Universidad de Valladolid

E.T.S.I. Telecomunicación

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías Especificas de telecomunicación, Mención en Sistemas Electrónicos

Desarrollo de una librería para el microcontrolador STM32 para sensores relacionados con la monitorización del agua

Autor:
D. Jesús Arribas López

Tutor:
D. Jesús Arias Álvarez

Título:	**Desarrollo de una librería para el microcontrolador STM32 para sensores relacionados con la monitorización del agua.**
Autor:	**D. Jesús Arribas López**
Tutor:	**D. Jesús Arias Álvarez**
Departamento:	**Departamento de Electrónica**

Tribunal	
Presidente:	**Dña. Lourdes Pelaz Montes**
Vocal:	**D. Jesús Arias Álvarez**
Secretario:	**D. Héctor García García**
Suplente 1:	**D. Jesús Manuel Hernández Mangas**
Suplente 2:	**D. José Emiliano Rubio**

Fecha:	
Calificación:	

Resumen

En este Trabajo de Fin de Grado, se llevará a cabo la tarea de desarrollar una **librería de funciones** para el control de **sensores ambientales** incorporados en una estación de monitorización del agua basada en un **microcontrolador STM32**. Estos sensores son **WSEN-PADS, WSEN-HIDS, URM06 y AS7341.** Cada uno dedicado a medir diferentes parámetros que sirven como apoyo a la estación para realizar un análisis de los parámetros del agua relacionados con su grado de contaminación y calidad. En concreto estos parámetros son los siguientes: **presión atmosférica**, **temperatura**, **distancia al agua** y **luminosidad** en un amplio rango espectral.

El objetivo del proyecto será realizar una librería capaz de obtener el dato de cada sensor respetando las indicaciones y requisitos del proyecto en el que esta tarea se engloba. Uno de los principales desafíos será integrar el código desarrollado en el ya existente, ya que la estación tiene cierto grado de desarrollo. Por lo tanto, además del desarrollo de estas funciones se deberá hacer respetando el ciclo de trabajo que ya posee la estación.

Palabras cable

Microcontrolador, STM32, WSEN-PADS, WSEN-HIDS, URM06, AS7341, monitorización del agua, sensores ambientales, MEMS, I2C.

Abstract

In this final degree project, the task of developing a library of functions for the control of environmental sensors incorporated in a water monitoring station based on an STM32 microcontroller will be carried out. These sensors are **WSEN-PADS**, **WSEN-HIDS**, **URM06** and **AS7341**. Each one dedicated to measuring different parameters that serve as support to the station to carry out an analysis of water parameters related to its degree of contamination and quality. Specifically, these parameters are the following: **atmospheric pressure**, **temperature**, **distance from water** and l**uminosity** in a wide spectral range.

The objective of the project will be to create a library capable of obtaining data from each sensor, respecting the indications and requirements of the project in which this task is included. One of the main challenges will be to integrate the developed code into the existing one, since the station has a certain degree of development. Therefore, in addition to the development of these functions, it must be done respecting the work cycle that the station already has.

Keywords

Microcontroller, STM32, WSEN-PADS, WSEN-HIDS, URM06, AS7341, water monitoring, environmental sensors, MEMS, I2C.

1. Introducción.

El presente proyecto está centrado en el desarrollo de una librería de funciones, capaz de manejar correctamente las diferentes funcionalidades de varios sensores ambientales instalados en una estación de monitorización del agua. Debido a las necesidades que presenta este proyecto, estos sensores deben realizar sus funciones integrados de manera óptima en el ciclo te trabajo que realiza la estación. Por ello, comenzare haciendo un pequeño resumen de en qué consiste este proyecto y los objetivos de este.

Aquacorp S.L [1] es una empresa situada en León, creada con el objetivo de diseñar una estación capaz de obtener parámetros fisicoquímicos y biológicos del agua. Lo que diferencia a esta tecnología de otras diseñadas actualmente con este objetivo es que estos parámetros se obtendrán a partir de imágenes superficiales del agua mediante procesos de visión por computador. La estación, por lo tanto, está compuesta de una cámara encargada de tomar las imágenes que son la base de la obtención de los parámetros. Además, la estación incorpora numerosos sensores que toman parámetros ambientales del entorno en el que se encuentra la estación y que se utilizan para apoyar a la IA encargada de extraer los parámetros del agua de las imágenes. Estos parámetros ambientales son **humedad, temperatura ambiente, distancia al agua, multiespectral, luminosidad y presión atmosférica**. Además, la estación consta de otros sensores enfocados en la adquisición de parámetros utilizados para el correcto funcionamiento de esta. Como un **acelerómetro** y un **sensor de temperatura interior**.

Debido a estas necesidades que presenta la estación, la integración de las funciones de estos sensores debe realizarse de forma que no interfiera con el resto de los procesos que la estación debe realizar. Además, la toma de los parámetros se debe realizar en tiempo y forma. En tiempo, porque deben tomarse en el instante adecuado para que estos parámetros sean de utilidad a la hora de que la IA analice las imágenes. Y en forma, porque deben tomarse en las unidades correctas y bajo las indicaciones que el equipo de IA nos indica que son las adecuadas.

Por lo tanto, el presente proyecto no solo consta del desarrollo de una librería de funciones para los mencionados sensores ambientales. Sino también de que estas funciones sean desarrolladas de una forma óptima, teniendo en cuenta el contexto en el que se van a encontrar.

El marco sobre el que se deben desarrollar estas funciones lo define el microprocesador que utiliza la estación [2], perteneciente a la marca ST. Este micro de 32 bits contiene muchos módulos destinados a comunicarse con dispositivos externos. Estos módulos son los comunes que se suelen encontrar en estos dispositivos: I2C, UART, SPI, USB. En el cuerpo del documento se detalla que tipo de conexión utiliza cada sensor, siendo la más empleada el I2C, pero también se emplea un pin de propósito general GPIO.

Cabe destacar que el propósito de este TFG es el desarrollo de una librería, por lo tanto, no se entrará en valoraciones de funcionalidad y selección de sensores. Estos ya han sido probados y elegidos según las conveniencias del proyecto.

En este marco el objetivo de este proyecto estará centrado en el diseño de una librería capaz de obtener los siguientes parámetros**: humedad, temperatura ambiente, luz multiespectral, presión atmosférica y distancia al agua**.

Para ello se emplearán los siguientes sensores:

- WSEN-HIDS. Humedad y temperatura.
- WSEN-PADS. Presión y temperatura.
- AS7341. Luz multiespectral.
- URM06. Distancia al agua.

La estructura del documento de este proyecto está dividida en las siguientes partes:

1. Introducción.
2. Estado del arte.
3. Análisis del problema.
4. Diseño de la solución.
5. Implementación.
6. Resultados.
7. Conclusiones.
8. Bibliografía.

2. Estado del arte

En la actualidad, con la llegada de las tecnologías IoT, numerosos dispositivos y proyectos han sido diseñados para que mediante una conexión a internet y una determinada inteligencia software estos sean capaces de medir parámetros físicos o realizar remotamente algunos procesos. En el caso que nos ocupa nos centraremos en analizar los dispositivos diseñados para medir ciertos parámetros ambientales y enviar dicha información a un servidor central con esta conexión a internet.

En una busca rápida, en uno de uno de los proveedores de conectividad enfocado al IoT [3], se pueden encontrar decenas de proyectos destinados a la medición de parámetros ambientales. Estos proyectos se diseñaron a medida del usuario y tienen características orientadas a los requerimientos del proyecto desarrollado. Pero si nos queremos centrar en algo más comercial, también podemos encontrar numerosos productos diseñados con esta finalidad. Por lo tanto, se podría decir que existen en la actualidad una gran cantidad de proyectos diseñados con este objetivo.

Haciendo un análisis superficial de los proyectos encontrados en “Sigfox”, nos encontramos que cada uno de los productos está diseñado con unas características muy diferentes de los demás. No solo los parámetros que se desean tomar hacen que cada proyecto tenga sus propias especificaciones, si no que el objetivo con el que estos se quieren tomar, el lugar, e incluso la forma en la que se recogen hacen que cada proyecto tenga sus propias características y difiera en gran medida de los demás.

En conclusión, existen una gran cantidad de proyectos destinados a la medición de parámetros ambientales y cada con sus propias características. Para realizar un análisis

más pormenorizado de estos proyectos, es posible separar estos en los cuatro siguientes bloques:

1. **Adquisición a nivel de usuario.** Particulares que adquieren estos productos para hacer mediciones ambientales en hogar, oficina, fincas, etc.
2. **Agricultura y ganadería**. Empresas que adquieren estos productos para monitorizar diferentes aspectos ambientales en los cultivos, ganaderías, invernaderos, etc.
3. I**nstalaciones y energía.** La instalación de estos dispositivos también se suele utilizar para el control de las condiciones de temperatura y ambientales de instalaciones eléctricas, energéticas o tecnológicas.
4. **Edificios inteligentes.** En la actualidad, y acentuado por la pandemia global, se está tratando de monitorizar las condiciones ambientales de edificios como hogares, aulas, tiendas, centros comerciales.

En estos cuatro sectores se requiere un aparato capaz de medir parámetros ambientales, habiendo por ello un gran mercado alrededor de este tipo de estaciones. Si hubiese que incluir este proyecto en alguno de estos grupos debería hacerse en el grupo de instalaciones y energía, ya que la mayoría de los sensores incluidos en la electrónica están destinados a la monitorización del funcionamiento de esta.

Centrando el foco en el objetivo de este TFG, que es el diseño del software que maneja estas estaciones. La mayoría de estos proyectos, al tratarse de sistemas cerrados, no se comparte el software que incluyen, e incluso, tampoco el hardware. Por ello es complicado hacer un análisis del componente software de estos proyectos.

Si es posible realizar un análisis de las características que portan la mayoría de estos proyectos en cuanto a sus funciones y capacidades. Siendo estas, las cuatro características más importantes que presentan la mayoría de ellos y que deberían tener la mayoría de los proyectos dedicados a la monitorización de un determinado entorno.

1. **Recopilación y gestión de datos**. Los últimos avances en el software de estaciones de recogida de parámetros medioambientales se centran en la recopilación y gestión eficiente de datos. Se han desarrollado sistemas capaces de adquirir datos en tiempo real a través de sensores distribuidos en una red de estaciones, y de almacenarlos de forma segura en bases de datos centralizadas. Estos sistemas permiten un acceso rápido y fácil a los datos, así como su integración con otras fuentes de información relevantes, como datos meteorológicos o de calidad del suelo.
2. **Visualización y análisis de datos**. El software de las estaciones medioambientales ha evolucionado en términos de capacidades de visualización y análisis de datos. Las interfaces gráficas intuitivas y personalizables permiten a los usuarios explorar los datos de manera interactiva, realizar análisis estadísticos, generar gráficos y mapas, y detectar patrones o tendencias significativas. Además, se han implementado herramientas avanzadas de análisis, como algoritmos de aprendizaje automático y minería de datos, que facilitan la identificación de relaciones complejas entre variables ambientales.
3. **Integración de sistemas y estándares**. El desarrollo de software para estaciones medioambientales ha enfocado esfuerzos en la interoperabilidad y la integración de

sistemas. Se han adoptado estándares abiertos y protocolos de comunicación, como el estándar de intercambio de datos “Sensor Observation Service” y el estándar de metadatos de observación “SensorML”, para facilitar la integración de estaciones de diferentes fabricantes y la compartición de datos entre distintas plataformas. Asimismo, se han desarrollado interfaces de programación de aplicaciones (API) para permitir la conexión con otros sistemas y facilitar el desarrollo de aplicaciones personalizadas.

4. **Monitoreo remoto y función de alerta**. Se ha desarrollado un software para estaciones de recolección ambiental para implementar sistemas de monitoreo y alerta remota. Estos sistemas permiten a los operadores de planta monitorear sus operaciones en tiempo real, identificar deficiencias o fallas y recibir alertas automáticas cuando ocurren ciertas condiciones climáticas. Esto ayuda a responder rápidamente a emergencias o eventos ambientales adversos.

De estas cuatro características relevantes, el objetivo de este TFG se centrará en la primera, la recopilación y la gestión de los datos. Esta lectura y gestión de los datos se realiza mediante algoritmos diseñados según las especificaciones del fabricante en la documentación del sensor. Estos algoritmos están muy ligados al método de comunicación que emplee el sensor y el sistema central que deseamos utilizar para realizar esta comunicación, que puede ser un microcontrolador o un microprocesador con las numerosas variantes que presentan cada uno de ellos.

Una vez seleccionados estos elementos llega el turno de desarrollar estos algoritmos que, mediante determinadas instrucciones, harán los pasos necesarios para obtener los datos tomados por el sensor. Al conjunto de estas funciones y rutinas se le suele denominar librería, estas librerías suelen ser desarrolladas por el fabricante del sensor o por la comunidad de desarrolladores.

Al utilizar una librería de sensores, los desarrolladores pueden evitar tener que implementar directamente la comunicación de bajo nivel con el hardware del sensor. En su lugar, pueden aprovechar las abstracciones proporcionadas por la librería para acceder rápidamente a los datos del sensor y utilizarlos en sus aplicaciones. Esto permite un desarrollo más eficiente y acelera el tiempo de implementación de proyectos que involucran sensores.

El objetivo, por tanto, de este proyecto es diseñar una librería que permita la lectura de los datos de los sensores antes mencionados dentro del sistema tecnológico que se proporciona que permita al usuario emplear estos datos. Bien, simplemente para su visualización, o bien, como en este proyecto para utilizarlos como apoyo a la obtención de los parámetros del agua a través de imágenes.

Una vez planteado el objetivo, es posible realizar un análisis de varias librerías que los fabricantes de los sensores proporcionan para que los clientes las apliquen en sus proyectos.

Los diferentes fabricantes de sensores suelen proporcionar librerías que faciliten la implementación de sus sensores en los proyectos de los usuarios. En el caso en el que nos encontramos podemos encontrar librerías de los sensores AS7341, WSEN HIDS y WSEN PADS.

La mayoría de estas librerías están implementadas en código C y en un formato de archivos compatible con la placa de desarrollo Arduino, como es el caso de estos tres ejemplos. Esto es debido a la gran versatilidad que tiene esta placa, siendo una de las mejores opciones para el primer testeo de este tipo de sensores. Gracias a estas librerías y a su fácil implementación en la placa puedes estar probando el sensor en cuestión de minutos. Siendo esto de gran utilidad a la hora de probar los sensores, sin necesidad de realizar todo el trabajo necesario para integrar el sensor en nuestro proyecto.

El esquema de estas librerías suele ser el mismo para todas. En ellas se suele encontrar un archivo principal con formato .cpp en el que se encuentran las funciones de bajo nivel que implementan las diferentes funcionalidades que tiene el sensor. Acompañando a este archivo se suelen incluir algunos ejemplos de código que realizaran algunas de las funciones básicas del sensor, ya sea la lectura de datos, configuración de parámetros, inicialización, etc. Estos ejemplos utilizan las funciones del archivo principal comentado anteriormente.

Normalmente, estas librerías son un buen punto de partida para entender el funcionamiento de nuestro sensor y ayudan al desarrollo de nuestro propio código. Al ser diseñadas por el propio fabricante implementan todas las funcionalidades que permite el sensor.

Para los sensores que estamos manejando podemos encontrar librerías para el AS7341 [4], WSEN_HIDS y WSEN_PADS [5].

En ambas librerías podemos encontrar un archivo principal con funciones que permiten configurar todas las características que incluye el sensor, como modificar algún parámetro de configuración o realizar la lectura de algún dato. Además, incluyen archivos de ejemplos que emplean estas mismas funciones para crear un script que realice alguna de las funcionalidades del sensor. Por ello, analizar estas librerías es de gran ayuda para comprender el funcionamiento de los sensores y así adaptarlos mejor al entorno de nuestro proyecto.

A diferencia de estas librerías, las diseñadas en este proyecto solo estarán enfocadas en cubrir las necesidades que se vayan a utilizar. Mientras que las funcionalidades que no vayan a utilizarse no serán desarrolladas.

Analizar estas librerías, también es de gran utilidad para ver cómo están documentadas. Una de las partes más importantes de una librería son las descripciones de las funciones, los parámetros, ejemplos de código que permitan a los desarrolladores o al usuario entender sin complicaciones cómo funciona la librería. En este aspecto ambas librerías cuentas con buenas descripciones tanto para las funciones como para los parámetros de esta, y como hemos dicho incluyen algunos ejemplos para que podamos ver como implementar estas funciones para lograr alguna funcionalidad concreta.

En mi opinión, la característica principal de estas librerías es que cubren todas las funcionalidades del sensor. Siendo esto de gran ayuda para no tener que analizar en profundidad la documentación del sensor para conocer alguna de estas funcionalidades o si tenemos alguna duda de como implementar alguna de ellas. En

ocasiones puede suceder que el “data sheet” no es todo lo preciso o completo que se quisiera y buscar una solución en estas librerías a veces es una buena opción, como se verá más adelante.

En resumen, se puede considerar que existe un gran mercado entorno a los dispositivos de monitorización de parámetros físicos, la mayoría pensados para la monitorización de hogares, oficinas y en menor medida para entornos exteriores. En lo que respecta al software de estos productos también se podría decir lo mismo, ya que la mayoría de los sensores que emplean estas tecnologías suelen tener diseñada alguna librería como las exploradas anteriormente. La mayoría están diseñadas o pensadas para usarse con una placa de Arduino, o con código PYTHON, lenguaje muy popular. En estos casos, la mayoría del trabajo estaría dedicado a la migración de estas funciones a nuestro proyecto y entorno de programación. Si hay alguna ocasión en la que el fabricante no aporta ninguna librería de ejemplo, como es el caso del sensor URM06. Para el que se necesitó desarrollar su código sin basarse en ninguna de estas librerías de ejemplo.

Por lo tanto, lo que se propone en este proyecto es el trabajo de desarrollo de una librería para un proyecto enfocado en la toma y envío de imágenes. Debido a esta condición y al empleo de un microcontrolador STM32, se hará un gran empleo del entorno de programación de ST y de la librería HAL. Esta es una libraría proporcionada por ST que aporta funciones y macros que abstraen las operaciones de los periféricos del microcontrolador, como GPIO, UART, SPI, I2C, temporizadores, etc. Estas funciones son de gran ayuda para no tener que diseñar los bloques de código de bajo nivel, pero esto se explicará en mayor detalle en la sección análisis del problema.

En conclusión, existe una amplia variedad de documentación y recursos disponibles para el desarrollo de funciones destinadas a sensores de este tipo. Tanto las librerías proporcionadas por los fabricantes como las funciones de la librería HAL simplifican significativamente la integración de estos sensores en nuestros proyectos. El trabajo restante, en la mayoría de los casos, se centra en adaptar estas librerías a las necesidades específicas de nuestro proyecto.

3. Análisis de problema

En el siguiente apartado se procede a realizar un análisis de los requisitos que se solicitan para la resolución del problema, y en base a ellos, se plantearan las soluciones correspondientes.

El requisito principal del proyecto es el diseño de unas librerías capaces de obtener **datos atmosféricos y operacionales** a través de diferentes sistemas electrónicos bajo el contexto de una estación de análisis de los parámetros del agua. Como he explicado en la introducción, la estación consta de numerosos sensores encargados de apoyar el análisis de las imágenes que toma, el objetivo de este TFG es **desarrollar una librería para cuatro de ellos**. Encargados de tomar los siguientes parámetros relevantes del entorno de la estación; **humedad**, **temperatura, distancia al agua, luminosidad en varios anchos de banda y presión atmosférica**.

Debido al requisito de la integración de estas librerías en un proyecto con cierto grado de desarrollo hay características del diseño de estas librerías que vienen impuestas de este proyecto, por lo tanto, es necesario adaptarse a ellas.

La primera de ellas es el tipo de microprocesador que se utiliza y el entorno de desarrollo que emplea este para el desarrollo de su software. El microprocesador corresponde a un Cortex-M4 de la gama de ST, en concreto el **STM32F429ZGT6** [6]. Entre sus características más destacadas están la integración de **DSP**, **FPU**, **1Mbyte de memoria flash** y capacidad para conectar mediante **FMC** una SDRAM externa, todo ello controlado a una velocidad de reloj de hasta 180 MHz. En cuanto al entorno de desarrollo, ST ha desarrollado el software **STMCubeIDE** que facilita el desarrollo de software para sus microprocesadores, esta aplicación permite diseñar y compilar código, así como un modo "debug" que se puede emplear para hacer simulaciones paso a paso.

Una de las características más destacadas de este software es el empleo de la **librería HAL**, que maneja las capas de bajo nivel del microprocesador, reduciendo el tiempo y coste para desarrollar el software del proyecto. Mas adelante se especificará con más detalle en que consiste esta librería y el por qué es de gran utilidad para el proyecto.

A continuación, detallo en una breve introducción, una pequeña descripción del funcionamiento de cada sensor, que método de comunicación emplea, cuál es la medida que se debe obtener de cada uno de ellos y cuáles de sus funcionalidades o características son necesarias implementar a través del código.

URM06 [7]. Sensor de distancia ultrasónico diseñado por DFROBOT. Los sensores ultrasónicos URM06 emiten pulsos ultrasónicos y, al medir el tiempo en que el pulso ultrasónico llega al objeto y regresa al transductor, se calcula la distancia del sensor al objeto. Son ampliamente utilizados para detectar desplazamiento, espesor, distancia, **nivel de agua**, nivel de material y objetos transparentes. Esta marca posee varios modelos de sensores con este propósito, en concreto este es capaz de medir distancias entre 20 cm y 10 m. Además, este modelo posee un ángulo de proyección de las ondas ultrasónicas de 15º. Esta característica es muy importante para nuestro proyecto ya que un ángulo de proyección demasiado grande puede hacer que las ondas choquen con objetos u obstáculos que pueda haber en el entorno de la estación, dificultando la medición de la distancia del sensor al agua. En cuanto al método de comunicación que el sensor utiliza para enviar el dato de distancia, este sensor envía la información contenida en un pulso cuadrado. Por lo tanto, para la lectura de esta medida se utilizara un pin de propósito general del micro (GPIO) capaz de detectar cuando esta señal pasa de alta a baja y viceversa. En el apartado diseño de la solución de detallará con mayor precisión en que consiste este proceso.

Ilustración 1. Sensor de distancia URM06.

La mayoría de las configuraciones que son necesarias implementar vienen de la necesidad de disminuir al máximo el consumo producido por los sensores. En este caso, este sensor no permite realizar ningún tipo de configuración en su funcionamiento. Pero si es posible controlar cuando el sensor está capturando o no medidas.

Pin Definition

- VCC: 6~12V@Max 2A (5V is acceptable but not recommended)
- GND: Ground
- Trig: Measurement trigger pin - Low level pulse over 50us triggers one distance measurement. Holding this pin LOW will repeats the measurement every 100ms.
- ECHO: Pulse signal output pin – 1us high level pulse equals 1mm in distance. If error occurs in detection, a 15000us high level pulse will be generated.

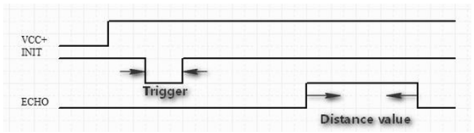

Ilustración 2. Descripción de las funcionalidades de los pines del sensor URM06

Por ello, únicamente tendremos como requisito habilitar un pin en el microcontrolador para actuar en el pin “trigger” y de esta forma ser capaces de activar y desactivar el sensor. Así, cuando hayamos terminado de realizar la medida pondremos en alta este pin para desactivar la captura de medidas del sensor y de esta manera reducir el consumo de energía del equipo.

WSEN PADS [8]. Sensor empleado para la medición de la presión absoluta diseñado por la marca WURTH ELECTRONICS. Tanto este sensor como el siguiente están diseñados por el mismo fabricante, por ello tienen bastantes similitudes entre ellos. Este se emplea para medir la presión atmosférica y la temperatura. Como método de comunicación emplea la línea I2C.

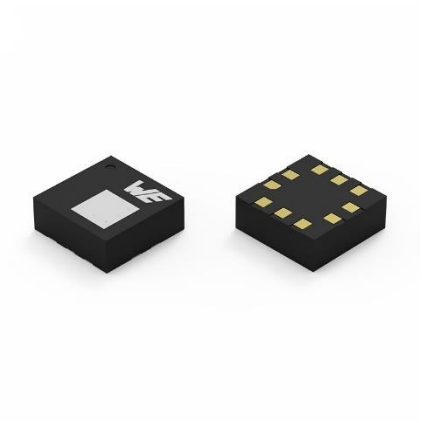

Ilustración 3. Sensor de presión absoluta WSEN-PADS.

Este sensor si tiene un mayor número de opciones de configuración que el anterior, las más importantes de cara a las necesidades del proyecto son las relacionadas con el ahorro de consumo. Para reducir el consumo, será necesario configurar el sensor de forma que realice medidas en el modo “single conversion” y activar su modo de bajo consumo.

Para realizar las medidas el sensor facilita un diagrama con los pasos necesarios a seguir para realizar lecturas en este modo de funcionamiento. Por lo tanto, seguiremos este esquema a la hora de realizar el algoritmo de lecturas de este sensor.

6.5 Sensor operation: single conversion mode

Flow chart shows sensor operation in the single conversion mode.

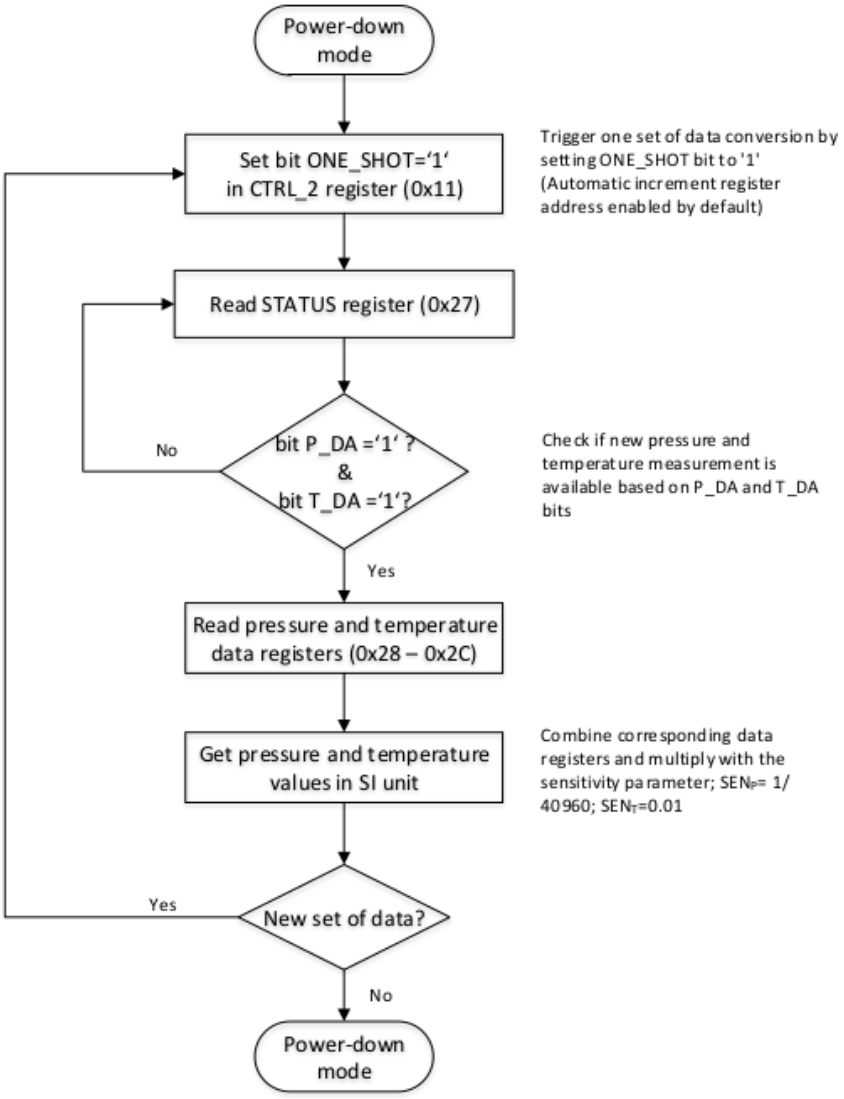

Ilustración 4. Diagrama de flujo para ejecución del modo "single conversion".

Complementando este modo de captura este sensor implementa la característica de poder aplicar un modo de bajo consumo o de bajo ruido. En el primero se reduce el consumo mientras que en el segundo se trata de reducir el ruido en la medición a costa de algo más de potencia consumida. Por ello será necesario configurar el sensor en el modo de bajo consumo.

7.4.1 Low-power or low-noise configuration

In the low-power configuration, the device is configured to minimize the current consumption. In the low-noise configuration, the device is configured to reduce the noise. During the continuous mode and single conversion mode, either one of these configurations can be selected as shown in Table 12. By default the sensor operates in the low-power configuration.

Address	Register	Bit	Bit value	Configuration
0x11	*CTRL_2*	LOW_NOISE_EN	0	Low-power (default)
0x11	*CTRL_2*	LOW_NOISE_EN	1	Low-noise

Ilustración 5. Características de los modos de bajo consumo y bajo ruido.

WSEN HIDS [9]. Sensor empleado para medir la humedad y temperatura, diseñado por la marca WURTH ELECTRONICS. Al igual que el anterior emplea una línea I2C para comunicarse con el micro.

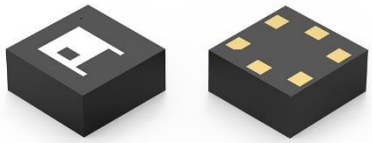

Ilustración 6. Sensor de humedad WSEN-HIDS.

De la misma forma que en el resto de los sensores se tratará de reducir al mínimo el consumo de energía aplicando la configuración correspondiente. Para este sensor, al igual que en el anterior, es posible configurar un modo “one-shoot” el cual realiza una medida y se mantiene a la espera de que solicitemos más sin consumir prácticamente energía.

9.2 One shot mode

One shot mode can be used when the humidity and/or temperature values are necessary in much less than 1Hz period instead of continuous measurement value. An application example could be a room monitoring system where humidity and temperature changes slowly with one measurement per 30 seconds is intended. This will also reduce the average power consumption of the sensor in comparison to continuous mode operation with 1Hz ODR.

Writing ONE SHOT bit to '1' in the *CTRL_2* register (0x21) will trigger a new measurement. After the new measurement is done, ONE SHOT bit in *CTRL_2* register (0x21) will automatically set to '0'. When new samples are available in the output registers, data ready bits H_DA and T_DA in *STATUS* register (0x27) will set to '1' notifying host controller that one shot measurement is completed and the values are ready to read.

Ilustración 7. Características del modo "one shot mode".

La forma de implementar este modo es similar al anterior y el data sheet presenta un esquema de cómo hacerlo.

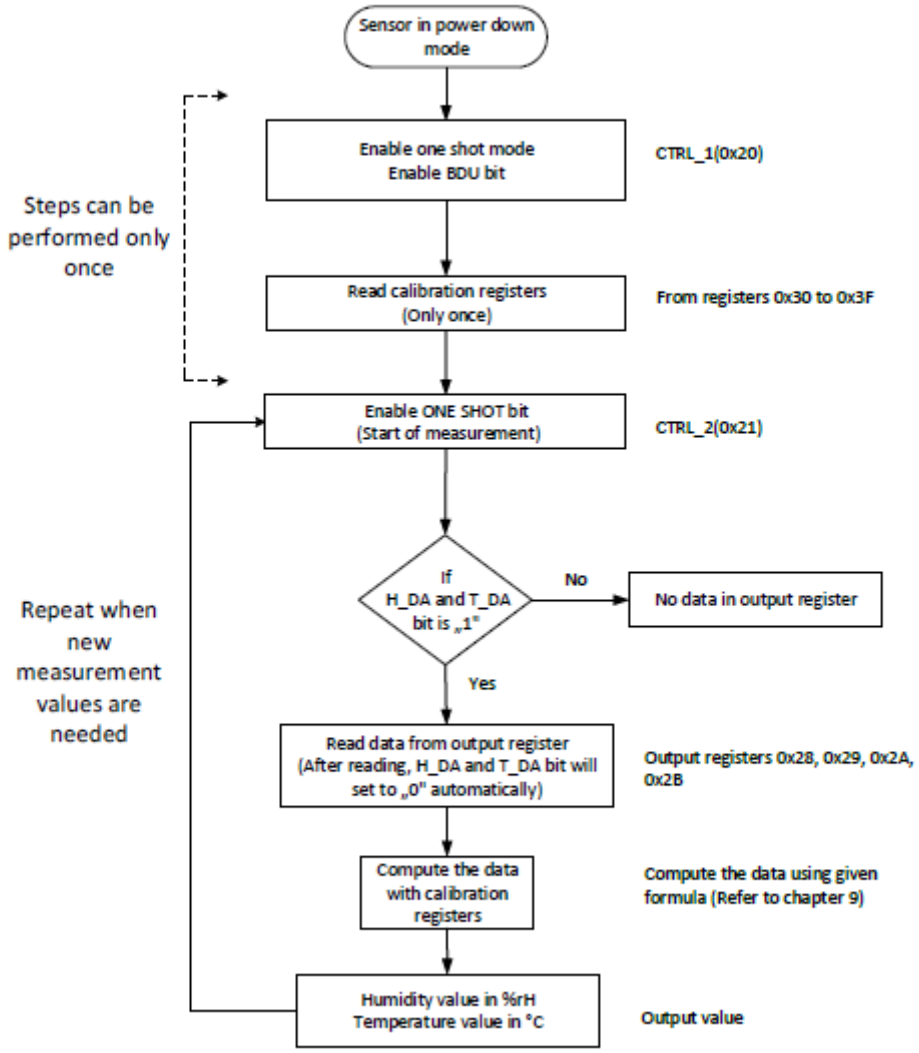

Ilustración 8. Diagrama de flujo para la ejecución del modo "one shot mode".

AS7341 [10]. Este sensor es utilizado para obtener, en ocho bandas diferentes, la intensidad lumínica y como los dos sensores anteriores hace uso de la línea I2C para comunicarse con el microcontrolador. Esta diseñado por la marca AMS.

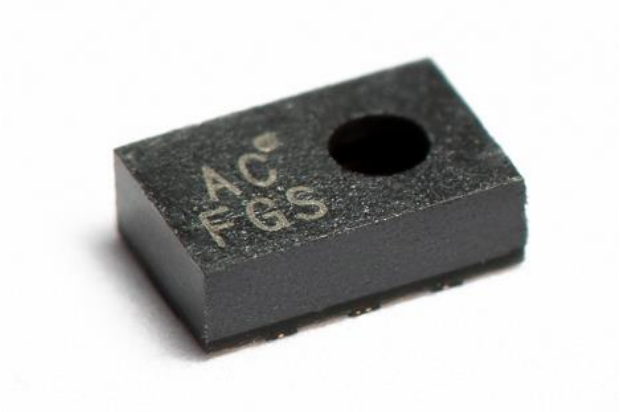

Ilustración 9. Sensor multiespectral AS7341.

El AS7341 tiene una característica muy útil para nuestro proyecto y es que permite configurar la ganancia en modo automático. Esto es muy conveniente, ya que debido a las variaciones lumínicas que hay en los entornos en los que están colocadas las estaciones. Es muy fácil que la resolución por la que se mueva el ADC no sea la adecuada, o bien porque el dato sature o bien por que la variación sea tan mínima que no se puedan apreciar cambios. Gracias a este modo la ganancia se autorregulará en función de la luminosidad que haya en el medio en ese instante. Por ello es imprescindible implementar esta opción para que el sensor configure la ganancia más adecuada.

Benefits	Features
Precision color, spectral composition and distribution measurments	8 optical channels distributed over the visible spectral range + clear and NIR channels realized via silicon nano-optic interference filter deposition technologies
Low power consumption and efficient I²C communication	• 1.8VDD operation, max 300µA • Configurable sleep mode to <5µA • Interrupt-driven device
Integrated ambient light flicker detection on chip and light source detection through NIR channel	• Dedicated channel • Independently configurable timing and gain • Automatic gain adjustment • 50Hz and 60Hz flicker detection flags
Electronic shutter/trigger and synch control	GPIO can be used as external trigger input or light source synchronization output
External photodiodes to expand detection range	GPIO can be used as input for external photodiodes including mid-IR range

Ilustración 10. Características del sensor AS7341.

Además, en cuanto al diseño de las funciones, es importante tener en cuenta que estos datos se enviarán a un servidor en el cual se mostrarán junto con la imagen de la que van acompañados. Por lo tanto, es importante enviar estos datos cuando estemos seguros de que son válidos, por ello se incluirán variables que permitan reconocer si los sensores están funcionando correctamente y así estar seguros de que estos se pueden enviar al servidor.

Por último, se realizará un análisis de la librería HAL para así entender por qué es una pieza tan importante en el diseño de la solución, pero antes de ello explicaré en que consiste el entorno de desarrollo STMCuceIDE, ya que esta librería está incorporada en este programa que facilita enormemente el desarrollo de código en microcontroladores de la marca ST.

STM32CubeIDE es un entorno integrado de desarrollo (IDE) desarrollado por STMicroelectronics para programar y desarrollar aplicaciones para microcontroladores STM32.

STM32CubeIDE proporciona herramientas de desarrollo completas, que incluyen un editor de código, un compilador, un depurador y otras utilidades necesarias para el desarrollo de software para microcontroladores STM32.

El entorno STM32CubeIDE está diseñado para ser intuitivo y ofrece características como la configuración gráfica de periféricos, la generación automática de código inicial y herramientas de depuración avanzadas. Facilita el desarrollo de software para aplicaciones embebidas, permitiendo a los desarrolladores centrarse en la lógica de la aplicación en lugar de preocuparse por la configuración de hardware de bajo nivel. Además, integra la biblioteca STM32Cube, que es un conjunto de software periférico y middleware que facilita el desarrollo de aplicaciones para los microcontroladores STM32. Es esta biblioteca la que incorpora la librería HAL.

La librería HAL (Hardware Abstraction Layer) de STMicroelectronics es una capa de abstracción de hardware diseñada para proporcionar una interfaz consistente y portátil para acceder a las funciones del hardware en los microcontroladores STM32. Esta capa de abstracción facilita el desarrollo de software independiente del hardware subyacente, permitiendo a los desarrolladores escribir código que sea más portable entre diferentes modelos de microcontroladores STM32.

Gracias a esta biblioteca y al entorno de desarrollo STMCubeIDE es posible realizar una configuración de forma gráfica del microcontrolador STM32 que estemos utilizando. El programa, ayudándose de la librería HAL, inicializara todos los registros necesarios para configurar todos los periféricos del micro tal y como el usuario desee.

Además de esto, la biblioteca incluye funciones que permiten realizar alguna acción determinada con algún periférico. Por ejemplo, hay funciones que permiten escribir y leer registros de un sensor I2C simplemente aportando en los argumentos de la función la dirección I2C y el registro que se quiere leer o escribir del mismo. Esta función en concreto será muy utilizada en el diseño de la solución ya que la mayoría de los sensores utilizan la línea I2C como método de comunicación. En ese apartado se explicará más detalladamente como está diseñada esta función. Todas estas funciones permiten no tener que preocuparse por las operaciones de bajo nivel del microcontrolador, lo que permiten primero ahorrar un gran tiempo de desarrollo y segundo te dan la garantía de que van a funcionar correctamente.

Entrando más en detalle en el funcionamiento de cada sensor, podemos separarlos en dos grupos diferentes. Los conectados mediante la interfaz I2C (AS7341, PADS y HIDS) y el URM06.

Los conectados mediante I2C requerirán en gran medida del empleo de la librería HAL para realizar la comunicación entre el microcontrolador y los sensores. I2C [11] es un sistema de comunicación que se usa en multitud de dispositivos como memorias, sensores y otros circuitos integrados. Posee dos líneas, el SCL y el SDA que deben estar conectadas en todos los dispositivos con los que se desea comunicar.

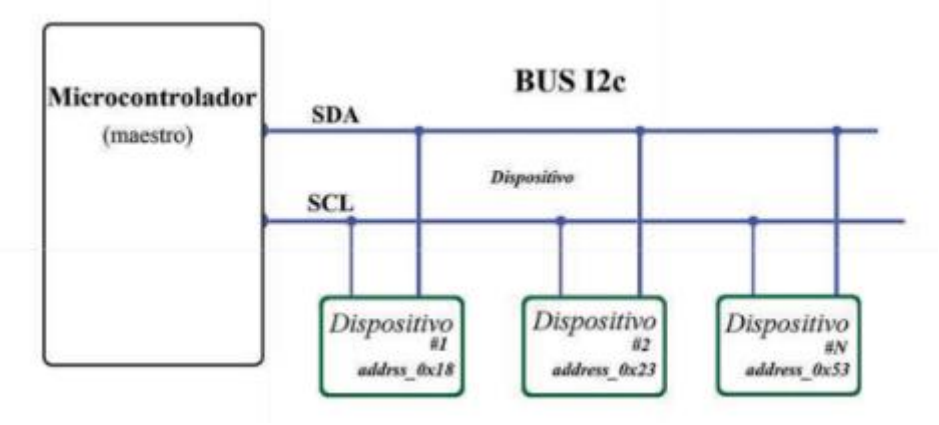

Ilustración 11. Implementación física de una línea I2C.

Los paquetes de información deben enviarse según un protocolo. En el cual debe enviarse primero una condición de inicio (Start), el envío de los datos que se quieren enviar y una condición de finalización (Stop).

Cada condición tiene unos requisitos específicos que debe cumplir para que el dispositivo esclavo las detecte correctamente. La información se envía dentro de tramas que contienen la dirección I2C del dispositivo al que se quiere enviar la información y paquetes que configuran o solicitan al dispositivo cierta información necesaria. Cuando un dispositivo recibe una trama que corresponde con su dirección I2C, este envía un bit de reconocimiento llamado ACK para confirma al dispositivo maestro que ha recibido la trama correctamente.

Como se puede ver, sin profundizar demasiado en que consiste el protocolo I2C, el dispositivo maestro debe de encargarse de realizar varios procesos para ejecutar la comunicación en un bus I2C. Es posible realizar una programación de los pines SDA y SCL de forma manual para realizar estos procesos y que estos se comporten como el protocolo dicta, pero gracias a la librería HAL y las funciones orientadas a la comunicación en este protocolo no será necesario diseñar estos algoritmos. Esto ahorra una gran carga de trabajo y además las funciones incluyen multitud de opciones y banderas que nos permiten depurar la comunicación de forma óptima.

Debido a esto, el diseño de la parte de comunicación será sencillo gracias a esta librería, las dificultades se pueden encontrar a la hora de entender cómo funcionan. Ya que estos sensores son algo más complejos y permiten una mayor configuración que por ejemplo el sensor de distancia URM06. Por lo tanto, el foco del trabajo de desarrollo de estos sensores estará en sus “data sheet”. A los que habrá que prestarles especial atención y leerlos detenidamente.

Por el contrario, la dificultad a la hora de diseñar la parte correspondiente al sensor URM06, estará en el diseño de un algoritmo que permita la lectura del tren de pulsos

que contiene la información de la distancia. Para realizar esta tarea con un microcontrolador se deberá hacer uso de interrupciones y temporizadores, los cuales también se configurarán con la librería HAL.

Resumen de los requisitos:

- Implementación de una librería para la adquisición de datos con el sensor de distancia URM06 en el entorno de desarrollo STMCubeIDE empleando como método de comunicación un pin de propósito general GPIO.
- Implementación de una librería para la adquisición de datos con el sensor de temperatura y presión PADS en el entorno de desarrollo STMCubeIDE empleando como método de comunicación una línea I2C.
- Implementación de una librería para la adquisición de datos con el sensor de temperatura y humedad HIDS en el entorno de desarrollo STMCubeIDE empleando como método de comunicación una línea I2C.
- Implementación de una librería para la adquisición de datos con el sensor lumínico multiespectral AS7341 en el entorno de desarrollo STMCubeIDE empleando como método de comunicación una línea I2C.
- Establecer un método de control que permita reconocer cuando los sensores están o no disponibles.
- Configurar cada uno de los sensores de la forma que exige el proyecto de monitorización de la calidad del agua.

Estas serían las principales cuestiones que plantea el proyecto, en el apartado diseño de la solución se detallará como se ha resuelto cada una de ellas.

Por último, en esta sección, se añadirán algunas imágenes de la electrónica empleada para dar contexto del material empleado.

La estación consta de varias placas interconectadas entre sí por diferentes métodos en las cuales los componentes y sensores necesarios están repartidos entre ellas. Los sensores empleados en el proyecto se pueden ver en las siguientes imágenes.

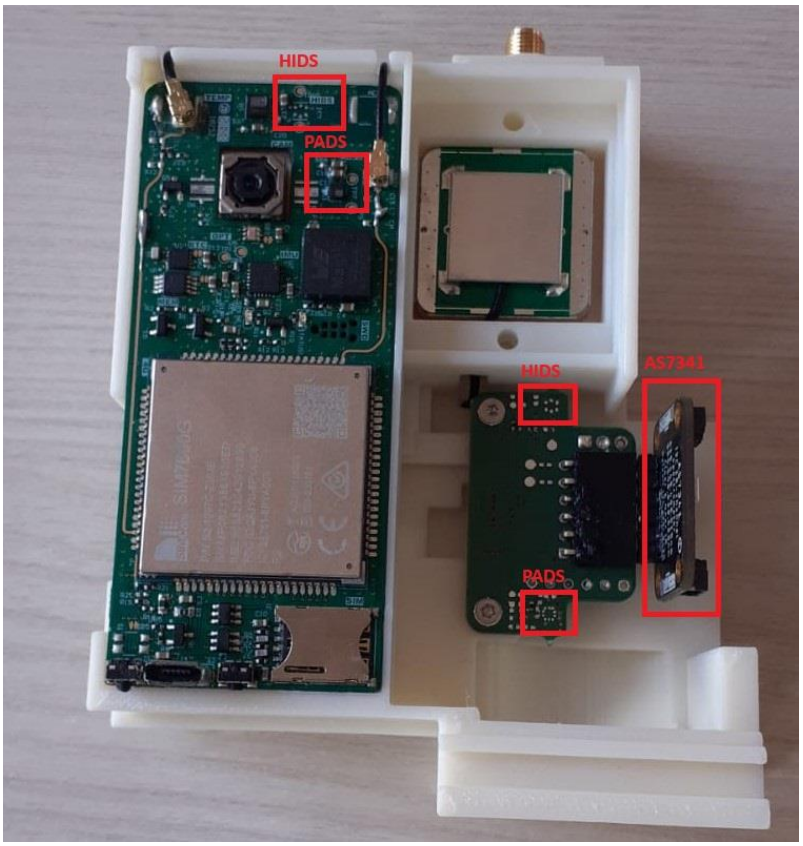

Ilustración 12. Estación de monitorización del agua.

Se distribuyen dos sensores HIDS y PADS para evitar que los sensores estén demasiado influidos por la temperatura que alcanzan las placas principales de la estación. De esta forma dos sensores están dedicados a medir la temperatura interna de la estación y otros dos a captar la temperatura externa. A la hora de escribir el código simplemente habrá que tener en cuenta la dirección I2C de cada uno para realizar la lectura correspondiente.

El sensor multiespectral está incluido en una placa de desarrollo diseñada por ADAFRUIT. Este sensor queda orientado hacia el cielo cuando la estación está colocada. Su objetivo es obtener la luminosidad del mismo.

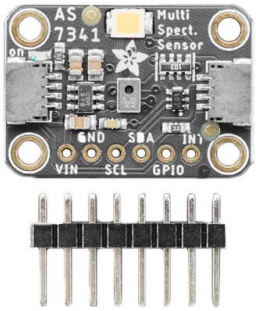

Ilustración 13. Placa de desarrollo del sensor AS7341.

La localización del sensor URM06 está más aislada aun de la estación debido a que para que el sensor funcione correctamente debe situarse a cierta distancia de esta para evitar obstáculos y colocarse perpendicular al agua. Este está situado al extremo de una barra para alcanzar esta posición y conectado a la estación mediante un cable de 4 hilos que llega a la misma a través de la barra.

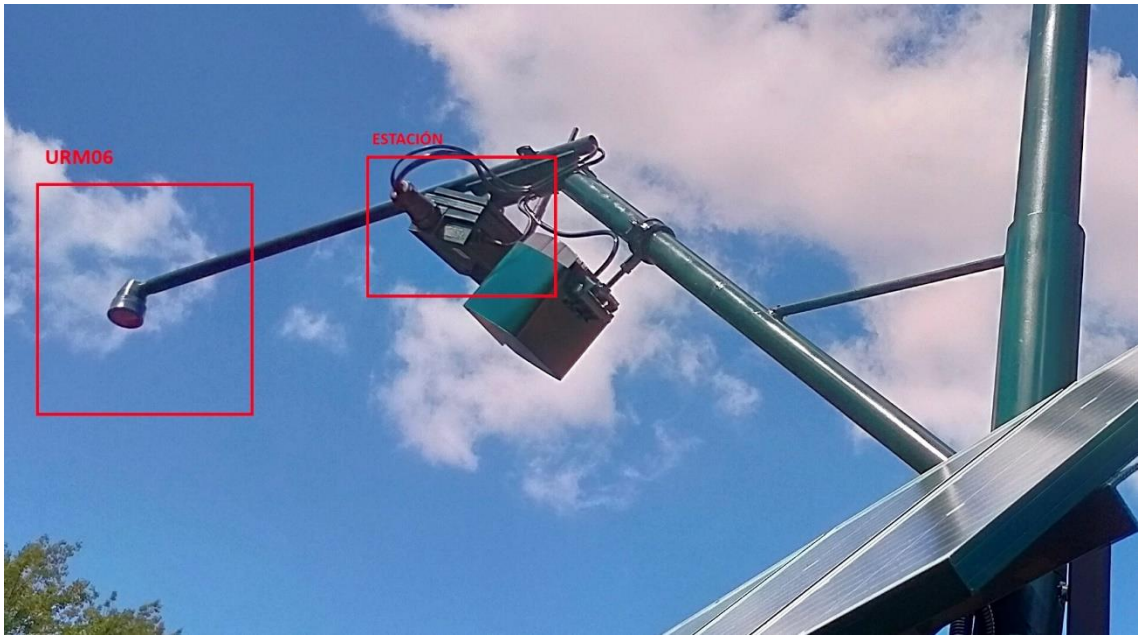

Ilustración 14. Imagen de una de las estaciones de monitorización.

4. Diseño de la solución.

4.1. Configuración de los pines del microcontrolador.

En esta sección se explica cómo se han configurado los pines del sensor para que funcionen como terminales I2C, haciendo uso de la interfaz gráfica de STMCubeIDE y que funciones de la librería HAL se utilizaran para realizar las lecturas y escrituras en sensores I2C.

Para realizar esta configuración el entorno de desarrollo de ST dispone de una interfaz gráfica que permite seleccionar los pines correspondientes a nuestra línea I2C y configurarlos para que realicen esta función.

Esta interfaz también permite configurar los parámetros más importantes de la interfaz I2C como la frecuencia de la línea y el tamaño de las direcciones I2C entre otras.

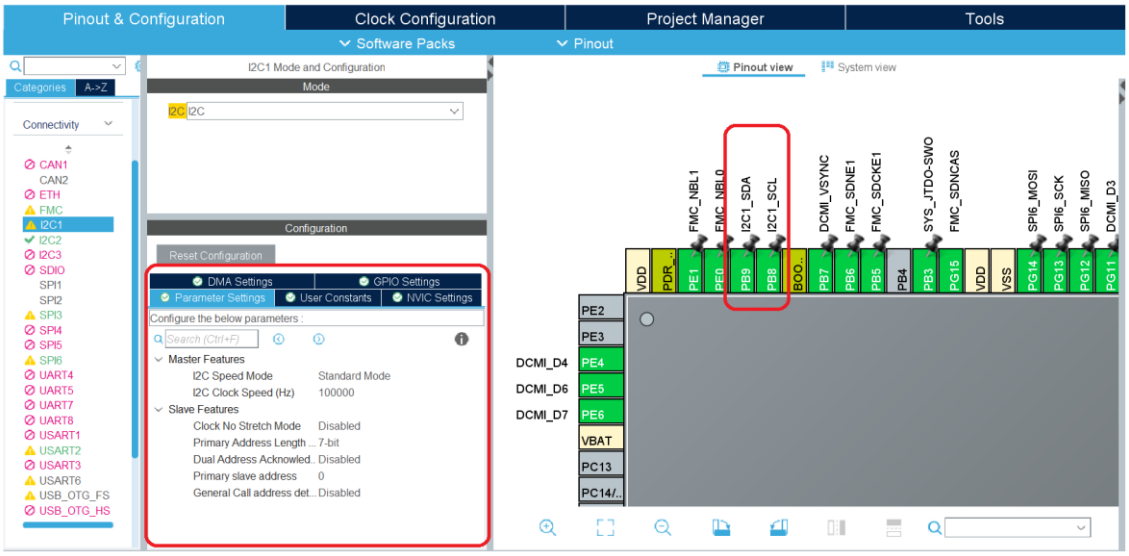

Ilustración 15. Interfaz de STMCubeIDE para la configuración de los pines.

Una vez seleccionados los parámetros correspondientes STMCubeIDE genera el código para que el hardware funcione tal y como se ha seleccionado en la interfaz. Para ello, genera dos funciones dedicadas a inicializar los relojes de los pines correspondientes y configurar los registros del micro relacionados con la interfaz I2C.

```
88   /* Initialize all configured peripherals */
89   MX_GPIO_Init();
90   MX_I2C1_Init();
```

Ilustración 16. Líneas de código para la inicialización de la línea I2C.

Estas funciones configuran los registros del micro para que los pines y la interfaz I2C se comporten como hemos seleccionado en la interfaz.

Además de estas funciones de configuración e inicialización de la línea I2C, la librería HAL dispone de multitud de funciones para el manejo y la utilización de esta interfaz. En este proyecto emplearemos principalmente dos de ellas. Las utilizadas para leer y escribir registros de los sensores disponibles en la línea I2C.

La función de escritura es la siguiente y consta de los siguientes parametros.

- hi2c. Es la estructura utilizada para almacenar todas las variables y datos necesarios para el correcto funcionamiento de la línea I2C. Se le suele llamar el "Handle".
- DevAddress. Parámetro utilizado para definir a que sensor I2C de los disponibles en la línea queremos acceder. Cada uno de ellos tiene su propia dirección I2C, la cual se emplea saber con que sensor queremos intercambiar información.

- MemAddress. Dentro de los sensores I2C existen un número variable de registros que permiten configura el sensor o realzar la lectura de los datos. En este argumento se selecciona a cuál de estos registros queremos acceder. Para saber el nombre de estas direcciones es necesario acceder a la documentación proprocionada por el fabricante del sensor.
- MemAddSize. Este argumento indica el tamaño de los registros de sensor, en la mayoría de casos será 8 bits, pero es posible que sean de 16. Para que la función se ejecute correctamente es necesario indicar este valor.
- pData. Puntero que señala a la estructura o variable en la cual se almacenan los datos que queremos escribir en el registro o registros anteriormente seleccionados.
- Size. Es posible escribir en mas de un registro empleando una sola vez la función de escritura, en este argumento se indica el número de registros en los que se escribirá.
- Timeout. Tiempo máximo para realizar la función.

```
2391 /**
2392   * @brief  Write an amount of data in blocking mode to a specific memory address
2393   * @param  hi2c Pointer to a I2C_HandleTypeDef structure that contains
2394   *               the configuration information for the specified I2C.
2395   * @param  DevAddress Target device address
2396   * @param  MemAddress Internal memory address
2397   * @param  MemAddSize Size of internal memory address
2398   * @param  pData Pointer to data buffer
2399   * @param  Size Amount of data to be sent
2400   * @param  Timeout Timeout duration
2401   * @retval HAL status
2402   */
2403 HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c,
2404                                     uint16_t DevAddress, uint16_t MemAddress,
2405                                     uint16_t MemAddSize, uint8_t *pData,
2406                                     uint16_t Size, uint32_t Timeout)
```

Ilustración 17. Descripción y parámetros de la función de escritura en un dispositivo I2C.

La función de lectura es la siguiente y consta de los siguientes parámetros.

- hi2c. Es la estructura utilizada para almacenar todas las variables y datos necesarios para el correcto funcionamiento de la línea I2C. Se le suele llamar el "Handle".
- DevAddress. Parámetro utilizado para definir a que sensor I2C de los disponibles en la línea queremos acceder. Cada uno tiene su dirección I2C, que usa par a saber con qué sensor queremos intercambiar información.
- MemAddress. Dentro de los sensores I2C existen un número variable de registros que permiten configurar el sensor o realizar la lectura de los datos. En este argumento se selecciona a cuál de estos registros queremos acceder. Para saber el nombre de estas direcciones es necesario acceder a la documentación proprocionada por el fabricante del sensor.
- MemAddSize. Este argumento indica el tamaño de los registros del sensor, que en la mayoría de los casos será de 8 bits, pero es posible que sean de 16. Para que la función se ejecute correctamente es necesario indicar este valor.
- pData. Puntero que señala la estructura o variable en la que se almacenan los datos que queremos leer del registro o registros seleccionados.
- Size. Es posible leer mas de un registro empleando una sola vez la función de lectura, en este argumento se indica el número de registros que se leerán.

- Timeout. Tiempo máximo para realizar la función.

```
2524 /**
2525   * @brief  Read an amount of data in blocking mode from a specific memory address
2526   * @param  hi2c Pointer to a I2C_HandleTypeDef structure that contains
2527   *               the configuration information for the specified I2C.
2528   * @param  DevAddress Target device address
2529   * @param  MemAddress Internal memory address
2530   * @param  MemAddSize Size of internal memory address
2531   * @param  pData Pointer to data buffer
2532   * @param  Size Amount of data to be sent
2533   * @param  Timeout Timeout duration
2534   * @retval HAL status
2535   */
2536 HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c,
2537                                    uint16_t DevAddress,
2538                                    uint16_t MemAddress,
2539                                    uint16_t MemAddSize,
2540                                    uint8_t *pData,
2541                                    uint16_t Size,
2542                                    uint32_t Timeout)
```

Ilustración 18. Descripción y parámetros de la función de lectura de un sensor I2C.

Gracias a estas dos funciones diseñaremos las librerías de los sensores que emplean la interfaz I2C sin tener que diseñar las funciones de bajo nivel encargadas de transmitir y recibir datos a través de la línea I2C, por lo que serán muy utilizadas en el código de los sensores WSEN-PADS, WSEN-HIDS y AS7341.

4.2. Diseño de la librería del sensor WSEN-PADS respondiendo a los criterios del diseño.

Después de haber configurado los pines del microcontrolador y aprendido como se utilizan las funciones de la librería HAL relacionadas con la interfaz I2C, es posible utilizar estas funciones para ejecutar los diferentes algoritmos que he diseñado para la utilización de los sensores I2C. Para ilustrar mejor la secuencia que he seguido para realizar la inicialización, lectura y cálculo de los datos que aporta cada sensor. He diseñado diagramas de flujo en los que se puede ver con claridad cual es el orden que he seguido para realizar estos procesos.

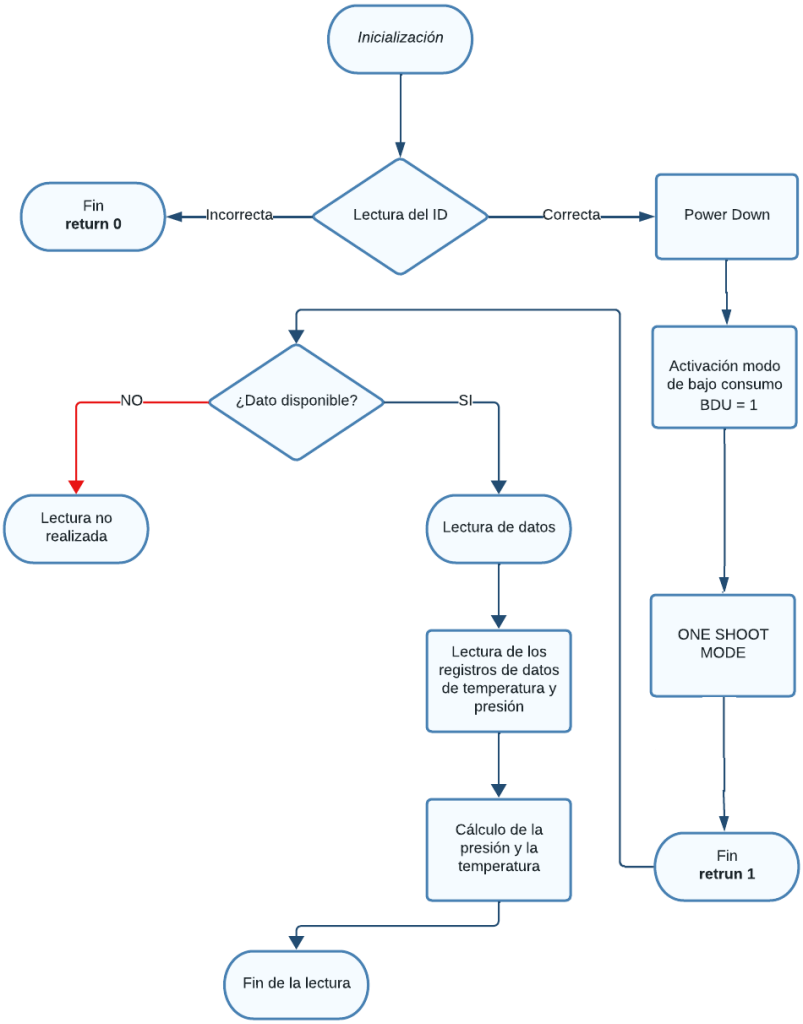

Ilustración 19. Diagrama de flujo del funcionamiento del sensor WSEN-PADS.

El diagrama comienza con una inicialización que se encarga de configurar el sensor de la forma más óptima para el proyecto. En primer lugar, se realiza una comprobación de que el sensor está disponible haciendo una lectura de su ID. A continuación, se configura el sensor en modo “power down/single conversion” ya que es el modo que menos consumo requiere. Para ello configuramos los bits del ODR del registro CTRL_1 con “000”.

12.6 CTRL_1 (0x10)

Address: 0x10
Type: R/W
Default Value: 00000000b

7	6	5	4	3	2	1	0
0	ODR[2:0]			EN_ LPFP	LPFP_ CFG	BDU	0

ODR[2:0] Selection of operating mode and ODR as per table 20

ODR[2:0]	Output Data Rate [Hz]
000	Power-down mode / single-conversion mode
001	1
010	10
011	25
100	50
101	75
110	100
111	200

Table 20: Output data rate selection

EN_LPFP Enable/disable Low-pass filter. For more information refer to section 7.4.2
0: low-pass filter is disabled; 1: low-pass filter is enabled

EN_LPFP Configure low-pass filter. For more information refer to section 7.4.2

BDU Block data update feature
0: data register updates continuously; 1: data register not updated until MSB and LSB has been read

Ilustración 20. Configuración del registro CTRL_1 del sensor WSEN-PADS.

Antes de realizar las modificaciones en el registro CTRL_2 se configura el bit BDU a “1” para que el sensor no actualice los registros de medición hasta que no se hayan leído todos. Esto garantiza que todos los registros contienen el dato de una única medición. Ya que puede suceder, que si se realizan lecturas de forma más lenta que la frecuencia asignada en ODR el dato almacenado en los registros sea de muestras de diferentes tiempos. Al configurar el sensor en modo “single conversión” esto no debería suceder, pero el data sheet indica que es altamente recomendable hacerlo.

It is strongly recommended to enable BDU feature. This avoids an update of the DATA_x_x registers until all the parts of the corresponding DATA_x registers have been read.

Ilustración 21. Recomendación para el sensor WSEN-PADS.

Por último, una vez configurado el sensor en modo “single-conversion” y activado el BDU, configuramos el modo de bajo consumo e indicamos al sensor que realice una medición mediante el bit “ONE SHOOT” del registro CTRL_2.

12.7 CTRL_2 (0x11)

Address: 0x11
Type: R/W
Default Value: 00010000b (0x10)

7	6	5	4	3	2	1	0
BOOT	INT_H_L	PP_OD	IF_ADD_INC	0	SW RESET	LOW_NOISE_EN	ONE_SHOT

BOOT — Reboots memory content. For details refer to section 6.3
0: normal operation; 1: reboot memory content

INT_H_L — Select interrupt: active high or active low
0: active high; 1: active low

PP_OD — Push-pull or open-drain selection on the *INT* pin
0: push-pull; 1: open-drain

IF_ADD_INC — Register address is automatically incremented during multiple byte access
0: disabled; 1: enabled
This is a multi-read/write feature that enables a repeated read/write operation during a single bus transaction by automatically incrementing the register address. This feature is enabled by default.

SWRESET — Perform a software reset. For more information, refer to section 6.4
0: Normal operation; 1: software reset

LOW_NOISE_ENABLE — Enable low-noise or low-power configuration. For more information refer to section 7.4.1
0: low-power mode; 1: low-noise mode

ONE_SHOT — Enables single data acquisition of pressure and temperature. For more information refer to section 7.2
0: normal operation 1: a new data set is acquired

Ilustración 22. Configuración del registro CTRL_2 del sensor WSEN-PADS.

Con esto finalizaría el proceso de inicialización, para pasar a realizar la lectura y el cálculo de los datos. La lectura y el cálculo se realiza de la siguiente forma.

```
109     //Lectura de temperatura y presion
110     if(HAL_I2C_Mem_Read(lhi2c1, I2C_Adress, 0x28, I2C_MEMADD_SIZE_8BIT, Data_T_P, 5, 10000) != HAL_OK)
111     {
112             Error_Handler();
113     }
114
115     //Cálculo de temperatura y presión
116     Presion= (Data_T_P[2]<<16) + (Data_T_P[1]<<8) + Data_T_P[0];
117     DatosStruct->presion = Presion/40960;
118     Temp = (Data_T_P[4]<<8) + Data_T_P[3];
119     if(Temp > 32768)
120     {
121         T_neg = Temp - 32768;
122         DatosStruct->temp_pads = (double)(0 - 32768 + T_neg)/100;
123     }
124     else
125     {
126         DatosStruct->temp_pads = (double)Temp/100;
127     }
```

Ilustración 23. Líneas de código para lectura y cálculo del dato de temperatura y presión.

Para realizar el cálculo de la presión basta con concatenar los tres registros que contienen el dato bruto de la presión y dividir entre 40960. Mientras que para calcular la temperatura hay que realizar la concatenación considerando que el dato de 16 bits está en nomenclatura de complemento a dos. Esto es necesario tenerlo en cuenta con la temperatura ya que puede ser negativa. Por ello, el cálculo de la temperatura incluye un "if" para diferenciar los casos en los que el resultado es positivo y negativo. Los datos se obtienen en ºC y KPa.

Para el diseño y mejor organización del código de este sensor se han diseñado tres funciones que agrupan en tres bloques diferentes todo el proceso que se ha visto en el diagrama anterior:

- DATA_PADS_DISP. Esta función está diseñada para realizar una espera dinámica que permita al sensor actualizar los datos de presión y temperatura. Si el dato ya está disponible, no se realiza espera. Si aún no lo está, se esperan 200 ms y se realiza una nueva comprobación. Hasta un máximo de cinco intentos, es decir, un segundo máximo de espera.
- INIT_WSEN_PADS. Para realizar la lectura y que el sensor se mantenga configurado tal y como deseamos, se diseñó una función encargada de la inicialización de este, que configura algunos parámetros del sensor. Esta función incluye un argumento que permite cambiar la dirección I2C del sensor al que se quiere acceder. Esto es porque se disponen de varios WSEN_PADS en la electrónica para detectar la temperatura en distintos puntos de ella. Esto es posible gracias a que este sensor permite cambiar la dirección I2C del mismo mediante un pin (SAO) diseñado para esta utilidad. Por lo tanto, se empleará una u otra dirección en función del sensor al que se desea acceder.

4.3.6 Slave address for the sensor

The slave address is transmitted after the start condition. Each device on the I^2C bus has a unique address. Master selects the slave by sending corresponding address after the start condition. A slave address is 7 bits long followed by a Read/Write bit.

Figure 7: Slave address format

The 7-bit slave address for this sensor is 101110xb. LSB of the 7-bit slave address can be modified with the *SAO* pin. When SAO is connected to positive supply voltage, the LSB is '1', making 7-bit slave address 1011101b (0x5D). If SAO is connected to ground, the LSB is '0', making 7-bit address 1011100b (0x5C).

The R/W bit determines the data direction. A '0' indicates a write operation (transmission from master to slave) and a '1' indicates a read operation (data request from slave).

Slave address[6:1]	Slave address[0]	7-bit slave address	R/W	Slave address + R/W
101110	SAO=0	1011100b (0x5C)	0	10111000b (0xB8)
101110			1	10111001b (0xB9)
101110	SAO=1	1011101b (0x5D)	0	10111010b (0xBA)
101110			1	10111011b (0xBB)

Table 8: Slave address and Read/Write commands

Ilustración 24. Configuración del pin de selección de dirección I2C del sensor WSEN-PADS.

- PRESTEMP_WSEN_PADS. Función que realiza la lectura y el cálculo de la presión y la temperatura.
- OFF_WSENPADS. Función que pone en modo "Stand-by" el sensor.

Las funciones son llamadas en el siguiente orden en la función "main".

```
335         if(INIT_WSENPADS(&hi2c1, WSEN_PADS_INT_I2C_ADDRESS) == 1)                    //pads interior
336         {
337             Sens_Disp.pads = 1;
338             DATA_PADS_DISP(&hi2c1, WSEN_PADS_INT_I2C_ADDRESS);
339             PRESTEMP_WSENPADS(&hi2c1, &Param_Sens, WSEN_PADS_INT_I2C_ADDRESS);
340             myFtoa(Param_Sens.temp_pads, Temp_int_s);
341             myFtoa(Param_Sens.presion, Pres_s);
342         }
343         else
344         {
345             Sens_Disp.pads = 0;
346         }
```

Ilustración 25. Líneas de código para la ejecución de las funciones diseñadas para el sensor WSEN-PADS.

En primer lugar, se llama a la función de inicialización, que además de realizar la configuración del sensor nos permite saber si el sensor está operativo, si es el caso se realiza una espera con la función que detecta si hay un dato disponible y a continuación se realiza la lectura y el cálculo de los datos. Por último, se almacenan los datos como un “array” en las variables Pres_s y Temp_int_s, para ello se utiliza la función myFtoa que convierte un “float” en un array. Esto es necesario para hacer el envío a través del protocolo HTTP.

También, en esta sección, dependiendo de si el sensor se inicializa correctamente o no se pone a “1” la bandera Sens_Disp.pads que permite decidir si se quiere enviar o no el dato de presión más adelante. Ya que si la inicialización no es correcta es importante no realizar el envío de este dato.

En el modo “single conversion” no es necesario apagar el sensor al terminar la lectura.

4.3. Diseño de la librería del sensor WSEN-HIDS respondiendo a los criterios del diseño.

De igual forma, empleando las funciones de la librería I2C de lectura y escritura de registros se emplean diferentes procesos para realizar la inicialización y lectura de datos del sensor WSEN-HIDS. En el siguiente esquema se indican los procesos y el orden de estos que se ha seguido.

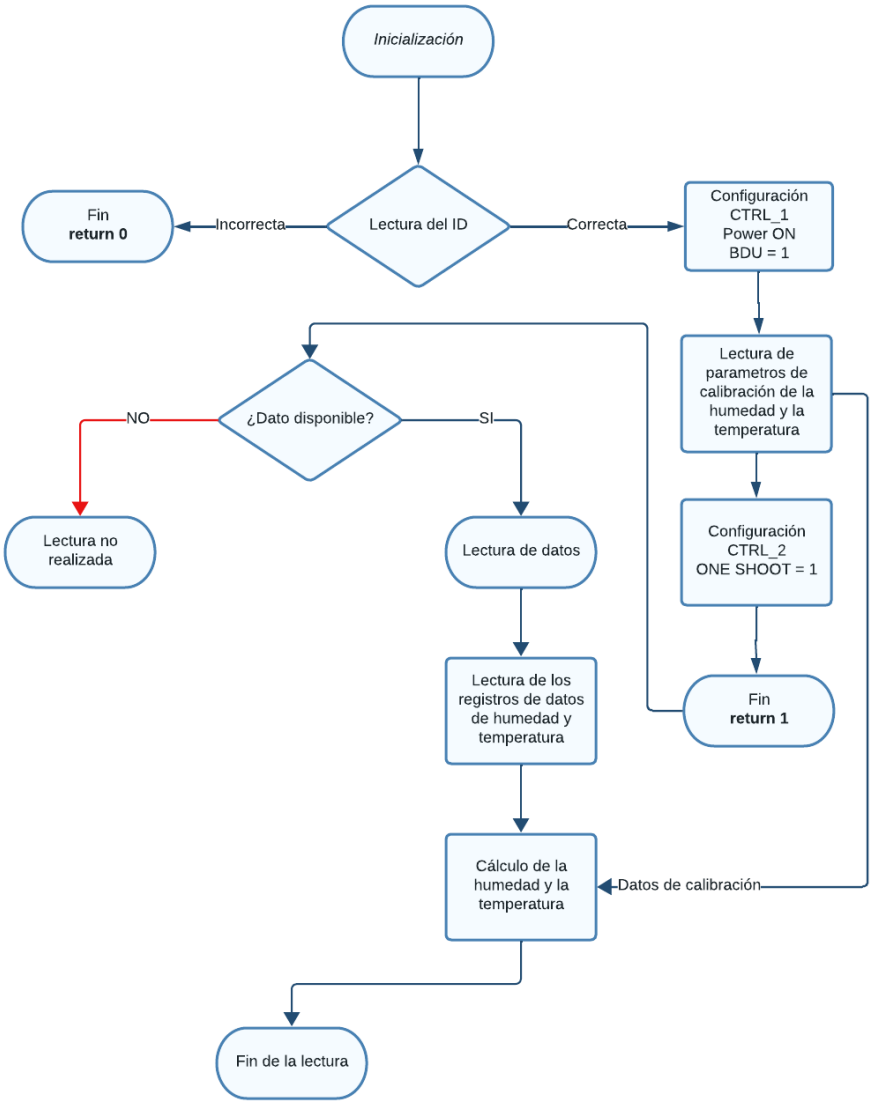

Ilustración 26. Diagrama de flujo del funcionamiento del sensor WSEN-HIDS.

Se comienza realizando la inicialización del sensor. En esta inicialización, en primer lugar, se realiza la lectura del ID del sensor para comprobar la disponibilidad de este. A continuación, se configuran los CTRL_1 y CTRL_2 y además en este sensor se leen algunos parámetros de calibración necesarios para el cálculo de la humedad y la temperatura.

En la configuración del registro CTRL_1 se aplican los siguientes cambios. Se activa el bit PD y BDU, el primero activa el sensor y el segundo hace los datos no se actualicen hasta que no se haya accedido a todos los registros de lectura del dato. Al igual que ocurre con el WSEN-PADS también indica que es recomendable hacerlo con este sensor.

Por último, en este registro también se configura el ODR como “00” para poner el sensor en modo “one shoot”.

12.3 CTRL_1 (0x20)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Type
PD	Reserved				BDU	ODR[1:0]		R/W

Table 16: *CTRL_1* register

bits	Description
PD	Power down. By default, PD: 0 which is power down mode. PD: 1 Continuous mode for active measurement
BDU	Block data update. 0: Continuous update, 1: Output registers are not updated until MSB and LSB is read
ODR[1:0]	Output data rate of the humidity and temperature value

Table 17: CTRL_1 register description

Ilustración 27. Configuración del registro CTRL_1 del sensor WSEN-HIDS.

ODR[1:0]	Output data rate
00	One shot mode
01	1 Hz
10	7 Hz
11	12.5 Hz

Table 18: Output data rate

Ilustración 28. Modos de configuración de los bits ODR.

Después de configurar el registro CTRL_1 se leen los parámetros de calibración del sensor que más tarde serán necesarios para realizar el cálculo de la humedad y la temperatura.

Este sensor tiene una peculiaridad que hay que considerar al leer múltiples registros consecutivos, función útil en este apartado, ya que son varios registros de calibración y están organizados consecutivamente.

El data sheet indica que para realizar una lectura o escritura de varios registros consecutivos es necesario cambiar la dirección del primer registro al que se quiere acceder. En este caso hay que poner el bit más significativo a “1”. Normalmente esto es necesario para los sensores que no tienen autoincremento de la dirección para lecturas múltiples, como es el caso.

6.4 I²C Multiple bytes read operation

In order to read multiple bytes incrementing the register address, it is necessary to assert the most significant bit of the sub-address field (output register address). The bit [7] must be equal to 1 while bit [6-0] represents the address of the first register to be read. Here is an example of how to read 4 bytes of data from output registers by requesting the data only from 0x28.

- To read multiple bytes from the output registers 0x28 to 0x2B, the first register address must be changed from 0x28 (b0010 1000) to 0xA8(b1010 1000 (MSB is changed from '0' to '1').
- After sending I²C address, start with reading from the register 0xA8 and request 4 bytes to read.
- The received 4 bytes gives the content from registers 0x28 to 0x2B.

There is no auto increment bit (enable/disable) implemented in the control register for multiple bytes of read.

Ilustración 29. Funcionamiento de la lectura de varios registros consecutivos en el sensor WSEN-HIDS.

Al leer los datos brutos de medición del sensor hay que tener en cuenta esto de nuevo. Además, es necesario utilizar los datos de calibración leídos en la inicialización que se han almacenado en variables declaradas en el global del archivo .c del sensor. Así podemos acceder a ellas sin problemas para calcular la humedad y la temperatura.

El cálculo de la humedad y la temperatura consta de bastantes pasos, los cuales están detallados correctamente en el "data sheet". Las unidades de las medidas son ºC y % de humedad relativo.

$$Temperature = \frac{(T1\_degC - T0\_degC) * (T\_OUT - T0\_OUT)}{(T1\_OUT - T0\_OUT)} \quad (3)$$

$$Temperature(degree\ Celsius) = Temperature + T0\_degC \quad (4)$$

Ilustración 30. Fórmula para el cálculo de la temperatura.

$$Humidtiy = \frac{(H1\_rH - H0\_rH).(H\_T\_OUT - H0\_T0\_OUT)}{(H1\_T0\_OUT - H0\_T0\_OUT)} \quad (1)$$

$$Humidity(\%\ relative\ Humidity) = Humidity + H0\_rH \quad (2)$$

Ilustración 31. Fórmula para el cálculo de la humedad.

- DATA_HIDS_DISP. Función que detecta si el sensor tiene datos actualmente. Funciona como una espera dinámica.
- INIT_WSENHIDS. Para realizar la lectura y que el sensor se mantenga configurado tal y como deseamos, se diseñó una función encargada de la inicialización de este, que configura algunos parámetros del sensor.
- HUM_TEMP_WSENHIDS. Función que realiza la lectura y el cálculo de la humedad y la temperatura.
- OFF_WSENHIDS. Función que pone el sensor en modo Stand-by.

Las funciones se llaman en la función “main” en el siguiente orden.

```
321         if(INIT_WSENHIDS(&hi2c1) == 1)                                  //hids exterior
322         {
323             Sens_Disp.hids = 1;
324             DATA_HIDS_DISP(&hi2c1);
325             HUMTEMP_WSENHIDS(&hi2c1, &Param_Sens);
326             myFtoa(Param_Sens.humedad, Hum_ext_s);
327             myFtoa(Param_Sens.temp_hids, Temp_ext_s);
328         }
329         else
330         {
331             Hum_ext_s[0] = 48;
332             Temp_ext_s[0] = 48;
333         }
```

Ilustración 32. Líneas de código para la ejecución de las funciones diseñadas para el sensor WSEN-HIDS.

En primer lugar, se llama a la función de inicialización, que además de realizar la configuración del sensor nos permite saber si el sensor está operativo, si es el caso se realiza una espera con la función que detecta si hay un dato disponible y a continuación se realiza la lectura y el cálculo de los datos. Por último, se almacenan los datos como un “array” en las variables Hum_ext_s y Temp_ext_s, para ello se utiliza la función myFtoa que convierte un “float” en un array. Esto es debido a que es necesario para realizar él envió a través del protocolo HTTP.

También, en esta sección, dependiendo de si el sensor se inicializa correctamente o no se pone a “1” la bandera Sens_Disp.hids que permite decidir si se quiere enviar o no el dato de presión más adelante. Ya que si la inicialización no es correcta es importante no realizar el envío de este dato.

En el modo “single conversion” no es necesario apagar el sensor al terminar la lectura.

4.4. Diseño de la librería del sensor AS7341 respondiendo a los criterios de diseño.

Al igual que en el diseño de los dos sensores anteriores, el desarrollo de la solución del sensor AS7341, a pesar de estar diseñado por un fabricante diferente, guarda mucha relación con los sensores WSEN-PADS y WSEN-HIDS. Debido, principalmente, a que comparten la característica de que emplean la interfaz I2C como método de comunicación. Por lo tanto, se mantendrá el empleo de las funciones mencionadas en el primer apartado para realizar las lecturas y escrituras de los registros necesarias para desarrollar el algoritmo de lectura del sensor.

En cuanto al esquema seguido para realizar las lecturas de los diferentes canales de luminosidad requiere de más procesos que los diseñados para los otros sensores, ya

que este sensor es algo más complejo, debido principalmente a su arquitectura y como realiza la conversión del dato analógico a digital.

El sensor dispone de 6 ADC para realizar la conversión de 8 canales de luminosidad más los canales NIR, CLEAR y FLICKER. Esto supone que no se pueda realizar la lectura de todos los canales simultáneamente. Para solucionar esto, el sensor incluye un multiplexor que permite conmutar las entradas de los ADC's.

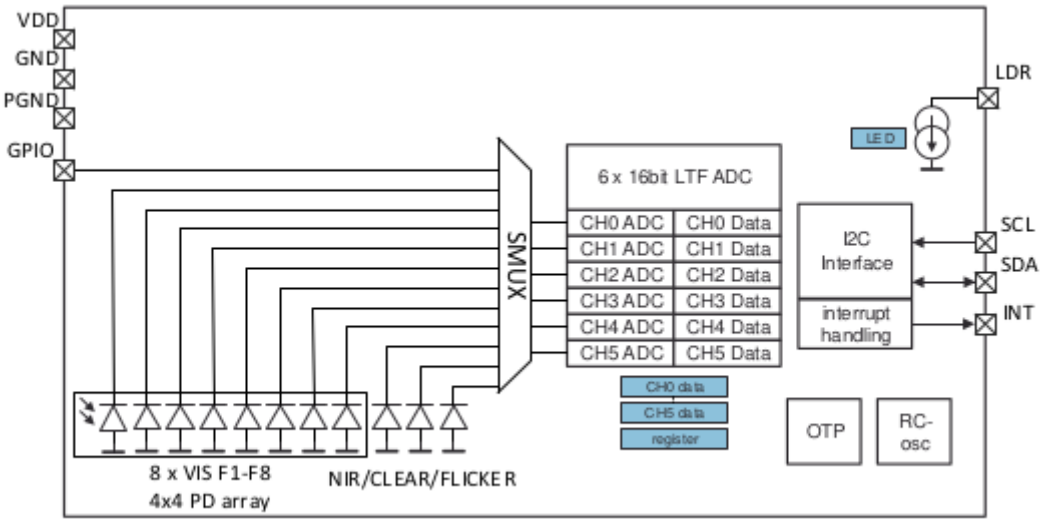

Ilustración 33. Arquitectura del sensor multiespectral AS7341.

Así, haciendo uso de este multiplexor es posible realizar la lectura de todos los canales del sensor en dos fases. La primera de ellas con el multiplexor configurado para realizar la lectura de F1, F2, F3, F4, CLEAR y NIR, y la segunda para realizar la lectura de F5, F6, F7, F8 y, CLEAR y NIR de nuevo. A la hora de realizar esta función se presentó la complicación de que el "data sheet" del sensor no incluye ninguna información de cómo se realiza la configuración del multiplexor. Para solucionar esto se empleó la librería que aporta el fabricante, ya que en ella parecen los datos que son necesarios escribir en los registros pertenecientes al multiplexor.

Al igual que para los dos sensores anteriores el código de este sensor se separa entre inicialización y lectura de datos. En la inicialización se configuran los valores de ATIME, ASTEP y ganancia del sensor. Mientras que en la lectura de los datos se realizar los diferentes algoritmos para configurar el multiplexor y hacer la lectura de los datos de los canales lumínicos.

En el siguiente diagrama se puede ver mejor los pasos que se han seguido para realizar todo el proceso.

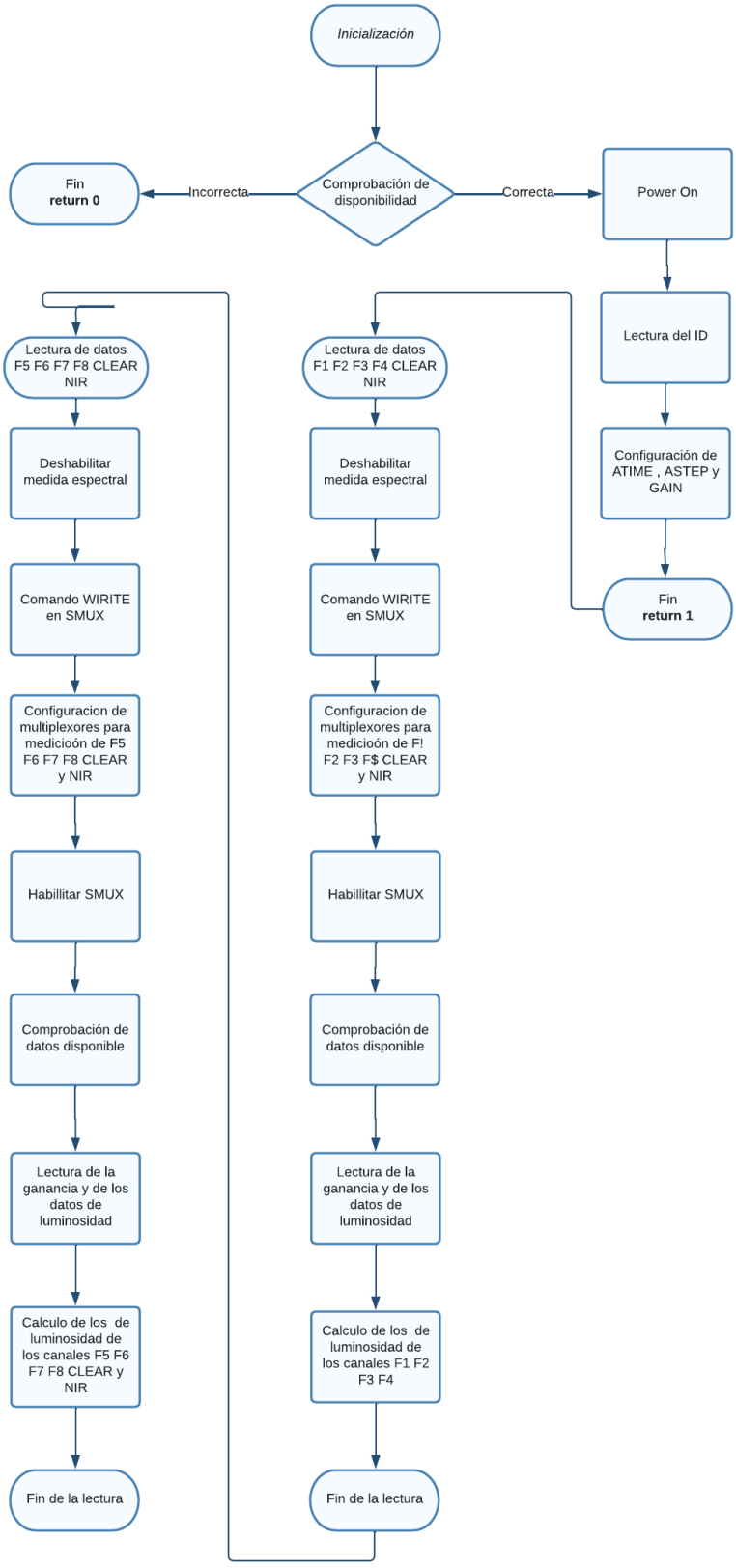

Ilustración 34. Diagrama de flujo del funcionamiento del sensor AS7341.

Para la configuración del ASTEP y ATIME se tiene en cuenta la recomendación del fabricante en el data sheet. Lo que resulta en un tiempo de integración de 50 ms.

$$t_{int} = (ATIME + 1) \times (ASTEP + 1) \times 2.78\mu s$$

The reset value for ASTEP is 999 (2.78ms) and the recommended configuration for these two registers is ASTEP = 599 and ATIME = 29, which results in an integration time of 50ms. It is not allowed that both settings –ATIME and ASTEP – are set to "0".

Ilustración 35. Formula y valores recomendados para el tiempo de integración.

La configuración de la ganancia automática se realiza accediendo al registro correspondiente especificado en el data sheet. Para activarla solo es necesario poner en 1 el bit 2 de este registro.

CFG8 Register (Address 0xB1)

Figure 52:
CFG8 Register

Addr: 0xB1		CFG8		
Bit	**Bit Name**	**Default**	**Access**	**Bit Description**
7:6	FIFO_TH	2	R/W	**FIFO Threshold.** Sets a threshold on the FIFO level that triggers the first FIFO buffer interrupt (FINT). **VALUE** / **FIFO_LVL** 0 / 1 1 / 4 2 / 8 3 / 16
5:4	reserved	0		reserved
3	FD_AGC	1	RW	**Flicker Detect AGC Enable**. If set, device uses automatic gain control for the flicker detect engine to maximize flicker signal and avoid saturation.
2	SP_AGC	0	RW	**Spectral AGC enable.** If asserted, device uses automatic gain control for the spectral engines to maximize signal while avoiding saturation.
1	reserved	0		reserved
0	reserved	0		reserved

Ilustración 36. Configuración del registro CFG8 del sensor AS7341.

Debido a esta configuración la ganancia será variable en función de la cantidad de luz que llegue al sensor por lo tanto es importante realizar una lectura de la ganancia con la que se ha tomado cada muestra para luego poder realizar la conversión de "data_raw" a "basic_counts".

Para obtener el valor de esta ganancia se accede un registro de "status" que contiene este valor.

ASTATUS Register (Address 0x60 or 0x94)

Figure 68:
ASTATUS Register

Addr: 0x60 and 0x94	ASTATUS			
Bit	**Bit Name**	**Default**	**Access**	**Bit Description**
7	ASAT_STATUS	0	R, SC	**Saturation Status.** Indicates if the latched data is affected by analog or digital saturation.
6:4	reserved	0	R	reserved
3:0	AGAIN_STATUS	0	R, SC	**Gain Status.** Indicates the gain applied for the spectral data latched to this ASTATUS read. The gain from this status read is required to calculate spectral results if AGC is enabled.

Ilustración 37. Configuración del registro ASTATUS del sensor AS7341.

A continuación, se emplea este valor junto al tiempo de integración configurado anteriormente para convertir el dato obtenido a “basic_counts”. Este formato permite comparar los datos entre sí. Esto no podría hacerse con los datos en formato raw ya que han sido obtenido con diferentes ganancias y sus escalas son diferentes. Por lo tanto, es necesaria esta conversión para que los datos sean manejables por las otras áreas del proyecto.

De nuevo, se separa el código en tres bloques diferentes para una mejor organización de este.

- AS7341_Init. Función destinada a realizar la inicialización y configuración de sensor. En ella se pone en “power on” el sensor y se configuran ATIME, ASTEP y la ganancia.
- AS7341_Read_Channels. Función que realiza la lectura de los 10 canales de luminosidad.
- AS7341_Off. Función que apaga el sensor AS7341.

Las funciones se llaman en la función “main” en el siguiente orden.

```
433         if((AS7341_Init(&hi2c1) == 1))
434         {
435             Sens_Disp.AS7341 = 1;
436             AS7341_Read_Channels(&hi2c1, &Param_AS7341);
437         }
438         AS7341_Off(&hi2c1);
```

Ilustración 38. Líneas de código para la ejecución de las funciones diseñadas para el sensor AS7341.

Primero se llama a la función de inicialización que detecta si el sensor está disponible y configura el sensor correctamente y a continuación si la inicialización ha sido exitosa se realiza la lectura de todos los canales. Por último, se llama a la función de apagado del sensor para minimizar el consumo de energía.

4.5. Diseño de la librería del sensor URM06.

Para el diseño de esta librería ha sido necesario desarrollar un conjunto de funciones capaces de obtener el tiempo que está en alta una señal cuadrada. La siguiente imagen obtenida del “data sheet” del sensor, ilustra como se incluye el dato en la señal. La relación entre el tiempo que la señal está en alta y el valor de distancia de 1 us por cada milímetro de distancia.

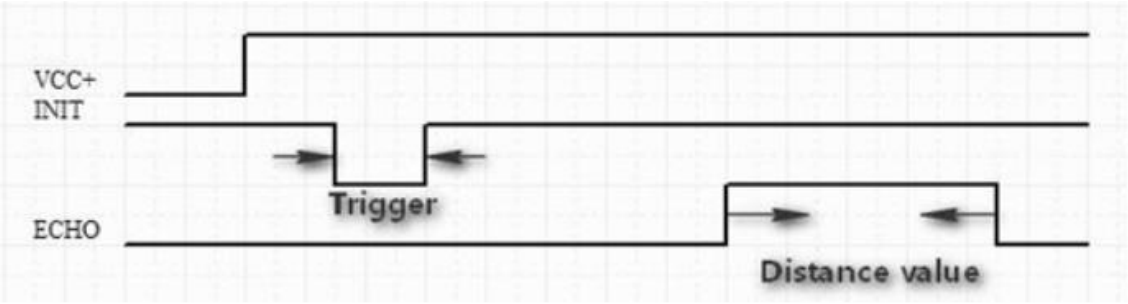

Ilustración 39. Funcionamiento de los pines del sensor URM06.

Para realizar esta lectura emplearemos un pin de propósito general del microcontrolador, capaz de leer el estado de la señal ECHO. Así, usando también temporizadores e interrupciones, seremos capaces de obtener el tiempo que la señal está en alta.

El principio básico de esta lectura es realizar una medición del tiempo que transcurre entre que la señal pasa de alta a baja.

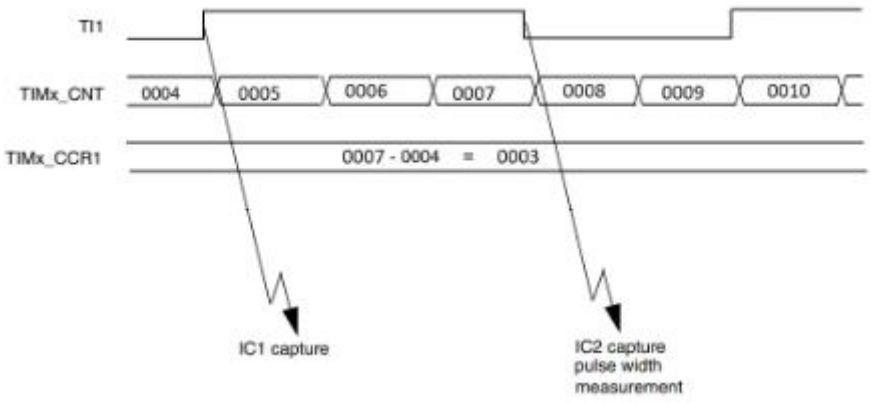

Ilustración 40. Funcionamiento del contador del microcontrolador STM32.

Para ello es necesario inicializar un temporizador que recoja los cambios de estado que se produzcan en el pin correspondiente. En este caso el PA3. Esto se realiza fácilmente haciendo uso de la interfaz gráfica de STMCube, seleccionando las siguientes opciones.

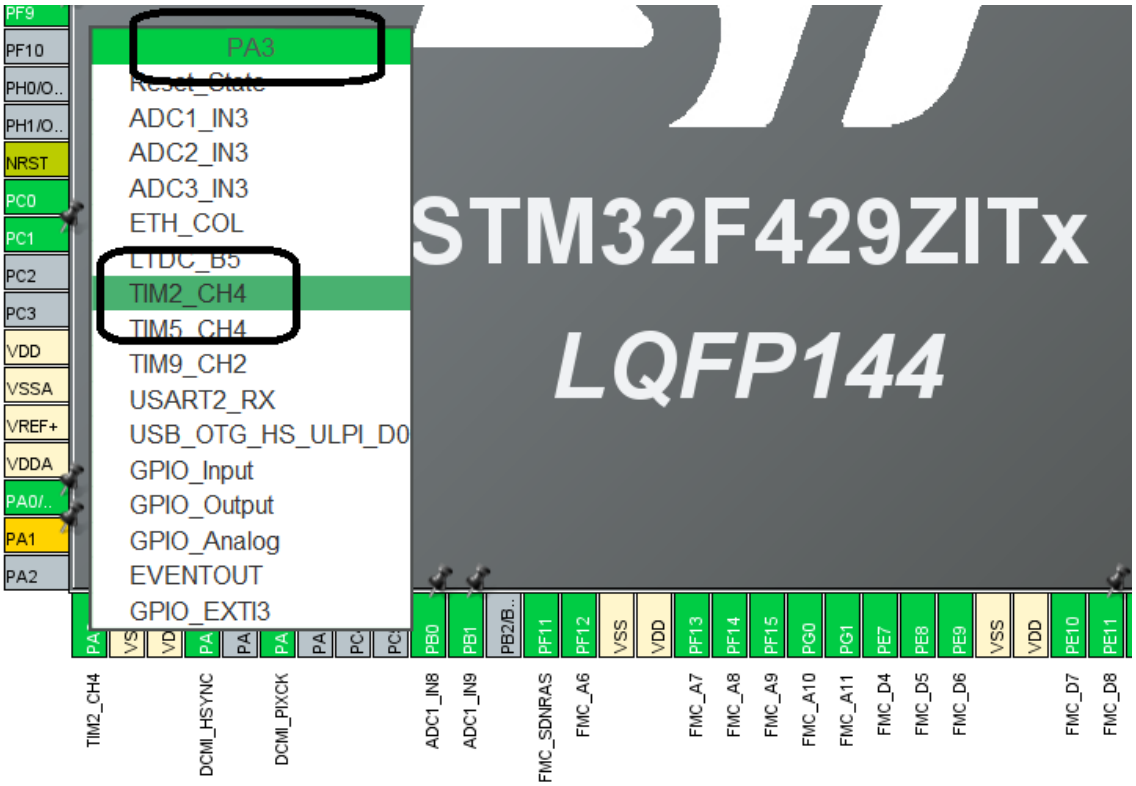

Ilustración 41. Interfaz de STMCubeIDE para la configuración del temporizador.

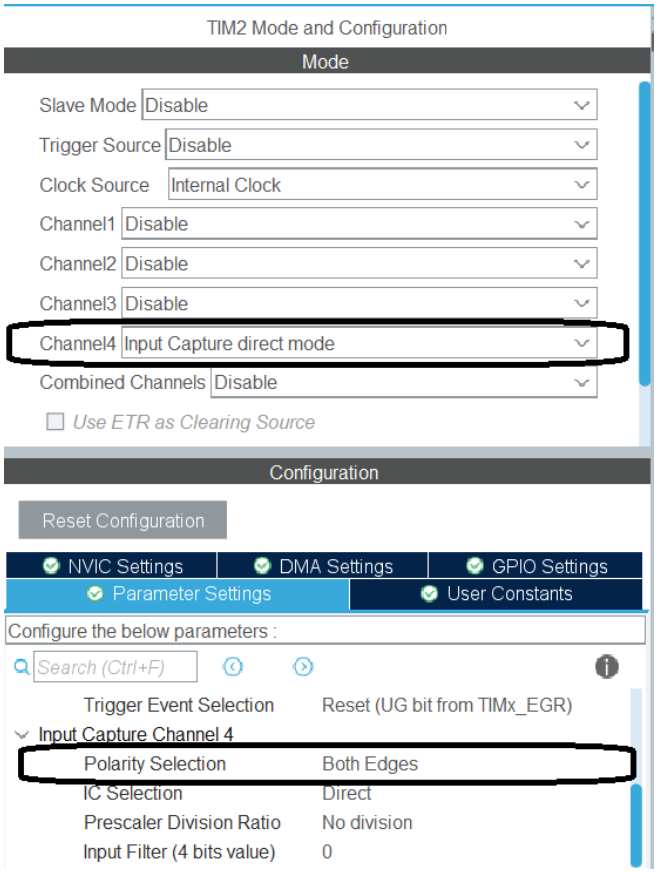

Ilustración 42.Interfaz de STMCubeIDE para configuración del temporizador.

Así inicializamos uno de los temporizadores de propósito general del micro, el temporizador 2, para detectar los flancos de subida y de bajada que ocurran en el pin PA3. Que es el pin al que está conectado la señal “ECHO” del sensor. Esto se consigue seleccionando el modo “Input Capture” y “Both Edges” en “Polarity Selection”. Si se quiere calcular el periodo o frecuencia de la señal, en vez de seleccionar estos parámetros se debe seleccionar los que permitan recoger solo el flanco de subida o de bajada para saber el tiempo que transcurre durante un ciclo completo de la señal.

Como el propósito de este apartado es detectar el tiempo que la señal está en alta es necesario capturar ambos flancos y calcular el tiempo que ha transcurrido entre ambos.

Para que el cálculo sea más sencillo se configura el temporizador de la siguiente manera.

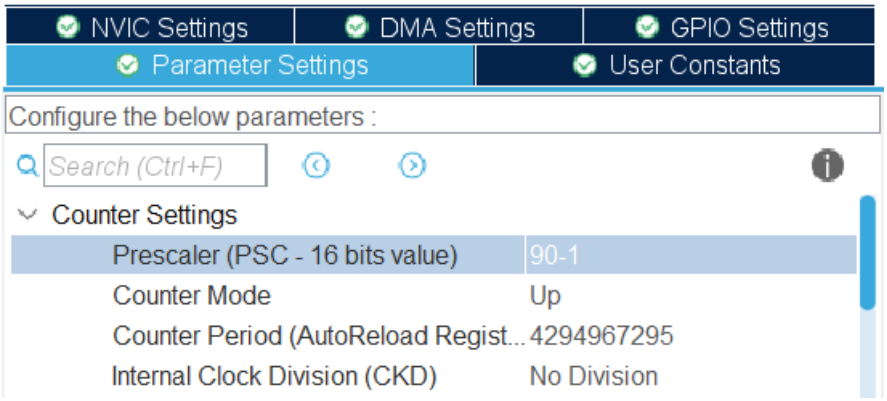

Ilustración 43. Interfaz de STMCubeIDE para la configuración del temporizador.

Fijando un “Prescaler” de 90 y debido a que el temporizador interno del micro funciona a 90 MHz, obtenemos que el contador se actualiza con una frecuencia de 1 MHz, es decir, cada 1 microsegundo. El periodo del contador lo fijamos al máximo ya que, para este caso el valor es irrelevante.

Así, usando la función que se ejecuta cuando se produce un evento en el pin que se está usando como “Input Capture, llamada “HAL_TIM_IC_CaptureCallback”. La cual, debido a esto, se ejecuta cada vez que la señal pasa de alta a baja y viceversa. Podemos calcular el tiempo que la señal está en alta mediante el siguiente código.

```
2674  void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
2675  {
2676      uint8_t pinA3 = 0;
2677      if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4)  //if para comprobar que la interrupción corresponde con el canal utilizado
2678      {
2679          pinA3 = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3);
2680          if (pinA3 == 1) //if para comprobar que la señal esta en alta
2681          {
2682              IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4); // se lee el primer valor
2683          }
2684
2685          else   //else del if anterior para ejecutar el codigo cuando la señal este en baja
2686          {
2687              IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4);  // se lee el segundo valor
2688
2689              if (IC_Val2 > IC_Val1)
2690              {
2691                  Difference = IC_Val2-IC_Val1;
2692              }
2693
2694              float refClock = TIMCLOCK/(PRESCALAR);
2695              float mFactor = 1000000/refClock;
2696
2697              usWidth = Difference*mFactor;
2698          }
2699      }
2700  }
```

Ilustración 44. Función que se ejecuta cuando se produce algún evento en el pin PA3.

En él, se almacena el valor del contador cuando se produce un evento en el pin, como puede ser tanto de bajada como de subida, se necesita leer el valor del pin para saber si la señal está en baja o en alta. La primera lectura debe ser cuando la señal este en alta y la segunda cuando este en baja. Si no realizásemos esta comprobación no sabríamos si el periodo que estamos recogiendo es cuando la señal está en baja o en alta, que es el que nos interesa para calcular la distancia.

Empleando las variables predefinidas “TIMCLOCK” y “PRESCALAR” podemos adaptar el código para cualquier frecuencia y divisor de frecuencia que hayamos definido. En este caso, el cálculo no sería necesario porque la diferencia ya está obtenida en microsegundos por los parámetros seleccionados anteriormente.

Una vez que esta función ha sido ejecutada en el evento en el que la señal pasa de alta a baja, la anchura del pulso se almacena en la variable usWidht. Debido a que esta variable esta declara como una función global en el archivo main del proyecto, es posible acceder a ella en todo el. De esta forma podemos tratar el dato en otra parte del código mientras este se va actualizando automáticamente debido a la interrupción que hemos programado.

Para tratar este dato y que la interrupción que se genera cuando se produce uno de los flancos solo se produzca cuando deseemos realizar la lectura del sensor, se utiliza el siguiente conjunto de líneas de código.

```
360             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
361             HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4);
362
363             HAL_Delay(500);
364
365             if (usWidth > 0){
366
367                 debug_serie("[URM13.c] URM13 encontrado\n");
368
369             while(i < 50 && exit == 0)
370             {
371                     if(usWidth > 500 && usWidth < 10000){
372                         value[i]= usWidth;
373                         HAL_Delay(500);
374                         e_sprintf(msj, "Distancia correcta %d: %d\n", i, value[i]);
375                         debug_serie(msj);
376                         i++;
377                     }
378                     else
379                     {
380                         e_sprintf(msj, "Distancia incorrecta %d: %d\n", j, usWidth);
381                         debug_serie(msj);
382                         HAL_Delay(500);
383                     }
384                     j++;
385                     if(j > 200)
386                     {
387                         exit = 1;
388                     }
389             }
```

Ilustración 45. Líneas de código que realizan la lectura del dato de distancia del sensor URM06.

En ella, se comienza poniendo en baja el pin trigger del sensor, de esta forma solo estará lanzando mediciones durante el periodo que se produzca la lectura. Luego, se activan las interrupciones del temporizador 2 para que la función diseñada antes se ejecute cuando se produzca algún evento de flanco. En este periodo estas dos funciones se ejecutan “simultáneamente” para recoger la distancia correctamente.

A continuación, esta función contiene algún algoritmo más como un cálculo de la mediana, descarte de ciertos datos que mejoran la medida del sensor en ciertas condiciones.

Para el cálculo de la mediana se recogen mediciones hasta que 50 de ellas son mayores que 500 mm y menores de 10 metros. Cuando se recogen 50 muestras correctas o 200 muestras, ya sean correctas o no, se termina de recoger datos.

Una vez obtenidos los datos se ordena el vector que contiene las muestras y si la mediana es mayor que 500 mm se almacena el dato en la estructura “Param_Sens” para a continuación ser enviado junto a las imágenes. En último lugar se desactivan las interrupciones y se apaga el sensor para dejar de recoger muestras.

```
392                 for(k=0; k< 50; k++){
393                     for(l=0; l<50; l++){
394                         if(value[l] > max){
395                             max= value[l];
396                             orden[k] = max;
397                             pos = l;
398                         }
399                     }
400                     max=0;
401                     value[pos] = 0;
402                 }
403 
404                 if(orden[i/2] > 500)
405                 {
406                     Sens_Disp.urm13 = 1;
407                     distance = orden[i/2];
408                     Param_Sens.distancia = distance;
409                     e_sprintf(msj, "\nDistancia Mediana: %d\n\n", distance);
410                     debug_serie(msj);
411                 }
412                 else
413                 {
414                     e_sprintf(msj, "\nDistancia Mediana Incorrecta <500: %d\n\n", distance);
415                     debug_serie(msj);
416                 }
417 
418             HAL_TIM_IC_Stop_IT(&htim2, TIM_CHANNEL_4);
419 
420             HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
421         }
```

Ilustración 46. Líneas de código que realizan la lectura del dato de distancia del sensor URM06.

4.6. Estructura final del código.

En este último apartado se explicará cómo están organizados los archivos diseñados dentro del proyecto y se explicará en que parte de la estructura del código del equipo se han incluido las lecturas de los datos de estos sensores.

El proyecto está diseñado con la siguiente estructura de archivos.

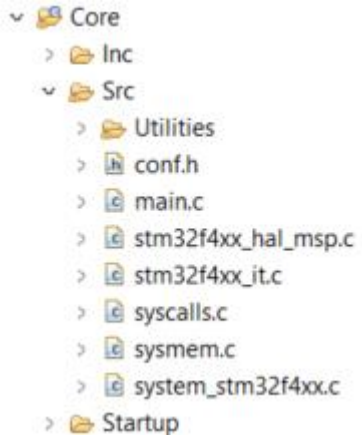

Ilustración 47. Organización de los archivos del proyecto.

La carpeta principal que contiene todos los archivos es la “src”. En ella está el archivo “main” que es el contiene la función principal que se ejecuta en el micro. Dentro de la carpeta “Utilities” están los diferentes archivos que sirven como apoyo para ser capaces de controlar el total de dispositivos que emplea la estación. Cámaras, sensores, modem,

almacenamiento. Es esta carpeta donde se sitúan los archivos diseñados para manejo de los sensores de este proyecto.

Ilustración 48. Organización de los archivos del proyecto.

Además, de los archivos .c con las funciones señaladas, cada sensor tiene un archivo .h que contiene las declaraciones de las funciones. De esta forma al llamar con #include desde el “main” a estos archivos tenemos total acceso a estas funciones.

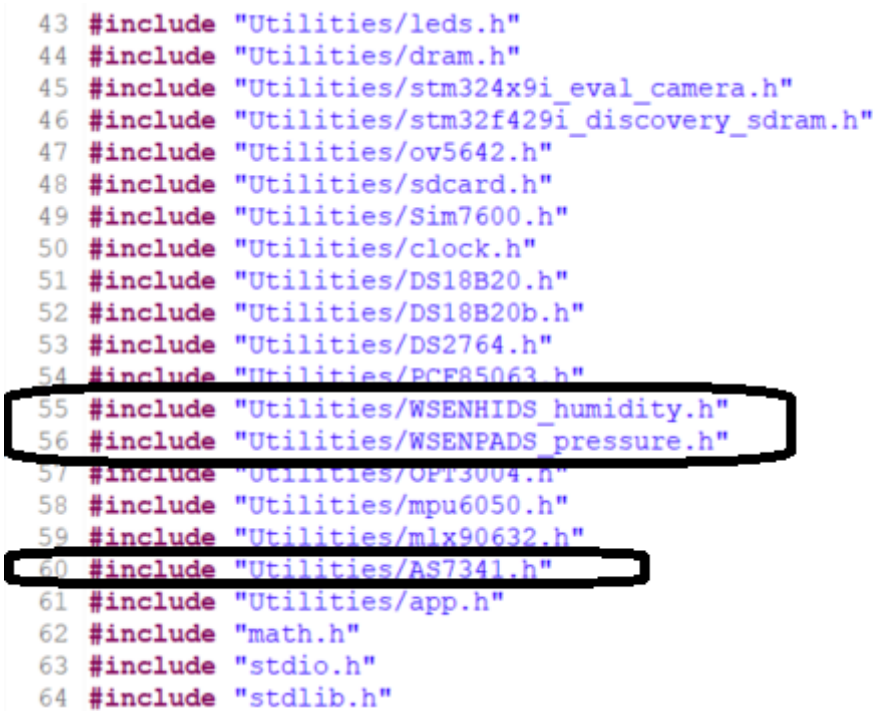

Ilustración 49. "Includes" del archivo principal del proyecto.

La estructura del código del sensor URM06 funciona algo diferente debido a la interrupción que necesita para la lectura del dato. La función está declarada por el entorno de desarrollo en el archivo “main” y no puede ser movida. El código de este sensor se diseñó en este archivo para acceder al dato de distancia.

Una vez que estas funciones son accesibles desde el archivo “main” del proyecto, se realiza las llamadas explicadas anteriormente para la obtención de los datos de cada sensor. Estas llamadas se sitúan en una función llamada “TomarDatos” que incluye el resto de funciones que leen el dato de todos los sensores disponibles en la estación. Así, el código queda totalmente integrado en el proyecto.

Después de haber llamado a esta función las estructuras y variables que contienen los datos de los sensores y su disponibilidad se emplean para diseñar un “string” en formato JSON que será empleado para enviar toda esta información al servidor a través del protocolo HTTP, concluyendo así el ciclo de trabajo de la estación.

5. Implementación.

Para poder acceder al código de una forma sencilla he subido todos los archivos diseñados a la web github. Con el siguiente enlace se puede acceder a todos los archivos de esta librería.

https://github.com/arribaslopezjesus/STM32_sensores

6. Informe de resultados.

Para hacer constar los resultados obtenidos de la ejecución del código diseñado se presentarán los datos leídos por los sensores en dos pruebas diferentes. Una de ellas será la visualización de las gráficas que se forman en la plataforma destinada a la

visualización de los datos e imágenes que envían las estaciones. La otra prueba serán las pruebas experimentales realizadas en el laboratorio. En estas pruebas el código de la estación es un bucle en el que solo se realizan lecturas de los sensores empleados en el proyecto, para así facilitar la comprobación del funcionamiento de estos sin necesidad de que la estación realice el resto de los procesos.

A continuación, se exponen los registros obtenidos durante la prueba realizada en el laboratorio. En estos registros se puede observar el funcionamiento de los sensores de forma continua en bucles de mediciones individuales de unos 20 segundos para cada sensor. Se pueden ver los valores obtenidos para cada uno de ellos en un entorno controlado.

[WSEN_HIDS.c] Temperatura:24.80052 Humedad:65.82687 %

[WSEN_HIDS.c] Temperatura:24.81877 Humedad:65.81133 %

[WSEN_HIDS.c] Temperatura:24.81877 Humedad:65.83076 %

[WSEN_HIDS.c] Temperatura:24.80052 Humedad:65.75695 %

[WSEN_HIDS.c] Temperatura:24.81877 Humedad:65.82687 %

[WSEN_HIDS.c] Temperatura:24.83702 Humedad:65.85406 %

[WSEN_HIDS.c] Temperatura:24.81877 Humedad:65.90844 %

[WSEN_HIDS.c] Temperatura:24.81877 Humedad:65.90844 %

[WSEN_HIDS.c] Temperatura:24.83702 Humedad:65.82299 %

[WSEN_HIDS.c] Temperatura:24.83702 Humedad:65.81910 %

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.06999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.08999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.10000

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.12999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.13999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.15999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.18000

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.18999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.20999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.22999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.25000

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.25999

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.28000

[WSENPADS.c] Presion:91.99999 kPa Temperatura:26.29999

[URM13.c] URM13 encontrado

Distancia correcta 0: 1877

Distancia correcta 1: 1874

Distancia correcta 2: 1876

Distancia correcta 3: 1872

Distancia correcta 4: 1874

Distancia correcta 5: 1876

Distancia correcta 6: 1878

Distancia correcta 7: 1877

Distancia correcta 8: 1878

Distancia correcta 9: 1879

Distancia correcta 10: 1878

Distancia correcta 11: 1877

Distancia correcta 12: 1878

Distancia correcta 13: 1878

Distancia correcta 14: 1877

Distancia correcta 15: 1878

Distancia correcta 16: 1877

Distancia correcta 17: 1876

Distancia correcta 18: 1878

Distancia correcta 19: 1876

Distancia correcta 20: 1876

Distancia correcta 21: 1878

Distancia correcta 22: 1877

Distancia incorrecta 23: 193

Distancia incorrecta 24: 194

Distancia incorrecta 25: 193

Distancia incorrecta 26: 193

Distancia incorrecta 27: 193

Distancia incorrecta 28: 193

Distancia incorrecta 29: 193

Distancia incorrecta 30: 194

Distancia incorrecta 31: 193

Distancia incorrecta 32: 200

Distancia correcta 23: 1877

Distancia correcta 24: 1877

Distancia correcta 25: 1876

Distancia correcta 26: 1878

Distancia correcta 27: 1877

Distancia correcta 28: 1876

Distancia correcta 29: 1875

Distancia correcta 30: 1877

Distancia correcta 31: 1872

Distancia correcta 32: 1874

Distancia correcta 33: 1873

Distancia correcta 34: 1875

Distancia correcta 35: 1871

Distancia correcta 36: 1872

Distancia correcta 37: 1872

Distancia correcta 38: 1873

Distancia correcta 39: 1872

Distancia correcta 40: 1872

Distancia correcta 41: 1873

Distancia correcta 42: 1874

Distancia correcta 43: 1874

Distancia correcta 44: 1875

Distancia correcta 45: 1873

Distancia correcta 46: 1875

Distancia correcta 47: 1874

Distancia correcta 48: 1874

Distancia correcta 49: 1875

Distancia Mediana: 1876

[AS7341.c] Medidas de los diferentes canales del AS7341. Formato RAW y BASIC

F1_ADC = 424 F1 = 0.033124

F2_ADC = 643 F2 = 0.050234

F3_ADC = 827 F3 = 0.064609

F4_ADC = 1044 F4 = 0.081562

F5_ADC = 1319 F5 = 0.103046

F6_ADC = 1439 F6 = 0.112421

F7_ADC = 1653 F7 = 0.129140

F8_ADC = 1597 F8 = 0.124765

CLEAR_ADC = 6676 CLEAR = 0.521562

NIR_ADC = 3939 NIR = 0.307734

[AS7341.c] Medidas de los diferentes canales del AS7341. Formato RAW y BASIC

F1_ADC = 424 F1 = 0.033124

F2_ADC = 642 F2 = 0.050156

F3_ADC = 826 F3 = 0.064531

F4_ADC = 1044 F4 = 0.081562

F5_ADC = 1317 F5 = 0.102890

F6_ADC = 1439 F6 = 0.112421

F7_ADC = 1653 F7 = 0.129140

F8_ADC = 1596 F8 = 0.124687

CLEAR_ADC = 6677 CLEAR = 0.521640

NIR_ADC = 3948 NIR = 0.308437

[AS7341.c] Medidas de los diferentes canales del AS7341. Formato RAW y BASIC

F1_ADC = 425 F1 = 0.033203

F2_ADC = 644 F2 = 0.050312

F3_ADC = 828 F3 = 0.064687

F4_ADC = 1045 F4 = 0.081640

F5_ADC = 1318 F5 = 0.102968

F6_ADC = 1440 F6 = 0.112500

F7_ADC = 1653 F7 = 0.129140

F8_ADC = 1598 F8 = 0.124843

CLEAR_ADC = 6684 CLEAR = 0.522187

NIR_ADC = 3962 NIR = 0.309531

Los resultados expuestos a continuación proceden de la plataforma de visualización que se entrega al usuario para que pueda acceder a los datos e imágenes enviados por las estaciones. El código empleado en estas mediciones es el código completo. En el cual

las funciones diseñadas en la librería están integradas con el resto de código de la estación.

En esta plataforma de visualización se puede seleccionar los datos que envía la estación y dibujar una gráfica que represente este dato durante el periodo que deseemos. La estación es una situada en la ciudad de Zamora y en las imágenes se pueden ver las variaciones que se producen en los sensores durante diferentes periodos de tiempo.

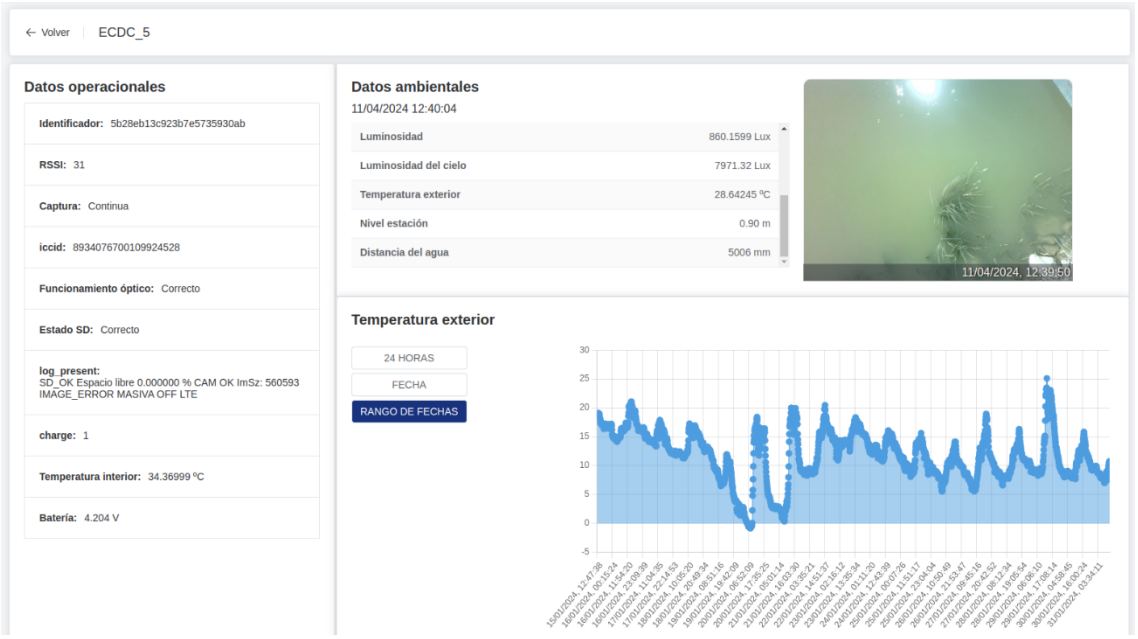

Ilustración 50. Captura de la plataforma para la visualización del dato de temperatura exterior.

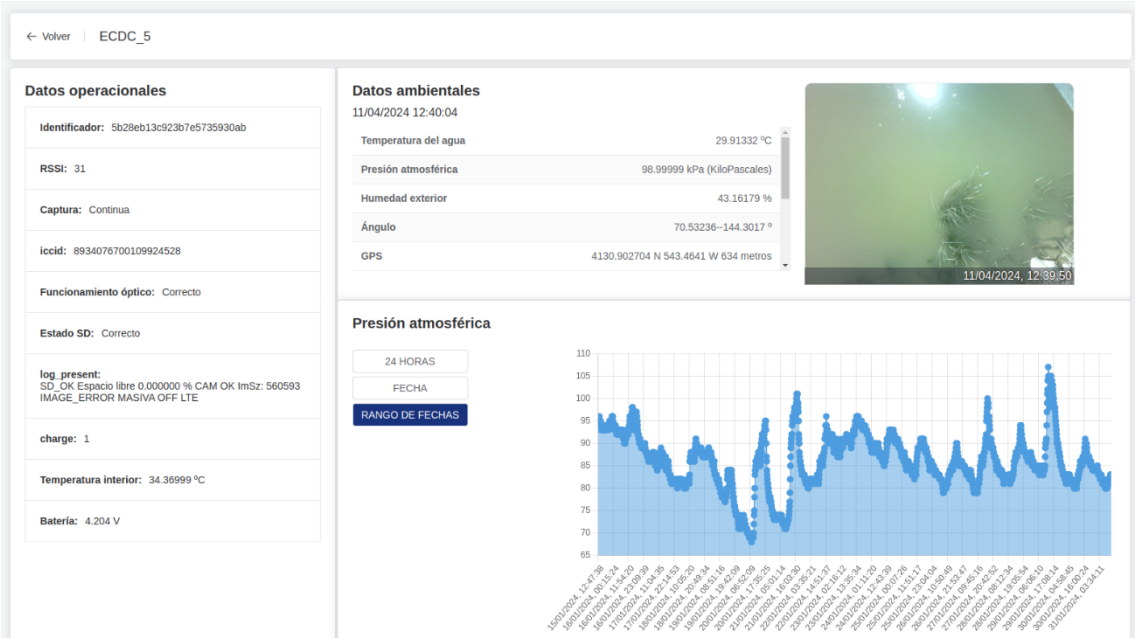

Ilustración 51. Captura de la plataforma para la visualización del dato de presión atmosférica.

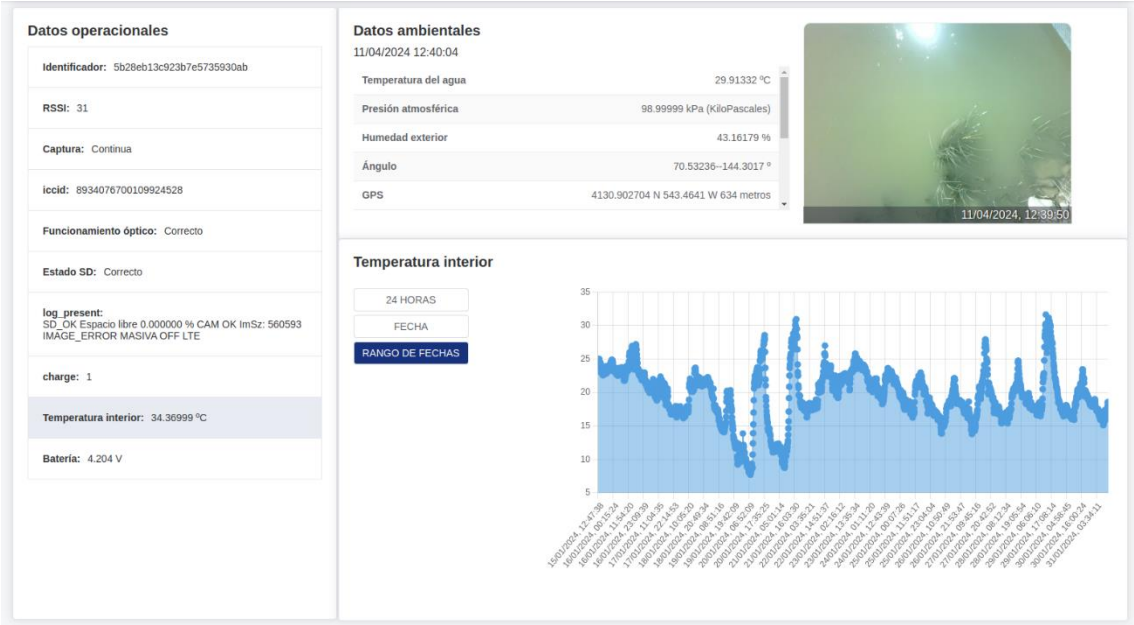

Ilustración 52. Captura de la plataforma para la visualización del dato de la temperatura interior.

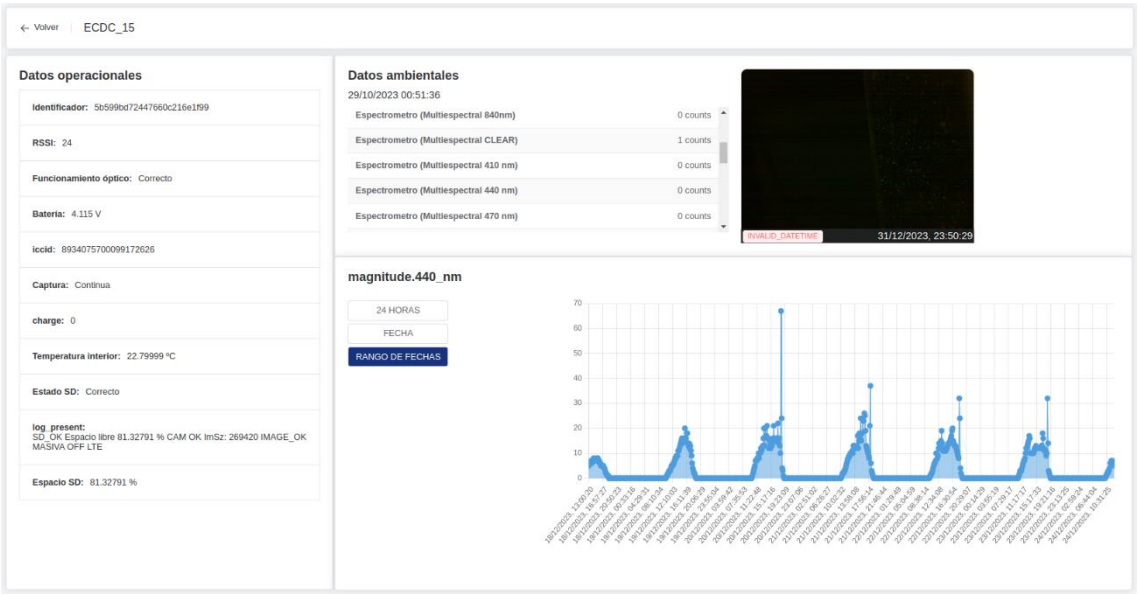

Ilustración 53. Captura de la plataforma para la visualización de uno de los canales del sensor AS7341.

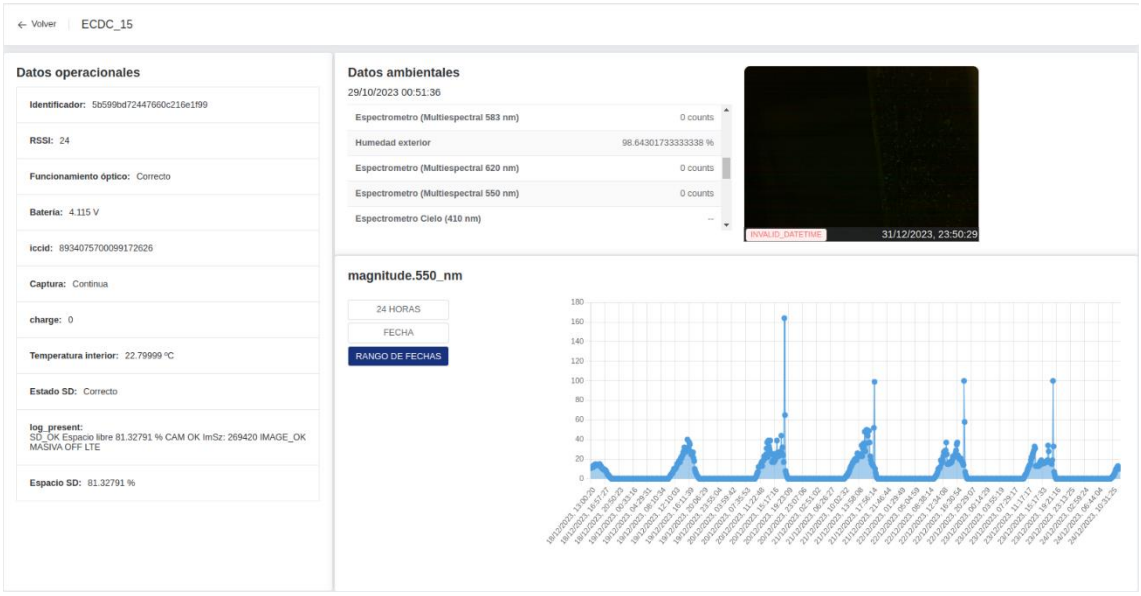

Ilustración 54. Captura de la plataforma para la visualización de uno de los canales del sensor AS7341.

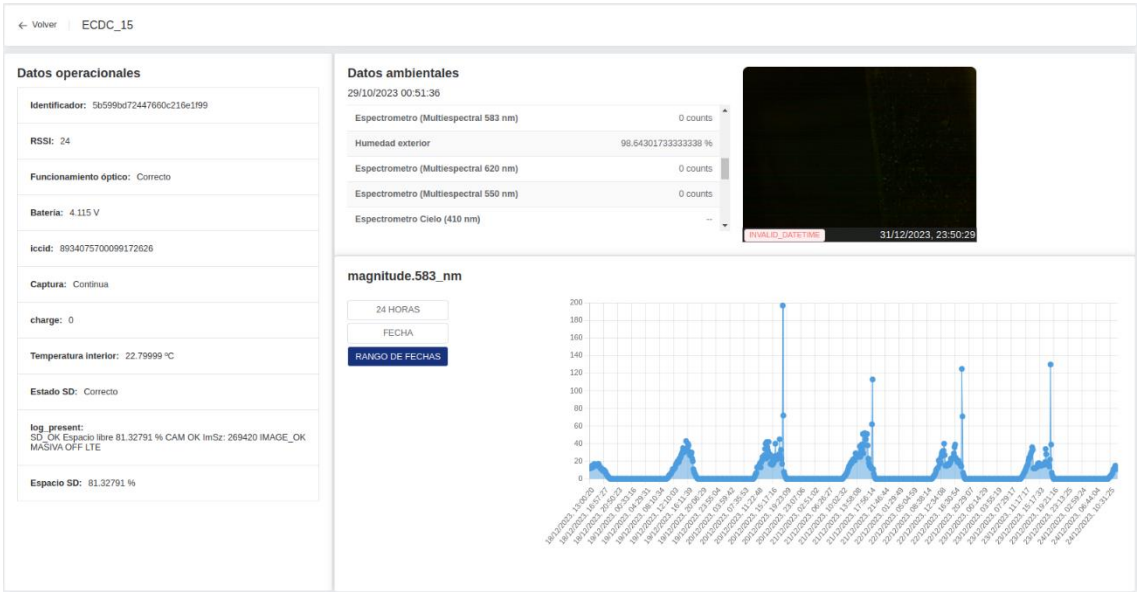

Ilustración 55. Captura de la plataforma para la visualización de uno de los canales del sensor AS7341.

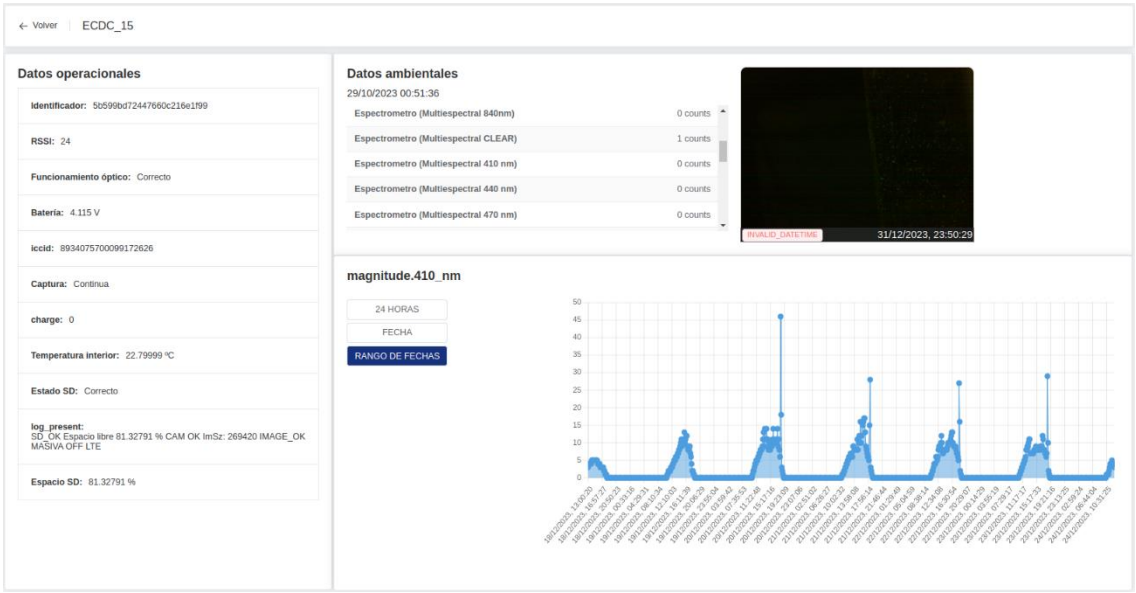

Ilustración 56. Captura de la plataforma para la visualización de uno de los canales del sensor AS7341.

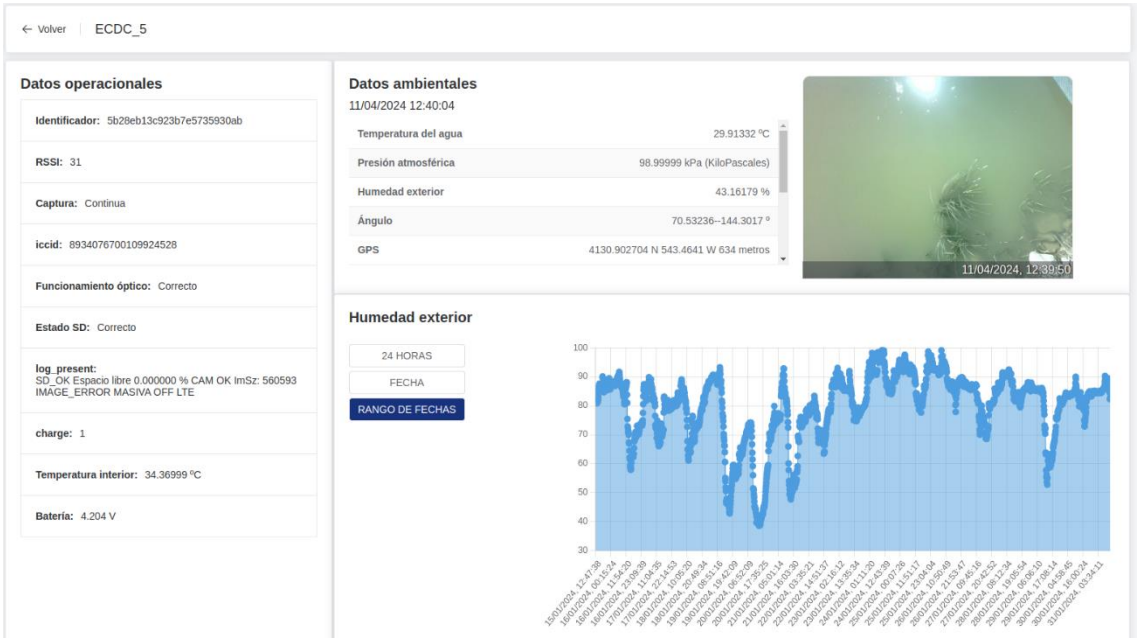

Ilustración 57. Captura de la plataforma para la visualización del dato de humedad exterior.

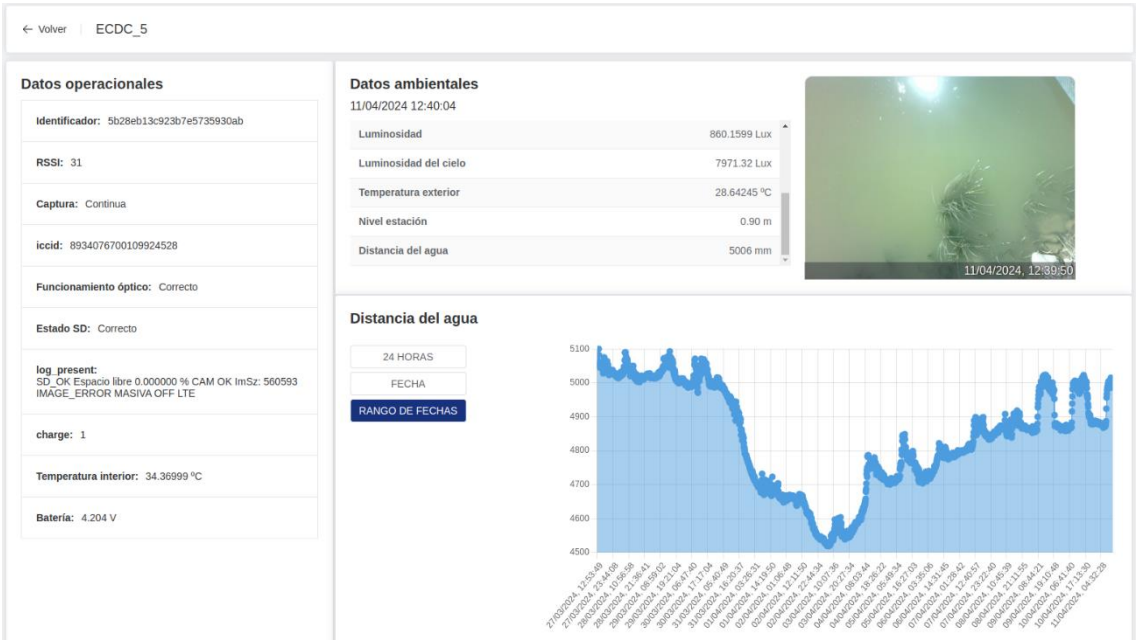

Ilustración 58. Captura de la plataforma para la visualización del dato de distancia al agua.

7. Conclusiones.

En conclusión, en este trabajo se ha llevado a cabo la tarea de diseñar una librería de funciones para los sensores AS7341, WSEN-HIDS, WSEN-PADS y URM06 de forma satisfactoria. Además, esta librería ha sido diseñada dentro de un proyecto con cierto grado de desarrollo. Por ello ha sido necesario diseñar estas funciones para que se puedan ejecutar dentro del ciclo de trabajo que tiene la estación sin que produzcan ningún tipo de conflicto en el mismo.

La implementación de estas funciones dentro del proyecto también se ha realizado de forma satisfactoria. Varias de las estaciones situadas en diferentes puntos de monitorización de la comunidad de Castilla y León poseen estas funciones y han presentado buenos resultados en términos de operatividad.

A la hora de realizar el diseño de esta librería han sido de gran ayuda mis conocimientos en materia de programación y en menor medida, áreas como la electrónica y microcontroladores.

Como habilidad clave a destacar a la hora de realizar esta tarea. Destacaría la capacidad para analizar detenidamente los “data sheet” de los sensores. Como se ha visto la mayoría de los sensores poseen librerías para facilitar el desarrollo del código que controle estos sensores, pero de igual forma es fundamental realizar un análisis pausado de los documentos y especificaciones que proporciona el fabricante por que ayudan a implementar y exprimir de forma óptima todas las funcionalidades que presenta el sensor.

También, me ha parecido clave el entorno de desarrollo que poseen los microcontroladores STM32. Permiten muchas facilidades a la hora de diseñar y desarrollar el código, permitiendo ahorrar una gran cantidad de horas de trabajo. Punto muy a tener en cuenta cuando estas diseñando un producto para una empresa privada que tiene que cumplir con ciertos requisitos antes de una fecha límite.

8. Bibliografía

[1] https://aquacorp.es/Technology

[2] https://www.st.com/en/microcontrollers-microprocessors/stm32f429zg.html

[3] https://partners.sigfox.com/search/products?or[businessBenefits][0]=monitor-environmental-conditions

[4] https://github.com/adafruit/Adafruit_AS7341

[5] https://github.com/WurthElektronik/SensorLibrariesArduino

[6] https://www.st.com/en/microcontrollers-microprocessors/stm32f429zg.html

[7] https://wiki.dfrobot.com/URM06-PULSE_Ultrasonic_SKU_SEN0151

[8] https://www.we-online.com/en/components/products/WSEN-PADS

[9] https://www.we-online.com/en/components/products/WSEN-HIDS

[10] https://ams.com/as7341

[11] Microcontrolador STM32 Programación y Desarrollo. Jesús María Pestano Herrera. Capitulo Programación I2C.