(unido/100.);
    case 0x08: WeatherMe.Presion = float(unido/10.);
    case 0x09: WeatherMe.Humedad = float(unido/100.);
    case 0x0A: WeatherMe.Vel_viento = unido;
    case 0x0B: WeatherMe.Sen_viento = unido;
    case 0x0C: WeatherMe.Lluvia = unido;
    case 0x0D: WeatherMe.UV = float(unido/100.);
    case 0x0E: WeatherMe.Sonido = unido;
    case 0x0F: WeatherMe.Luminosidad = unido;
    case 0x10: WeatherMe.CO2 = unido;
    case 0x11: WeatherMe.COV = unido;
  }
}
```

14.1.4.3. Escritura SD

Iniciar y detener SD

```
void iniciar_SD(void){
  pinMode(SD_ON_pin,OUTPUT);  //se fija el pin como salida
  digitalWrite(SD_ON_pin,HIGH); //se pone a 5V la salida
  delay(10);
  Serial.print(F("Iniciando SD ..."));
  while (!SD.begin(SD_CE_pin))
  {
    Serial.println(F("Error al iniciar"));
    delay(10);
  }
  Serial.println(F("Iniciado correctamente"));
}

void detener_SD(void){
  digitalWrite(SD_ON_pin,LOW);  //se pone a 0V la salida
  pinMode(SD_ON_pin,INPUT);     //se configura como entrada para aumentar la impedancia
}
```

Crear fichero

```
void crear_fichero(void){

  nombre_fichero = String(WeatherMe.FechaHora[3]) + "_" +                 //Año
                   String(WeatherMe.FechaHora[4]) + "_" +                 //Mes
                   String(WeatherMe.FechaHora[2]) + ".csv";               //Dia


  File fichero = SD.open(nombre_fichero, FILE_WRITE);

  while (!fichero);     //se espera hasta abrir el fichero
  String header = "Dia,Hora,Minuto,Temperatura,Presion,Humedad,Velocidad del viento,Sentido del viento,Lluvia,Indice UV,Sonido,Luminosidad,CO2,COV";
  fichero.println(header);
  fichero.close();

}
```

Crear string

```
void crear_string(void){

  nombre_fichero = String(dia_de_la_semana(WeatherMe.FechaHora[2])) + "_" +  //Dia de la semana
                   String(WeatherMe.FechaHora[3]) + "_" +                    //Dia del mes
                   String(WeatherMe.FechaHora[4]) + "_" +                    //Mes
                   String(WeatherMe.FechaHora[5]);                           //Año

  string_datos = String(WeatherMe.FechaHora[1]) + "," +   //Hora
                 String(WeatherMe.FechaHora[0]) + "," +   //Minuto
                 String(WeatherMe.Temperatura) + "," +
                 String(WeatherMe.Presion) + "," +
                 String(WeatherMe.Humedad) + "," +
                 String(WeatherMe.Vel_viento) + "," +
                 String(sentido_viento(WeatherMe.Sen_viento)) + "," +
                 String(cuanto_llueve(WeatherMe.Lluvia)) + "," +
                 String(WeatherMe.UV) + "," +
                 String(WeatherMe.Sonido) + "," +
                 String(WeatherMe.Luminosidad) + "," +
                 String(WeatherMe.CO2) + "," +
                 String(WeatherMe.COV);
}
```

Guardar datos

```
void guardar_datos(void){

  File datos;

  datos = SD.open(nombre_fichero, FILE_WRITE);

  while (!datos);     //se espera hasta abrir el fichero
  datos.println(string_datos);    //se escriben los datos
  datos.print("\n");
  datos.close();      // se cierra el fichero

}
```

14.1.4.4. Pantalla

Iniciar Display

```
void iniciar_display(void){
  u8g2.begin();
  u8g2.enableUTF8Print();
  pinMode(display_ON_pin,OUTPUT);
  digitalWrite(display_ON_pin,HIGH);
  pinMode(display_CE_pin,OUTPUT);
  digitalWrite(display_CE_pin,HIGH);
  delay(100);
  //Más fuentes en: https://github.com/olikraus/u8g2/wiki/fntlistall
}
```

Ahora

```
void ahora(void){

  u8g2.clearBuffer(); // se limpia la pantalla
  u8g2.setFont(u8g2_font_baby_tf);  //se fija la fuente

  u8g2.setCursor(0, 6);
  mostrar_dia_de_la_semana(WeatherMe.FechaHora[2]); //funcion que convierte numero en dia

  u8g2.print(",  ");
  u8g2.print(WeatherMe.FechaHora[3]);   //dia del mes
  u8g2.print("/");
  u8g2.print(WeatherMe.FechaHora[4]);   //mes
  u8g2.print("/");
  u8g2.print(WeatherMe.FechaHora[5]);   //año

  u8g2.setCursor(104, 6);
  if(WeatherMe.FechaHora[1]<10) u8g2.print(0);  //para mostrar un 06 en vez de 6
  u8g2.print(WeatherMe.FechaHora[1]);   //hora
  u8g2.print(":");
  if(WeatherMe.FechaHora[0]<10) u8g2.print(0);
  u8g2.print(WeatherMe.FechaHora[0]);   //minuto

  u8g2.drawLine(0, 8, 127, 8);      //se dibujan dos lineas para separar
  u8g2.drawLine(0, 10, 127, 10);

  u8g2.sendBuffer();
}
```

Pantallas

```
void pantalla(byte n_pantalla){

  u8g2.clearBuffer();
  switch(n_pantalla){
    case 0: salvapantallas(); break;
    case 1: pantalla_uno(); break;
    case 2: pantalla_dos(); break;
    case 3: pantalla_tres(); break;
  }
}
```

Salvapantallas

```
void salvapantallas(void){
  u8g2.clearBuffer();          // Limpia la memoria interna
  u8g2.setFont(u8g2_font_crox5tb_tf);  // Más fuentes en: https://github.com/olikraus/u8g2/wiki/fntlistall
  u8g2.setCursor(5,22);
  u8g2.print("WeatherMe");
  //u8g2.drawStr(5,22,"WeatherMe");
  u8g2.setFont(u8g2_font_open_iconic_weather_4x_t);  // Más fuentes en: https://github.com/olikraus/u8g2/wiki/fntlistall
  u8g2.drawGlyph(5,60,random(64,69));//sun 69 suncloud 65 CLOUD 64 RAIN 66 THUNDER 67 UMBRELLA 68
  u8g2.setFont(u8g2_font_baby_tf);
  u8g2.setCursor(50,47);
  u8g2.print("UN PROYECTO DE");
  u8g2.setCursor(50,55);
  u8g2.print("UNAI FERNANDEZ");
  u8g2.sendBuffer();
}
```

Pantalla uno

```
void pantalla_uno(void){

  ahora();

  byte x = 2;
  byte y = 16;
  byte sx = 9;  //separacion en x
  byte sy = 10;  //separacion en y
  byte a = 4;   //bits de anchura
  byte h = 4;   //bits de altura

  u8g2.drawFrame(x,y,a,h);
  u8g2.drawFrame(x,y+(1*sy),a,h);
  u8g2.drawFrame(x,y+(2*sy),a,h);
  u8g2.drawFrame(x,y+(3*sy),a,h);

  y = 20;
  u8g2.setCursor(x+sx,y);
  u8g2.print("TEMPERATURA: ");
  u8g2.print(WeatherMe.Temperatura);
  u8g2.print(" °C");

  u8g2.setCursor(x+sx,y+(1*sy));
  u8g2.print("PRESION: ");
  u8g2.print(WeatherMe.Presion);
  u8g2.print(" hPa");

  u8g2.setCursor(x+sx,y+(2*sy));
  u8g2.print("HUMEDAD: ");
  u8g2.print(WeatherMe.Humedad);
  u8g2.print(" %");

  u8g2.setCursor(x+sx,y+(3*sy));
  u8g2.print("VIENTO: ");
  u8g2.print(WeatherMe.Val_viento);
  u8g2.print(" m/s   ");
  mostrar_rumbo_viento(WeatherMe.Dir_viento);

  u8g2.sendBuffer();
}
```

Pantalla dos

```
void pantalla_dos(void){

  ahora();

  byte x = 2;
  byte y = 16;
  byte sx = 9;  //separacion en x
  byte sy = 10;  //separacion en y
  byte a = 4;   //bits de anchura
  byte h = 4;   //bits de altura

  u8g2.drawFrame(x,y,a,h);
  u8g2.drawFrame(x,y+(1*sy),a,h);
  u8g2.drawFrame(x,y+(2*sy),a,h);
  u8g2.drawFrame(x,y+(3*sy),a,h);

  y = 20;
  u8g2.setCursor(x+sx,y);
  u8g2.print("LUMINOSIDAD: ");
  u8g2.print(WeatherMe.Luminosidad);
  u8g2.print(" lux");

  u8g2.setCursor(x+sx,y+(1*sy));
  u8g2.print("SONIDO: ");
  u8g2.print(WeatherMe.Sonido);
  u8g2.print(" dB");

  u8g2.setCursor(x+sx,y+(2*sy));
  u8g2.print("INDICE - UV: ");
  u8g2.print(WeatherMe.UV);
  u8g2.print(" mW/m2");

  u8g2.setCursor(x+sx,y+(3*sy));
  mostrar_cuanto_llueve(WeatherMe.Lluvia);

  u8g2.sendBuffer();
}
```

Pantalla tres

```
void pantalla_tres(void){

  ahora();

  byte x = 2;
  byte y = 29;
  byte ax = 9;  //separacion en x
  byte ay = 13;  //separacion en y
  byte a = 4;   //bits de anchura
  byte h = 4;   //bits de altura

  u8g2.drawFrame(x,y,a,h);
  u8g2.drawFrame(x,y+(1*ay),a,h);

  y = 20;
  u8g2.setCursor(5,y);
  u8g2.print("CONTAMINACION ");
  u8g2.print("ATMOSFERICA");

  u8g2.setCursor(x+ax,y+(1*ay));
  u8g2.print("CO2: ");
  u8g2.print(WeatherMx.CO2);
  u8g2.print(" ppm");

  u8g2.setCursor(x+ax,y+(2*ay));
  u8g2.print("COV: ");
  u8g2.print(WeatherMx.COV);
  u8g2.print(" ppb");

  u8g2.sendBuffer();
}
```

Funciones extra

```
void mostrar_dia_de_la_semana(byte dia){
  switch(dia){
    case 1: u8g2.print("LUNES"); break;
    case 2: u8g2.print("MARTES"); break;
    case 3: u8g2.print("MIERCOLES"); break;
    case 4: u8g2.print("JUEVES"); break;
    case 5: u8g2.print("VIERNES"); break;
    case 6: u8g2.print("SABADO"); break;
    case 7: u8g2.print("DOMINGO"); break;
  }
  return 0;
}

void mostrar_sentido_viento(byte valor){
  switch(valor){
    case 1: u8g2.print("NORTE");  break;
    case 2: u8g2.print("NORESTE");  break;
    case 3: u8g2.print("ESTE");  break;
    case 4: u8g2.print("SURESTE"); break;
    case 5: u8g2.print("SUR");  break;
    case 6: u8g2.print("SUROESTE");  break;
    case 7: u8g2.print("OESTE");  break;
    case 8: u8g2.print("NOROESTE");  break;
  }
  return 0;
}

void mostrar_cuanto_llueve(byte intensidad){
  switch(intensidad){
    case 1: u8g2.print("NO "); u8g2.print("LLUEVE"); break;
    case 2: u8g2.print("LLUVIA "); u8g2.print("MODERADA"); break;
    case 3: u8g2.print("LLUVIA "); u8g2.print("INTENSA"); break;
  }
  return 0;
}
```