

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

2/12/2018

# Proyecto Final Estación meteorológica con conexión a internet

Complementos Sistemas Electrónicos Digitales Avanzados –  
MUIT

Several thin, curved, light blue lines that sweep upwards from the bottom left towards the center of the page.

Gallego Sánchez, Adrián - Constantin Boby, Nicusor  
UAH

## Tabla de Contenidos

<i>Índice de Figuras</i> .....	2
<i>Índice de Tablas</i> .....	3
<i>Introducción</i> .....	4
<i>Descripción del proyecto</i> .....	5
<i>Hardware</i> .....	5
<i>Sensores Analógicos</i> .....	5
<i>Sensores Digitales</i> .....	9
<i>Otros</i> .....	14
<i>Software utilizado</i> .....	18
<i>Bibliotecas</i> .....	18
<i>HTTP</i> .....	18
<i>GLCD</i> .....	19
<i>I2c_lpcxx</i> .....	19
<i>Código generado</i> .....	20
<i>init.c</i> .....	20
<i>Uart.c</i> .....	22
<i>DMA.c</i> .....	24
<i>PWM.c</i> .....	28
<i>DS1621.c</i> .....	29
<i>BMP180.c</i> .....	30
<i>Main.c</i> .....	32
<i>HTTP_CGI.c</i> .....	37
<i>Index.cgi</i> .....	39
<i>Web.inp</i> .....	39
<i>Resultados obtenidos</i> .....	40
<i>Página web</i> .....	40
<i>Display</i> .....	41
<i>UART</i> .....	42
<i>Conclusiones</i> .....	42

## Índice de Figuras

<i>Figura 1 - Esquema del proyecto. ....</i>	<i>4</i>
<i>Figura 2 - Pinout LM35. ....</i>	<i>6</i>
<i>Figura 3 - Modos de funcionamiento del LM35. ....</i>	<i>6</i>
<i>Figura 4 - Diseño propuesto para el circuito de filtrado de la alimentación de Vcc=+5V. ....</i>	<i>7</i>
<i>Figura 5 - Imagen del sensor HIH4000. ....</i>	<i>7</i>
<i>Figura 6 - Circuito de alimentación del sensor HIH400. ....</i>	<i>8</i>
<i>Figura 7 - Curva de comportamiento del sensor HIH4000. ....</i>	<i>8</i>
<i>Figura 8 - Imagen y pinout del sensor DS1621. ....</i>	<i>9</i>
<i>Figura 9 - Imagen del sensor digital BMP180. ....</i>	<i>11</i>
<i>Figura 10 - Circuito de alimentación del sensor BMP180. ....</i>	<i>12</i>
<i>Figura 11 - Imagen del micrófono amplificado MAX9812. ....</i>	<i>13</i>
<i>Figura 12 - Imagen del ventilador usado en el proyecto. ....</i>	<i>14</i>
<i>Figura 13 - Imágenes del anemómetro implementado. ....</i>	<i>15</i>
<i>Figura 14 - Esquema de alimentación del CNY70. ....</i>	<i>16</i>
<i>Figura 15 - Pinout e imagen del LCD HY28B. ....</i>	<i>17</i>
<i>Figura 16 - Definición de la función init_GPIO(). ....</i>	<i>20</i>
<i>Figura 17 - Definición de la función init_TIMER0. ....</i>	<i>20</i>
<i>Figura 18 - Definición de la función init_ADC_sensores. ....</i>	<i>21</i>
<i>Figura 19 - Definición de la función init_ADC_grabar. ....</i>	<i>21</i>
<i>Figura 20 - Definición de la función init_Timer1. ....</i>	<i>21</i>
<i>Figura 21 - Definición de la función init_timer3. ....</i>	<i>22</i>
<i>Figura 22 - Definición de la función uart0_init. ....</i>	<i>22</i>
<i>Figura 23 - Definición de la función Uart0_set_baudrate. ....</i>	<i>22</i>
<i>Figura 24 - Definición de la función tx_cadena_UART0. ....</i>	<i>23</i>
<i>Figura 25 - Definición de la función UART0_IRQHandler. ....</i>	<i>23</i>
<i>Figura 26 - Definición de la función init_DAM_ADC. ....</i>	<i>24</i>
<i>Figura 27 - Definición de la función init_DMA_DAC. ....</i>	<i>25</i>
<i>Figura 28 - Definición de la función DMA_IRQHandler. ....</i>	<i>25</i>
<i>Figura 29 - Definición de la función TIMER1_IRQHandler. ....</i>	<i>26</i>
<i>Figura 30 - Definición de la función rec_ADC. ....</i>	<i>26</i>
<i>Figura 31 - Definición de la función rec_DMA. ....</i>	<i>26</i>
<i>Figura 32 - Definición de la función play. ....</i>	<i>26</i>
<i>Figura 33 - Definición de la función init_PWM. ....</i>	<i>28</i>
<i>Figura 34 - Definición de la función set_ciclo_trabajo_PWM. ....</i>	<i>28</i>
<i>Figura 35 - Definición de la función config_DS1621. ....</i>	<i>29</i>
<i>Figura 36 - Definición de la función leer_Ds1621. ....</i>	<i>29</i>
<i>Figura 37 - Definición de la función read_uncompensated_temp. ....</i>	<i>30</i>
<i>Figura 38 - Definición de la función calculate_temp. ....</i>	<i>30</i>
<i>Figura 39 - Definición de la función read_uncompensated_press. ....</i>	<i>31</i>
<i>Figura 40 - Definición de la función calculate_press. ....</i>	<i>31</i>
<i>Figura 41 - Definición de la función calculate_altitude. ....</i>	<i>31</i>
<i>Figura 42 - Definición de la función timer_pool(). ....</i>	<i>32</i>
<i>Figura 43 - Definición de la función DHCP_check. ....</i>	<i>32</i>
<i>Figura 44 - Definición de la función init. ....</i>	<i>33</i>
<i>Figura 45 - Definición de la función TIMER0_IRQHnadler. ....</i>	<i>34</i>
<i>Figura 46 - Definición de la función ADC_IRQHandler. ....</i>	<i>34</i>
<i>Figura 47 - Definición de la función init_display. ....</i>	<i>35</i>
<i>Figura 48 - Definición de la función upd_display. ....</i>	<i>35</i>

<i>Figura 49- Definición de la función main.</i>	36
<i>Figura 50 - Definición de la función EINTX_IRQHandler.</i>	36
<i>Figura 51 - Definición de la función cgi_func.</i>	37
<i>Figura 52 - Definición de la función cgi_process_var.</i>	38
<i>Figura 53 - Información de comando para archivns .cgi.</i>	39
<i>Figura 54 - Contenido del archivo web.inp.</i>	39
<i>Figura 55 - Aspecto de la página web.</i>	40
<i>Figura 56 - Aspecto del display LCD.</i>	41
<i>Figura 57 - Representación de datos por la UART.</i>	42

## Índice de Tablas

<i>Tabla 1 - Archivos de la librería HTTP del Keil.</i>	18
<i>Tabla 2 - Archivos de la biblioteca GLD.</i>	19
<i>Tabla 3 - Funciones del archivo I2c_lpcxx.</i>	19

## Introducción

La propuesta de proyecto consiste en diseñar un sistema empujado basado en el microcontrolador LPC1768 (Cortex-M3). El objetivo es implementar una estación meteorológica que ofrezca la posibilidad de ser monitorizada de forma remota. El sistema contará con los siguientes elementos:

- Sensor analógico de temperatura LM35.
- Termómetro-termostato digital DS1621.
- Sensor analógico de humedad HIH4000.
- Sensor digital de presión BMP180.
- Micrófono.
- Anemómetro.
- Ventilador.
- Altavoz.
- Módulo de visualización (HY28B).

El sistema mostrará sobre un display los datos obtenidos de los diferentes sensores. Además, también podrán ser monitorizados mediante un entorno WEB o una interfaz serie asíncrona.

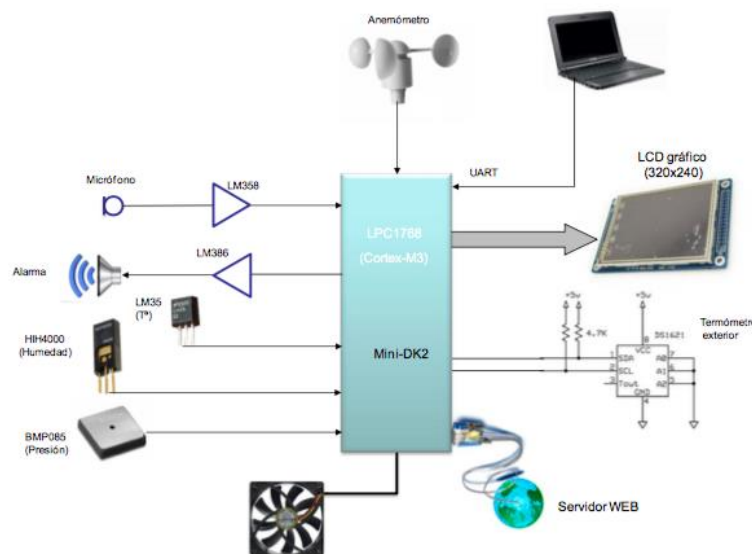


Figura 1 - Esquema del proyecto.

Como se puede observar, el objetivo perseguido es el de implementar todos los dispositivos antes mencionados, con sus respectivos circuitos de acondicionamiento si fueren necesarios, en un solo microcontrolador de manera que todos ellos trabajen a la par y se consiga un producto final similar a cualquier estación meteorológica de bajo coste que se pueda encontrar en el mercado.

## Descripción del proyecto

En los siguientes apartados se procede a describir los elementos que componen el proyecto, bien elementos hardware, bien código desarrollado.

### Hardware

---

En este apartado se especificarán las características de los distintos sensores y dispositivos hardware empleados, así como los sistemas de acondicionamiento empleados para adquirir las señales. Por otro lado, se distinguirán varias secciones:

- **Sensores analógicos:** En ella se describen las características del comportamiento para cada uno de los sensores de carácter analógico implementados.
- **Sensores digitales:** En ella se describen las características del comportamiento para cada uno de los sensores de carácter digital implementados.
- **Otros:** En ella se describe el comportamiento y las características del resto de dispositivos/módulos implementados.

### Sensores Analógicos

#### *Sensor LM35<sup>1</sup>*

---

El *LM35* es un dispositivo de circuito integrado que mide temperatura con una salida en tensión proporcional a la temperatura en grados centígrados. La ventaja de los dispositivos *LM35* sobre los sensores de temperatura calibrados en Kelvin es que el usuario no necesita restar a la salida una constante en tensión para obtener un escalado apropiado de los grados centígrados. En adición, los dispositivos *LM35* no necesitan un calibrado externo para ofrecer una precisión de hasta  $\frac{1}{4}$  de grado a temperatura ambiente y hasta  $\frac{3}{4}$  de grado cuando mide temperaturas en el **rango de -55°C a 150°C**. La salida de baja impedancia, la salida lineal y la calibración inherente hacen que la lectura y circuitería de control resulten especialmente sencillas. Se ha empleado el formato de sensor con empaquetado "LP" de 3 pines "TO-92".

#### Características principales:

- Calibrado directamente en **grados centígrados** (Celsius).
- Factor lineal de **+10 mV/°C**.
- Precisión **de 0.5°C** a temperatura ambiente (25°C).
- Funciona con una **alimentación desde 4 a 30 V**.

---

<sup>1</sup> <http://www.ti.com/lit/ds/symlink/lm35.pdf>

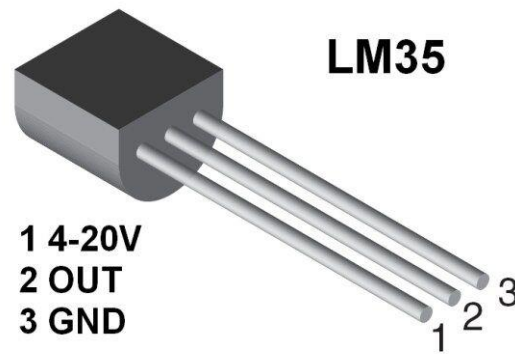
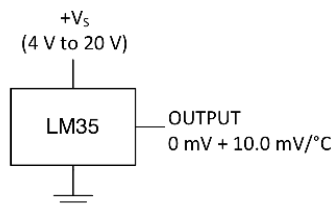


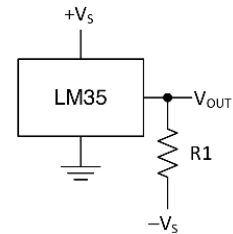
Figura 2 - Pinout LM35.

Este sensor permite implementar varios modos de funcionamiento dependiendo del rango dinámico que sea necesario implementar. A continuación, en la siguiente figura, se muestran los esquemas eléctricos de cada uno de dichos modos de funcionamiento:

**Basic Centigrade Temperature Sensor  
(2°C to 150°C)**



**Full-Range Centigrade Temperature Sensor**



Choose  $R_1 = -V_s / 50 \mu\text{A}$   
 $V_{OUT} = 1500 \text{ mV}$  at  $150^\circ\text{C}$   
 $V_{OUT} = 250 \text{ mV}$  at  $25^\circ\text{C}$   
 $V_{OUT} = -550 \text{ mV}$  at  $-55^\circ\text{C}$

Figura 3 - Modos de funcionamiento del LM35.

Puesto que se trata del sensor que va a realizar medidas sobre el exterior del encapsulado del sistema en desarrollo y dado que el rango dinámico de temperaturas puede obtener valores negativos, **se ha decidido implementar el modo "Full-Range" con una resistencia  $R_1$  de valor  $100\text{k}\Omega$ . Se alimenta el sensor con  $V_{cc}=+5\text{v}$ , extraídos desde la tarjeta de desarrollo.**

Para evitar posibles ruidos derivados de la alimentación desde la tarjeta, se propone el siguiente circuito de filtrado para todos aquellos dispositivos que se alimenten con  $V_{cc}=+5\text{v}$ :

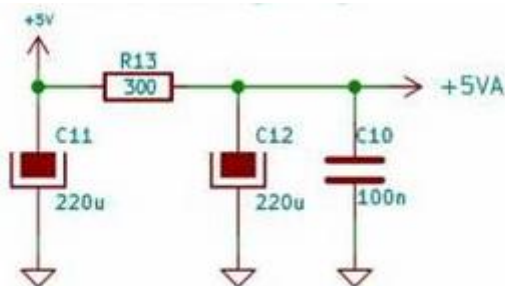


Figura 4 - Diseño propuesto para el circuito de filtrado de la alimentación de  $V_{cc}=+5V$  (realizado en Eagle<sup>2</sup>).

### Sensor HIH4000<sup>3</sup>

Este tipo de sensores de humedad, en concreto la serie HIH-4000, están específicamente diseñados para usuarios que empleen un alto volumen de **OEM** (Original Equipment Manufacturer). La gran ventaja de este sensor es que la salida casi lineal en tensión permite la conexión directa con un microcontrolador u otro dispositivo, lo cual es ideal para este proyecto. Con un **consumo de corriente de 200  $\mu A$** , típicamente, la serie HIH-4000 es ideal para sistemas de bajo consumo.

#### Características principales:

- Encapsulado de plástico termoendurecido.
- Salida en tensión casi lineal frente a % de humedad relativa.
- Diseño de bajo consumo.
- Tiempo rápido de respuesta.
- Precisión mejorada.
- Calibración de fábrica.

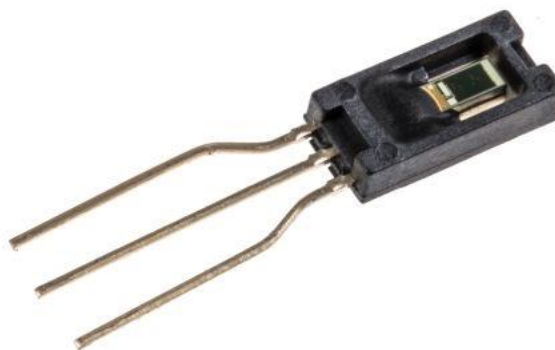


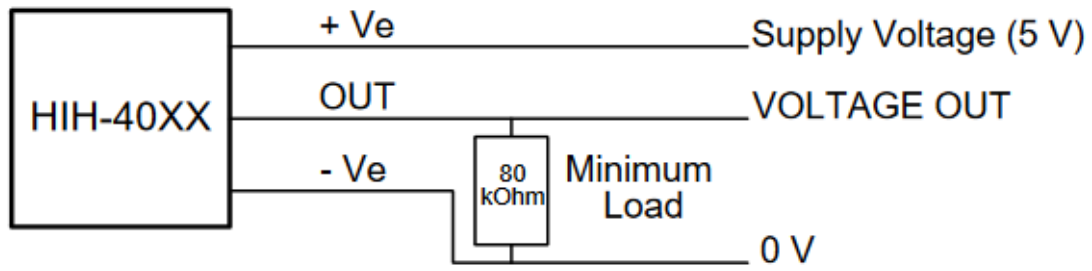
Figura 5 - Imagen del sensor HIH4000.

Este sensor permitirá realizar las mediciones de humedad relativa en el ambiente referidas al exterior del sistema generado. Para conectar este sensor a la entrada pertinente del microcontrolador (más adelante se detallará que dicha entrada es del ADC y cómo se han adquirido los datos), ha sido necesario implementar el siguiente circuito tomado del datasheet:

<sup>2</sup> <https://www.autodesk.com/products/eagle/overview>

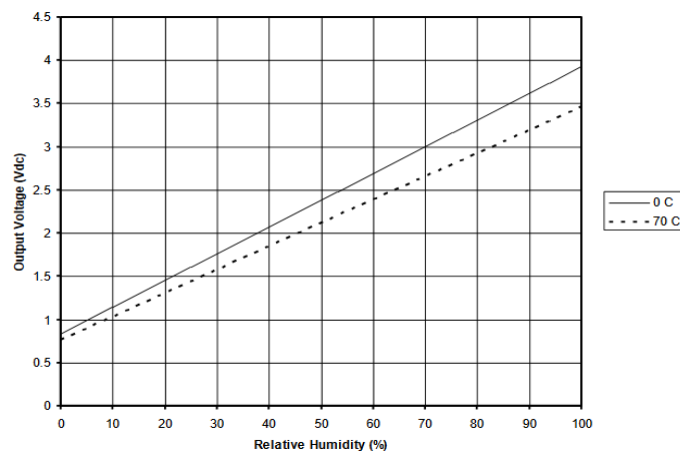
<sup>3</sup> <https://sensing.honeywell.com/honeywell-sensing-hih4000-series-product-sheet-009017-5-en.pdf>





**Figura 6 - Circuito de alimentación del sensor HIH400.**

Donde la resistencia indicada, en este caso, se ha implementado con un valor de 100KΩ. Finalmente, cabe destacar que a la hora de calcular la humedad relativa del sensor, será necesario tener en cuenta el offset reflejado en el siguiente diagrama:



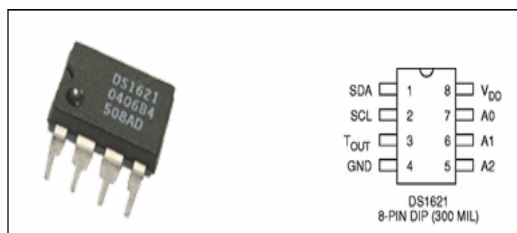
**Figura 7 - Curva de comportamiento del sensor HIH4000.**

## Sensores Digitales

### *Sensor DS1621<sup>4</sup>*

Se trata de un **termómetro-termostato digital** que proporciona lecturas de **temperatura de 9 bit**. Además, contiene una salida de alarma  $T_{out}$  que se activa cuando la temperatura del dispositivo supera el umbral definido por el usuario (TH), este tipo de comportamiento es el definido por el termostato (no se emplea en este proyecto). La salida permanece activa hasta que la temperatura se sitúa por debajo de un umbral (TL), también definido por el usuario.

Los ajustes definidos por el usuario se almacenan en memoria no volátil (EEPROM) por lo que puede ser programado antes de ser introducido en un sistema. Los ajustes y lecturas de temperatura se comunican hacia/desde el DS1621 mediante un interfaz serie de dos hilos (I2C). Se empleará este sensor para llevar a cabo la monitorización de la temperatura en el interior del posible encapsulado del sistema, de manera que sea posible activar dispositivos de enfriamiento en un cierto umbral evitando así el sobrecalentamiento del sistema.



*Figura 8 - Imagen y pinout del sensor DS1621.*

#### Características principales:

- El DS1621 es capaz de medir temperaturas en el **rango de -55°C a 125°C** con **incrementos de 0.5°C**.
- La configuración de termostato es definida por el usuario y guardada en memoria no volátil.
- Rango de **alimentación de 2.7 a 5.5 V**.
- La **comunicación se establece mediante I2C**.
- No requiere de componentes externos para medir la temperatura.

#### Comandos:

- **Leer temperatura [AAh]:** Lee el resultado de la última conversión de temperatura. EL DS1621 envía 2 bytes
- **Acceso a TH [A1h]:** Si el valor campo R/W es '0' se escribe en el registro TH (Alta Temperatura). Los dos siguientes bytes que se escriban/envíen al DS1621 configurarán el valor del umbral superior de temperatura para la alarma. Si el valor es '1' entonces se lee el valor del registro TH
- **Acceso a TL [A2h]:** Si el valor campo R/W es '0' se escribe en el registro TL (Baja Temperatura). Los dos siguientes bytes que se escriban/envíen al DS1621 configurarán el valor del umbral inferior de temperatura para la alarma. Si el valor es '1' entonces se lee el valor del registro TL

<sup>4</sup> <https://datasheets.maximintegrated.com/en/ds/DS1621.pdf>

- **Acceso a Config [ACh]:** Si el valor campo R/W es '0' se escribe en el registro de configuración. El siguiente byte se escribirá en el registro. Si el valor es '1' se lee el valor del registro de configuración
- **Leer Contador [A8h]:** Este comando lee el valor *Cout\_Remain*. El comando es válido únicamente si R/W es '1'
- **Leer Slope (pendiente)[A9h]:** Este comando lee el valor *Count\_Per\_C*. El comando es válido únicamente si R/W es '1'
- **Iniciar conversión T [EEh]:** Inicia la conversión de temperatura. En el modo disparo una vez que se realice la conversión el DS1621 quedará ocioso. En el modo continuo, el DS1621 el comando iniciará conversiones continuamente
- **Parar la conversión T [22h]:** Interrumpe la conversión de temperatura. Se utiliza para detener un DS1621 en modo conversión continua. Tras enviar este comando, el sensor finalizará la conversión en curso y se mantendrá ocioso hasta nueva orden

El sensor permite definir los tres últimos bits de la dirección (7bits) asignada a dicho dispositivo, siendo los cuatro bits de mayor peso de valor "1001". Los 3 bits restantes se han llevado a masa, es decir que los pines  $A_0=A_1=A_2=0$  y, por tanto, **la dirección asignada a este dispositivo ha sido la 0x48.**

Por otro lado, **se ha alimentado con  $V_{cc}=+5v$**  y se han implementado las sendas resistencias de pull-up necesarias para que el protocolo I2C sea plenamente funcional. **Dichas resistencias tienen, ambas dos, un valor de 2.2K $\Omega$ .**

Cabe destacar que fue necesario emplear un adaptador *SOIC-DIP* de 8 pines para poder emplear este sensor en el proyecto y testearlo en la *proto-board* ya que se compró, por error, el sensor con formato *SOIC*.

## Sensor BMP180<sup>5</sup>

Se trata de un sensor de presión diseñado para ser conectado directamente a un microcontrolador a través de un bus I2C. Su electrónica de ultra-bajo consumo lo hacen óptimo para teléfonos móviles, PDAs, navegadores GPS, equipamiento de campo o, para este caso, para el proyecto referido.

El BMP180 está formado por un sensor piezo-resistivo, un conversor analógico digital y una unidad de control con E<sup>2</sup>PROM e interfaz serie I2C. Lo cual, permite al dispositivo entregar el valor de presión y temperatura sin calibrar. Para obtener los datos de dichas magnitudes de manera representativa, en otras palabras, obtener las magnitudes calibradas y reales, el sensor incorpora una memoria E<sup>2</sup>PROM que contiene 176 bits de datos de calibración. Dicha memoria E<sup>2</sup>PROM, está dividida en 11 palabras de 16 bits cada una, cada palabra conforma un parámetro o coeficiente de calibración. Estos coeficientes son individuales para cada sensor, por lo tanto, el *máster* ha de leer los parámetros antes de la primera conversión.

**Características principales** (fabricante *Adafruit*):

- Rango de **alimentación de 1.8 a 3.6 V.**
- Rango de medida de **presión desde 300 hasta 1100 hPa.**
- **Bajo consumo:** 5  $\mu$ A por muestra/seg en modo estándar.
- **Precisión:** 0.03 hPa en modo de alta precisión.



*Figura 9 - Imagen del sensor digital BMP180.*

El objetivo principal de este sensor es el de permitir al sistema realizar medidas sobre la presión absoluta, con respecto al nivel del mar, y reflejarlas en los distintos sistemas de visualización. A su vez, y de manera adicional, realizará medidas sobre la temperatura en el exterior del sistema (esto permitirá realizar comparativas entre los resultados obtenidos por la medida analógica del *LM35* y este) y sobre la altitud en la que esté ubicado el sistema.

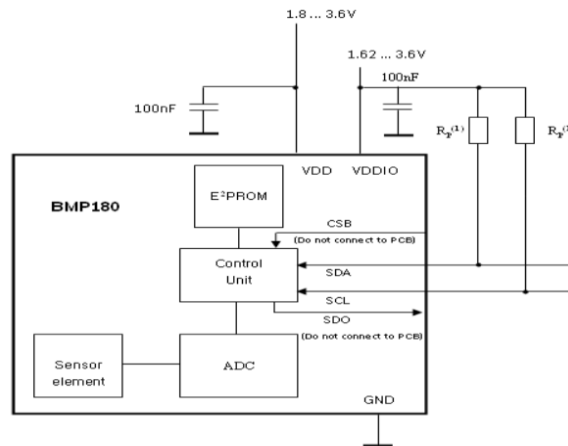
En contraste con el sensor *DS1621*, este no permite la configuración de la dirección de dispositivo esclavo y, por tanto, esta tiene un **valor fijo de 0x77**. Cabe destacar que este sensor en concreto incorpora un sistema para la comprobación del correcto funcionamiento de las comunicaciones maestro-esclavo, este sistema es algo tan simple como el hecho de incorporar un registro de solo lectura (0xD0) con un valor fijo conocido como **ID (0x55)**, de manera que el desarrollador, para comprobar si existe una comunicación con el dispositivo tan solo tiene que leer dicho registro y comprobar que el valor devuelto es un 0x55. Asimismo, contempla el uso de varios niveles de precisión, permitiendo el ajuste de estos (nºmuestras, ruido, tiempo de conversión...) mediante un parámetro configurable en el registro de control, en este proyecto se ha empleado el **modo estándar**, esto es:

- **Oss = 1**
- **Número de muestras = 2**

<sup>5</sup> <https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf>

- **Tiempo de conversión** = 7.5ms
- **Corriente media** = 5 $\mu$ A
- **Ruido RMS en presión (hPa)**: 0.05

Para terminar, pasa a describirse el circuito implementado para el correcto funcionamiento del protocolo de comunicación entre el maestro y el esclavo:



**Figura 10 - Circuito de alimentación del sensor BMP180.**

Como se puede observar en la figura anterior, ha sido necesario implementar un condensador de 100nF y dos resistencias de pull-up, en este caso, ambas de valor 2.2k $\Omega$ .

## Micrófono<sup>6</sup>

Con el objetivo de permitir al usuario grabar un mensaje de alarma que se emplee para avisar de manera sonora que se ha superado el umbral de la temperatura de trabajo o para futuras aplicaciones que pudieran incluir el trato con sonido, se ha decidido hacer uso del micrófono *MAX9812L*. Este modelo, junto al *MAX9813*, es un micrófono de salida única con una **ganancia fija de 20dB** y que introduce poco ruido, por ello, es ideal para el uso en PDAs, teléfonos móviles, etc. El nombre que recibe el micrófono hace referencia al amplificador operacional que lleva integrado en la placa, el *MAX9812*, el cual está en formato *SC70*, permite mantener el **ancho de banda del micrófono a 500kHz** y está optimizado para trabajar con una **tensión de entrada de 3.3v**. En la siguiente imagen es posible apreciar los distintos componentes mencionados:



*Figura 11 - Imagen del micrófono amplificado MAX9812.*

Cabe destacar que, a pesar de tener una ganancia fija de 20dB, esta es, en la mayoría de los casos, insuficiente. Por este motivo, ha sido necesario implementar un circuito de amplificación de la señal que extrae este micrófono, para ello se han empleado los siguientes componentes:

- **Amplificador Operacional SN741:** Configurado en modo no inversor con una ganancia de 101.
- **Resistencia R1:** 10K $\Omega$ .
- **Resistencia R2:** 1M $\Omega$ .

---

<sup>6</sup> <https://datasheets.maximintegrated.com/en/ds/MAX9812-MAX9813L.pdf>

## Otros

### Altavoz

En relación con lo mencionado en el apartado anterior sobre la señal de aviso/alarma, ha sido necesario implementar un dispositivo que permitiera al usuario ser consciente de que la señal de alarma se ha disparado y para ello, se ha usado un altavoz dinámico de bobina móvil con una **impedancia de entrada de, aproximadamente,  $8\Omega$** .

Para permitir que la señal que reproduce el altavoz sea nítida y tenga la amplitud requerida, se ha requerido el uso de una etapa de amplificación. Dicha etapa lo constituye el **Módulo de amplificación de sonido LM386**, el cual se compone del famoso amplificador operacional LM386, un potenciómetro para regular la ganancia y algunos condensadores para acondicionar y filtrar la señal, por lo tanto, es idóneo para los requisitos de la aplicación.

### Ventilador

El sistema diseñado cuenta con un ventilador que se encarga de refrigerar el sistema cuando la temperatura medida por el sensor de temperatura interior supera un cierto umbral. Dicho umbral también permite regular la velocidad y el ciclo de trabajo de este ventilador mediante una señal PWM de ciclo regulable como se especificará más adelante<sup>7</sup>.



*Figura 12 - Imagen del ventilador usado en el proyecto.*

---

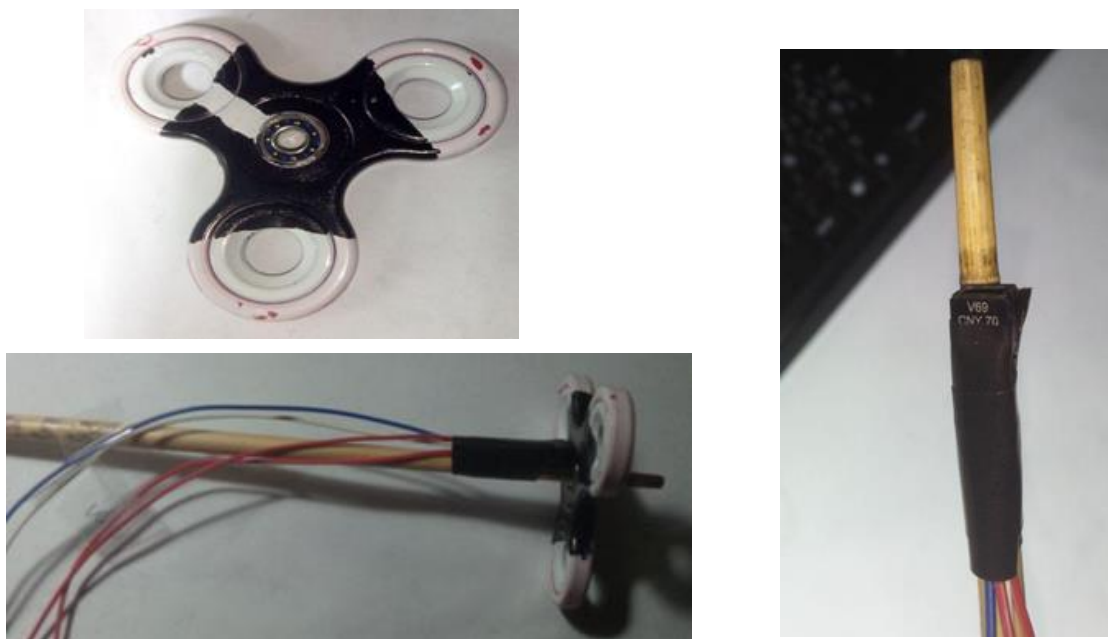
<sup>7</sup> Véase el apartado **PWM.c**.

### *Anemómetro*

Como toda estación meteorológica, el sistema diseñado también implementa un dispositivo que permite tomar medidas acerca de la velocidad del viento. En este caso se ha utilizado un “pseudo-anemómetro”, el cual se compone de:

- **Un Spinner.**
- **Un sensor CNY70<sup>8</sup>.**
- **Elemento de sujeción.**

El objetivo es el de emular el comportamiento de un anemómetro real, midiendo la frecuencia con que el sensor refleja una banda blanca pintada en el spinner. A continuación, se adjuntan imágenes del “diseño”:



*Figura 13 - Imágenes del anemómetro implementado.*

Como se puede observar, ha sido necesario pintar el Spinner con un fondo negro y una banda blanca para que el fotodiodo pueda trabajar de manera correcta. Dadas las características del CNY70, ha sido necesario acercar lo máximo posible el sensor al spinner (el datasheet recomienda una distancia de 0.5mm).

Para terminar, puesto que la señal obtenida por el CNY70 no está conformada para generar pulsos de 0-5v, es necesario tratar la señal con una etapa de acondicionamiento para adaptarla a las necesidades del microcontrolador. Con este objetivo, se ha empleado como base el siguiente circuito:

<sup>8</sup> <http://www.vishay.com/docs/83751/cny70.pdf>



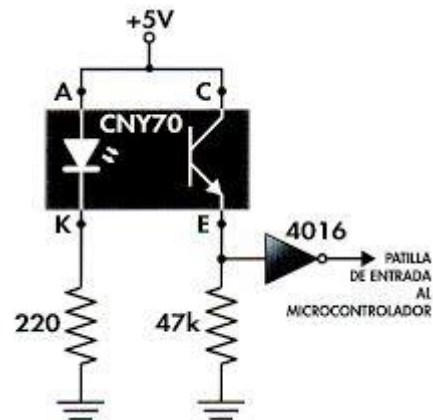


Figura 14 - Esquema de alimentación del CNY70.

A dicho circuito se le han realizado algunas modificaciones:

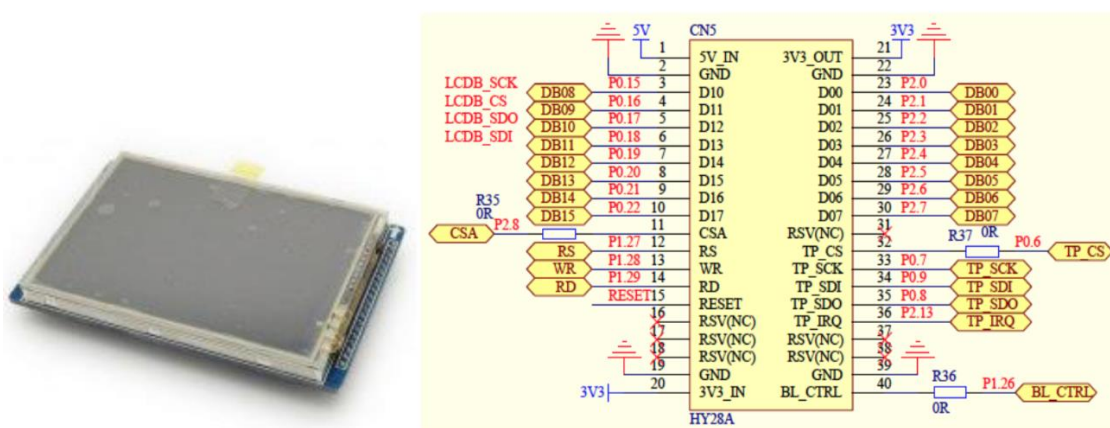
- Se ha **eliminado la resistencia de 47kΩ**.
- El **dispositivo 4016 se ha sustituido por un inversor SN74HC14N** que permite conformar los pulsos.
- Entre la salida de dicho inversor y masa **se ha colocado un led para conocer si se están generando los pulsos a medir**.

### *Módulo de visualización (HY28B)<sup>9</sup>*

El módulo de visualización consta de un display TFT de 2.8" a color táctil y con una resolución de 320x240 píxeles. Utiliza el driver ILI9325, un driver integrado que proporciona hasta 262.144 colores. Además, cuenta con una RAM de 172.800 bytes de memoria gráfica. El display está implementado en un PCB que incorpora los circuitos de la interfaz táctil XPT2046 y de la retroiluminación del panel.

El driver ILI9325 permite 3 modos de funcionamiento de alta velocidad: 8 bits en paralelo, 16 bits en paralelo o mediante SPI.

El controlador XPT2046 es un controlador de pantalla táctil resistiva que incorpora un conversor A/D de 12 bit y 125 KHz de muestreo. Es capaz de detectar la presión en la pantalla mediante dos conversiones A/D. Además de la posición, también mide la presión de la pulsación.



**Figura 15 - Pinout e imagen del LCD HY28B.**

<sup>9</sup> <http://www.i1sys.com/products/psoc/HY28B-Adapter-Datasheet.pdf>

## Software utilizado

En los siguientes apartados se describen las bibliotecas más destacables que se han empleado para desarrollar el proyecto, así como la función que desempeñan dentro de este.

### *Bibliotecas*

---

#### *HTTP*

Esta biblioteca es proporcionada por KEIL y nos permite generar páginas web dinámicas que es uno de los objetivos del proyecto. Los ficheros más destacables que incluye son:

Nombre	Descripción
<b>TCP_CM3.lib</b>	Proporciona las funciones de comunicaciones de KEIL RL-TCPnet.
<b>EMAC_LPC17xx_LAN8720.c</b>	Implementa la interfaz entre las funciones de la biblioteca de comunicaciones con el hardware, tanto del subsistema hardware del microcontrolador como con el chip externo que implementa el nivel físico.
<b>Net_Config.c</b>	Fichero a través del cual se configura la biblioteca de funciones de la comunicación TCP/IP. Entre otros, se puede configurar la dirección IP, la máscara de red, el servidor DNS, el Gateway o la dirección MAC.
<b>Net_Debug.c</b>	Permite depurar las comunicaciones TCP/IP.
<b>TCPD_CM3.lib</b>	Biblioteca de funciones de comunicaciones de Keil RL-TCPnet que permiten depuración. Se debe mencionar que no es compatible con la librería TCP_CM3.lib

Tabla 1 - Archivos de la librería HTTP del Keil.

## GLCD

Esta biblioteca se encarga de gestionar el display y su configuración. Las librerías principales son:

Nombre	Descripción
<b>GLCD.c</b>	Contiene las funciones de configuración y acceso al display.
<b>AsciiLib.c</b>	Contiene la tabla de codificaciones de los píxeles gráficos que corresponde a cada carácter ASCII.

Tabla 2 - Archivos de la biblioteca GLD.

## I2c\_Ipcxx

Esta biblioteca se encarga de implementar las funciones necesarias para simular el funcionamiento de i<sup>2</sup>c mediante los pines P0.0 y P0.1 que simulan SDA y SCL respectivamente. Las principales funciones son:

Nombre	Parámetros	Tipo	Descripción
<b>I2CSendAddr</b>	addr, rw	uchar, uchar	Genera la condición de START necesaria para comenzar las comunicaciones en el protocolo I2C y establece la comunicación con el dispositivo esclavo referenciados por la dirección de este.
<b>I2CGetByte</b>	ack	uchar	Se encarga de obtener la respuesta del esclavo ante una consulta del maestro y de generar el respectivo ACK-NACK por parte del maestro.
<b>I2CSendByte</b>	byte	uchar	Se encarga de enviar las órdenes desde el maestro al esclavo.
<b>I2CSendStop</b>	-	-	Envía la condición de Stop dada en el protocolo I2C.
<b>I2CDelay</b>	-	-	Genera un delay referido al retardo necesario entre el bus SDA y SCL para comenzar/acabar la comunicación.
<b>Pulso_SCL</b>	-	-	Genera los pulsos que se enviarán por el bus SCL para mantener una comunicación sincronizada.

Tabla 3 - Funciones del archivo I2c\_Ipcxx.

## Código generado

El código generado se ha dividido en varios archivos con extensión “.c”, en general, separados en función del dispositivo al que hacen referencia y cada uno de ellos con un “header” asociado con las declaraciones de variables, constantes y funciones necesarias. A continuación, se describe cada uno de ellos.

### init.c

Este archivo es uno de los más importantes del proyecto ya que se encarga de realizar la configuración de los *timers*, del ADC y de inicializar cada uno de los pines empleados de manera correcta. Las funciones que se implementan son:

- **Init\_GPIO():** Se encarga de inicializar los pines para los sensores, micrófono, pulsadores, anemómetro, ventilador, altavoz, PWM y para los GPIO que se utilizan como SDA y SCL para simular I2C. Como se puede observar en la siguiente figura se trata de una función sin argumentos de entrada y sin retorno.

```

3  /*----- Inicialización de los pines para los sensores, pulsadores, micrófono, altavoces, ventilador -----*/
4
5  void init_GPIO(void)
6  {
7      // Sensores analógicos y micrófono
8      LPC_PINCON->PINSEL1 |= (1<<18); // AD0.2 PIN0.25 (LM35)
9      LPC_PINCON->PINSEL3 |= (3<<28)|(3<<30); // AD0.4 PIN1.30 (HIH) | AD0.5 PIN1.31 (Micrófono)
10
11     LPC_PINCON->PINMODE1 |= (2<<18); // Se deshabilita el pullup/down
12     LPC_PINCON->PINMODE3 |= (2<<28)|(2<<30);
13
14     // Pulsadores
15     LPC_PINCON->PINSEL4 |= (1<<22)|(1<<24); // EINT1 Key1 (Pulsador-grabar) PIN2.11 | EINT2 Key2 (Pulsador-reproducir) PIN2.12
16     LPC_PINCON->PINMODE4 |= (2<<22)|(2<<24); // ¿¿¿Se deshabilita el pullup/down???
17
18     // Anemómetro
19     LPC_PINCON->PINSEL1 |= (3<<16); // CAP3.1 PIN0.24 (Anemómetro)
20     LPC_PINCON->PINMODE1 |= (2<<18); // Se deshabilita el pullup/down
21
22     // Ventilador
23     LPC_PINCON->PINSEL3 |= (2<<10); // PWM1.3 PIN1.21 (Ventilador)
24     LPC_PINCON->PINMODE3 |= (2<<10); // Se deshabilita el pullup/down
25
26     // Altavoz
27     LPC_PINCON->PINSEL1 |= (2<<20); // AOUT PIN0.26 (DAC output - Altavoz)
28     LPC_PINCON->PINMODE1 |= (2<<20); // Deshabilita pullup/pulldown
29
30     // PWM
31     LPC_PINCON->PINSEL3 |= (2<<4); // PWM1.1 P1.18
32     LPC_PINCON->PINMODE3 |= (2<<4); // Deshabilita pullup/pulldown
33
34     // I2C
35     LPC_PINCON->PINMODE0 |= (2<<0)|(2<<2); // Deshabilita pullup/pulldown de P0.0 y P0.1 para simular
36     // SDA y SCL
37 }

```

Figura 16 - Definición de la función init\_GPIO().

- **Init\_TIMER0():** Configura el TIMER0 para que interrumpa cada segundo. Su función de interrupción se utilizará para gestionar diferentes tareas del sistema, entre ellas, mandar los datos adquiridos por los diferentes sensores a través de la UART, el LCD y la Web, actualizar los valores de algunos de estos sensores. Como se puede observar en la siguiente figura se trata de una función sin argumentos de entrada y sin retorno.

```

43 void init_TIMER0(void)
44 {
45     LPC_SC->PCONP |= (1<<1); // Power-ON TIMER 0
46     LPC_TIMO->PR = 0x00; // Prescaler igual a 0
47     LPC_TIMO->MCR = 0x18; // Interrumpe y resetea TC cuando se alcance el valor de MR1
48     LPC_TIMO->MR1 = (F_pclk-1); // F_pclk=25e6 por lo tanto interrumpirá cada segundo
49     LPC_TIMO->TCR = 0x01; // Se habilita el TC y el PC para contar.
50     NVIC_EnableIRQ (TIMER0_IRQn); // TC y PC habilitados para contar
51 }

```

Figura 17 - Definición de la función init\_TIMER0.

- **Init\_ADC\_sensores():** Configura el ADC para que muestree el canal 2 y 4 del ADC donde están conectados los sensores analógicos, *LM35* y *HIH4000* respectivamente. Se configura para comenzar la conversión en función del Match1 del Timer0, por lo tanto, muestrea cada 2 segundos, y además, para interrumpir cuando finalice la conversión del canal 2. Como se puede observar en la siguiente figura se trata de una función sin argumentos de entrada y sin retorno. Como se puede observar en la siguiente figura se trata de una función sin argumentos de entrada y sin retorno.

```

74 /* ----- Configuración ADC ----- */
75
76 void init_ADC_sensores(void)
77 {
78     LPC_SC->PCONP |= (1<<12); // Power ON
79     LPC_SC->PCLKSELO &= ~(3<<24); // CLK ADC = CCLK/4 (Fclk después del reset) (100 Mhz/4 = 25Mhz)
80     LPC_ADC->ADCR = (0x01<<2); // Canal 2
81     LPC_ADC->ADCR = (0x01<<4); // Canal 4
82     LPC_ADC->ADCR = (0x0F<<8); // CLKDIV=15 (Fclk_ADC= 25Mhz/(15+1)= 1.5625 Mhz)
83     LPC_ADC->ADCR = (4<<24); // Inicio de conversión con el Match 1 del Timer 0
84     LPC_ADC->ADCR = (0x01<<21); // PDN=1 ADC funcional
85
86     LPC_ADC->ADINTEN = (1<<2); // Hab. interrupción fin de conversión del PENÚLTIMO canal (canal 2)
87     NVIC_EnableIRQ (ADC_IRQn);
88     NVIC_SetPriority (ADC_IRQn,4);
89 }

```

Figura 18 - Definición de la función init\_ADC\_sensores.

- **Init\_ADC\_grabar():** Configura el ADC para grabar el sonido del micrófono y convertir las muestras de la alarma, en este caso se emplea el canal 5, al cual estará conectado la salida del micrófono mencionado en apartados anteriores.

```

90
91 void init_ADC_grabar(void)
92 {
93     LPC_SC->PCONP |= (1<<12); // Power ON
94     LPC_SC->PCLKSELO &= ~(3<<24); // CLK ADC = CCLK/4 (Fclk después del reset) (100 Mhz/4 = 25Mhz)
95     LPC_ADC->ADCR = 0;
96     LPC_ADC->ADCR = (1<<5); // Canal 5
97     LPC_ADC->ADCR = (1<<8); // CLKDIV=1 (Fclk_ADC= 25Mhz/(1+1)= 12.5 Mhz)
98     LPC_ADC->ADCR = (7<<24); // Inicio de conversión con el Match 1 del Timer 1
99     LPC_ADC->ADCR = (1<<21); // PDN=1 ADC funcional
100     LPC_ADC->ADINTEN = (1<<5); // Hab. interrupción fin de conversión canal 5, necesario para DMA
101 }
102

```

Figura 19 - Definición de la función init\_ADC\_grabar.

- **Init\_TIMER1():** Configura el TIMER1 para la grabación de sonido mediante el micrófono y la conversión mediante el ADC. Se configura para que interrumpa a una frecuencia de 8Khz de manera que el ADC, cuando opere en modo grabación, muestree a dicha frecuencia.

```

53 // TIMER1 para grabación (ADC-DMA)
54 void init_TIMER1(void)
55 {
56     LPC_SC->PCONP |= (1<<2); // Power-ON TIMER 1
57     LPC_TIMER1->PR = 0x00; // Prescaler igual a 0
58     LPC_TIMER1->MCR = 0x18; // Reset del TC cuando se alcance el valor de MR1
59     LPC_TIMER1->MR1 = (F_pclk/F_muestreo_rec/2)-1; // DOS Match para iniciar la conversión!!!!
60     LPC_TIMER1->TCR = 0x03; // TC y PC habilitados para contar + reset del timer
61     LPC_TIMER1->TCR &= ~(1<<1); // Des-reset
62     NVIC_EnableIRQ(TIMER1_IRQn); //Habilitamos interrupción del Timer1
63 }

```

Figura 20 - Definición de la función init\_Timer1.

- **Init\_TIMER3():** Configura el TIMER3 en modo captura para calcular la velocidad a la que gira el anemómetro, para ello, se le obliga a contar los flancos de bajada del pint CAP3.1 al cual está conectado el anemómetro.

PROYECTO FINAL ESTACIÓN METEOROLÓGICA CON CONEXIÓN A INTERNET

```
66 void init_TIMER3(void)
67 {
68     LPC_SC->PCONP|=(1<<23); // Power-ON TIMER 2
69     LPC_TIM3->CTCR|=(1<<1)|(1<<2); // Modo contador, Cuenta en falling edges de CAP3.1 (P0.24)
70     LPC_TIM3->TCR |= 1 << 0; // Start contador
71 }
72
```

Figura 21 - Definición de la función init\_timer3.

## Uart.c

Este archivo contiene las funciones para configurar el funcionamiento del interfaz serie asíncrono empleado, en este caso, la UART0. Las principales funciones son:

- **Uart0\_init(baudrate):** Función que inicializa la UART0 con la velocidad que recibe como parámetro.

```
131 void uart0_init(int baudrate) {
132
133     LPC_PINCON->PINSEL0|=(1<<4)|(1<<6); // Cambia el modo de P0.2 y P0.3 a TXD0 y RXD0
134
135     LPC_UART0->LCR &= ~STOP_1_BIT & ~PARITY_NONE; // Modo 8N1 (8 bits/dato, sin paridad, y 1 bit de stop)
136     LPC_UART0->LCR |= CHAR_8_BIT;
137
138     uart0_set_baudrate(baudrate); // Se configura la velocidad
139
140
141     LPC_UART0->IER = THRE_IRQ_ENABLE|RBR_IRQ_ENABLE; // Se habilitan las interrupciones de transmisión y recepción de la UART
142     NVIC_EnableIRQ(UART0_IRQn); // Se habilita la interrupción de la UART
143
144 }
145
```

Figura 22 - Definición de la función uart0\_init.

- **Uart0\_set\_baudrate(baudrate):** Configura los registros de la UART0 para que tenga la velocidad en baudios por segundo que se le ha pasado como parámetro.

```
75 uClk = uClk >> 4; /* div by 16 */
76 /* The formula is :
77 * BaudRate= uClk * (mulFracDiv/(mulFracDiv+dividerAddFracDiv) / (16 * DLL)
78 *
79 * The value of mulFracDiv and dividerAddFracDiv should comply to the following expressions:
80 * 0 < mulFracDiv <= 15, 0 <= dividerAddFracDiv <= 15*/
81
82 for (mulFracDiv = 1; mulFracDiv <= 15; mulFracDiv++) {
83     for (dividerAddFracDiv = 0; dividerAddFracDiv <= 15; dividerAddFracDiv++) {
84         temp = (mulFracDiv * uClk) / (mulFracDiv + dividerAddFracDiv);
85
86         divider = temp / baudrate;
87         if ((temp % baudrate) > (baudrate / 2))
88             divider++;
89
90         if (divider > 2 && divider < 65536) {
91             calcBaudrate = temp / divider;
92
93             if (calcBaudrate <= baudrate) {
94                 relativeError = baudrate - calcBaudrate;
95             } else {
96                 relativeError = calcBaudrate - baudrate;
97             }
98
99             if (relativeError < relativeOptimalError) {
100                 mulFracDivOptimal = mulFracDiv;
101                 dividerAddOptimal = dividerAddFracDiv;
102                 dividerOptimal = divider;
103                 relativeOptimalError = relativeError;
104                 if (relativeError == 0)
105                     break;
106             }
107         }
108     }
109     if (relativeError == 0)
110         break;
111 }
112 if (relativeOptimalError < ((baudrate * UART_ACCEPTED_BAUDRATE_ERROR) / 100)) {
113
114     LPC_UART0->LCR |= DLAB_ENABLE; // importante poner a 1
115     LPC_UART0->DLM = (unsigned char) ((dividerOptimal >> 8) & 0xFF);
116     LPC_UART0->DLL = (unsigned char) dividerOptimal;
117     LPC_UART0->LCR &= ~DLAB_ENABLE; // importante poner a 0
118
119     LPC_UART0->FDR = ((mulFracDivOptimal << 4) & 0xF0) | (dividerAddOptimal & 0x0F);
120
121     errorStatus = 0; //< Success
122 }
123
124 return errorStatus;
```

Figura 23 - Definición de la función Uart0\_set\_baudrate.

- **Tx\_cadena\_UART0():** Se encarga de transmitir al registro de transmisión de la UART, las cadenas de caracteres que se le pasan como parámetro.

```
49 void tx_cadena_UART0(char *cadena)
50 {
51     ptr_tx=cadena;
52     tx_completa=0;
53     LPC_UART0->THR=*ptr_tx++; // IMPORTANTE: Introducir un carácter al comienzo para iniciar TX o
54     // activar flag interrupción por registro transmisor vacío
55 }
```

Figura 24 - Definición de la función tx\_cadena\_UART0.

- **UART0\_IRQHandler():** Es la función de atención a la interrupción de la UART0. Se encarga de gestionar la recepción y el envío de la información a través de la UART0 en función de la interrupción que se ha generado (RBR o THRE).

```
21 void UART0_IRQHandler(void) {
22
23     switch(LPC_UART0->IIR&0x0E) {
24
25     case 0x04: // RBR, El buffer de recepción está listo
26         *ptr_rx=LPC_UART0->RBR; // Lee el dato recibido y lo almacena
27         if (*ptr_rx++ ==13) // Caracter return --> Cadena completa
28         {
29             *ptr_rx=0; //Añadimos el caracter null para tratar los datos recibidos como una cadena
30             rx_completa = 1; // Recepción completa */
31             ptr_rx=buffer; // Puntero al inicio del buffer para nueva recepción
32         }
33         break;
34
35     case 0x02: // THRE, El registro de transmisión está vacío
36         if (*ptr_tx!=0)
37             LPC_UART0->THR=*ptr_tx++; // carga un nuevo dato para ser transmitido
38         else
39             tx_completa=1; //Transmisión completa
40         break;
41     }
42 }
43
44
45
```

Figura 25 - Definición de la función UART0\_IRQHandler.



## DMA.c

Contiene la configuración del DMA para trabajar con el ADC cuando se graba la alarma y para trabajar con el DAC cuando se reproduce el sonido de ésta. Además, contiene la función de interrupción del TIMER1 que se encarga de controlar el muestreo del audio cuando se dan las condiciones necesarias para grabar el mensaje de alarma. Se ha empleado este dispositivo para la grabación y reproducción del mensaje de alarma con el objetivo de eliminar carga de trabajo en la CPU, de manera que esta pueda seguir realizando operaciones mientras que el DMA se encarga de llevar a cabo la transmisión de datos de memorias a periférico o viceversa. Las principales funciones que componen el archivo son:

- **Init\_DMA\_ADC():** Configura el DMA para realizar transferencia periférico a memoria (registro ADDR5 del ADC a array de muestras) de las muestras obtenidas en el canal 5 del ADC. Se emplean bloques de transmisión de 4000bits, de manera que, al usar muestras 16000 muestras de 8 bits para obtener el mensaje adquiridas a una frecuencia de muestreo de 8Khz (duración del mensaje 2s), se necesiten cuatro transferencias de este tipo para completar el array. Cabe destacar que, debido a las limitaciones de memoria del microcontrolador y la placa de desarrollo, el número de muestras y el tamaño de estas está enormemente limitado, por ello, se ha apreciado en las pruebas llevados a cabo un detrimento notable de la calidad de la grabación usada. Emplea el canal 0.

```

68 void init_DMA_ADC(void)
69 {
70
71     LPC_SC->PCONF |= 1 << 29; //Power-On DMA
72     LPC_GPDMA->DMACConfig |= (1<<0); // Hab. DMA
73
74     //Origen y destino.
75     LPC_GPDMA->DMACDestAddr = (uint32_t)&muestras[index1]; //Dir.Array muestras
76     LPC_GPDMA->DMACSrcAddr = LPC_ADC_BASE + OFFSET_ADC; //Dir.ADGR del ADC, podriamos usar la dir. ADDR5 también, eso seria + 0x29
77     LPC_GPDMA->DMACCLI = 0;
78
79     //S.BURST=1 | D.Burst=1 | 8 bits | 8 bits | Incr. Dest | TC interrupt enable.
80     LPC_GPDMA->DMACControl |= (SBURST<<12) | (DBURST<<15) | (WIDTH_VALUE<<18) | (WIDTH_VALUE<<21) | (1 << 27) | (1 << 31);
81     LPC_GPDMA->DMACControl |= TAM_BLOCK_DMA; //Tamaño maximo.
82
83     //ORIGEN: ADC | Dest=mem. | P2M | ERROR/TC MASK | Inicio
84     LPC_GPDMA->DMACConfig = (4 << 1) | (0x00 << 6) | (2 << 11) | (1 << 15) | (1 << 14) | 1;
85     NVIC_EnableIRQ(DMA_IRQn);
86 }

```

Figura 26 - Definición de la función init\_DAM\_ADC.

- **Init\_DMA\_DAC():** Configura el DMA para realizar transferencias de memoria a periférico (array de muestras a DAC), con un tamaño de bloque de transferencia de 4000 bits y muestras de 8 bits. Emplea el canal 1.

PROYECTO FINAL ESTACIÓN METEOROLÓGICA CON CONEXIÓN A INTERNET

```

27 void init_DMA_DAC(void)
28 {
29     /*Configuración del DMA para llevar muestras del buffer al DAC*/
30
31     LPC_SC->PCONF |= (1<<29); // Power DMA
32     LPC_GPDMA->DMAConfig = 1; // enable the GPDMA controller
33     LPC_GPDMA->DMAChn0 = (1<<6); // enable synchro logic for all reqs
34
35     LPC_GPDMA->DMAChn0SsrAddr = (uint32_t) muestras[index1]; // Dir. Mem buffer muestras
36     LPC_GPDMA->DMAChn0DestAddr = (uint32_t) &(LPC_DAC->DACR); // Dir. Mem registro conversión DAC
37
38     LPC_GPDMA->DMAChn0Ctrl = 0; // No usamos link list
39     LPC_GPDMA->DMAChn0CCntrol = TAM_BLOCK_DMA; // Transferencia 4 veces ya que 2 seg de audio son 16000 muestras
40     | (0 << 12); // Tamaño de ráfaga de la fuente (12 - 14) = 1
41     | (0 << 13); // Tamaño de la ráfaga en el destino (15 - 17) = 1
42     | (0 << 18); // Bits de los datos de la fuente (18 - 20) = 8 bit poniendo los tres bits a 0
43     | (0 << 21); // Bits de datos en el destino (21 - 23) = 8 bit
44     | (0 << 24); // Debe ser 0
45     | (0 << 25); // Debe ser 0
46     | (1 << 24); // Se incrementa la dirección de la fuente (24)
47     | (0 << 27); // destination increment (27) en este caso no incrementa ya que es el Dac , si fuera una posición de memoria
48     | (0 << 28); // mode select (28) = access in user mode
49     | (0 << 29); // (29) = access not bufferable
50     | (0 << 30); // (30) = access not cacheable
51     | (1 << 31); // terminal count interrupt enabled
52
53     LPC_GPDMA->DMAChn0Config = 1; // canal activado (0)
54     | (0 << 1); // periférico de origen (1 - 5) = none seleccionamos de que va a ir a que va a ir
55     | (7 << 4); // periférico de destino (6 - 10) = DAC
56     | (1 << 11); // Flujo (11 - 13) = MEM to PERP
57     | (1 << 14); // (14) = mask out error interrupt
58     | (1 << 15); // (15) = mask out terminal count interrupt
59     | (0 << 14); // (14) = no locked transfers
60     | (0 << 17); // (17) = no MALT
61
62     /*Configuración del DAC*/
63     LPC_DAC->DACCTRL = F_pclk/8000-1; // El contador del DAC_DMA empieza a contar refrescándose desde este valor
64     LPC_DAC->DACCTRL = (1<<1) | (1<<2) | (1<<3); // Buffer doble, user-out y acceso de DMA habilitados
65     NVIC_EnableIRQ(DMA_IRQn);
66 }

```

Figura 27 - Definición de la función init\_DMA\_DAC.

- **DMA\_IRQHandler():** Se trata de la función de atención a la interrupción del DMA. Configura el comportamiento del DMA en función del canal que interrumpa, si se trata del canal 0, se querrán enviar las muestras desde el ADC al array de muestras y, en caso contrario, se querrán enviar desde dicho array al DAC. La función tiene en cuenta el número de transferencias realizadas por el DMA, de manera que cuando se realizan cuatro transferencias (el array de muestras estaría completo puesto que tiene 16000 muestras de 8 bits), esta función borra los flags necesarios del DMA, resetea los contadores, apaga los dispositivos empleados para ahorrar consumo (Timer1, DMA) y, en adición, devuelve al ADC a su comportamiento normal para adquirir datos de los sensores analógicos.

```

88 void DMA_IRQHandler(void)
89 {
90     if (counter < 4)
91     {
92         index1 = TAM_BLOCK_DMA * counter; //Aumentamos índice búffer muestras
93
94         if (LPC_GPDMA->DMAIntStat & 2)
95         {
96             if (LPC_GPDMA->DMAIntTCStat & 2)
97             {
98                 LPC_GPDMA->DMAIntTCClear = 2; // borramos int.Ch1
99                 init_DMA_ADC();
100             }
101         }
102         else // Como solo tenemos dos canales en uso, este sería el canal 0
103         {
104             LPC_GPDMA->DMAIntTCClear = 1; // borramos int.Ch0
105             init_DMA_DAC();
106         }
107         counter++;
108     }
109     else
110     {
111         //NVIC_DisableIRQ(DMA_IRQn);
112         LPC_GPDMA->DMAIntTCClear = 1;
113         LPC_GPDMA->DMAIntTCClear = 2;
114         LPC_GPDMA->DMAIntErrClr = 1;
115         LPC_GPDMA->DMAIntErrClr = 2;
116
117         //Reseteamos valores
118         counter = 1;
119         index1 = 0;
120
121         //Apagamos dispositivos
122         apagar_DMA();
123         apagar_TIMER1();
124
125         //Rehabilitamos la ADC Sensores
126         init_ADC_sensores();
127     }
128 }

```

Figura 28 - Definición de la función DMA\_IRQHandler.

- **TIMER1\_IRQHandler():** Es la función de atención a la interrupción del TIMER1. Se ha implementado para dar opción al usuario de configurar la grabación mediante el uso del DMA o directamente tomando las muestras desde el ADC,

en este caso, la CPU tendrá una mayor carga. Como se ha mencionado, esta función, tiene una frecuencia de interrupción de 8Khz y, por ello, toma las muestras del ADC a dicha velocidad, una vez adquiridas las 16000 muestras que puede almacenar el array de muestras, deshabilita su propia interrupción, apaga el timer1 para evitar un mayor consumo y devuelve al estado de adquirir datos de los sensores analógicos al ADC.

```

130 //Función de interrupción para tomar las muestras del ADC
131 void TIMER1_IRQHandler()
132 {
133     if((LPC_TIMER1->IR & 0x02) == 0x02) //Comprobamos si se interrumpe por MR1
134     {
135         LPC_TIMER1->IR |= (1<<1); //Reset flag interrupción MR1
136
137         muestras[index1]=((LPC_ADC->ADDR5) >> 8)&0xFF; //Muestras de 8bits
138         index1++;
139
140         if (index1 == 16000)
141         {
142             index1 = 0;
143             NVIC_DisableIRQ(TIMER1_IRQn);
144             apagar_TIMER1();
145             init_ADC_sensores();
146         }
147     }
148 }
149

```

Figura 29 - Definición de la función TIMER1\_IRQHandler.

- **Rec\_ADC():** Se encarga de llevar a cabo la grabación de sonido directamente desde el ADC, emplea la interrupción del Timer1.

```

151 void rec_ADC()
152 {
153     NVIC_DisableIRQ(ADC_IRQn);
154     init_ADC_grabar();
155     init_TIMER1();
156     LPC_SC->EXTINT=(1<<1); // Borrar flag Externa 1
157 }
158

```

Figura 30 - Definición de la función rec\_ADC.

- **Rec\_DMA():** Se encarga de llevar a cabo la grabación de sonido mediante ADC haciendo uso del DMA.

```

159 void rec_DMA()
160 {
161     NVIC_DisableIRQ(ADC_IRQn);
162     init_TIMER1();
163     NVIC_DisableIRQ(TIMER1_IRQn); //Al usar DMA, no es necesario que T1 interrumpa
164     init_ADC_grabar();
165     init_DMA_ADC();
166     LPC_SC->EXTINT=(1<<1); // Borrar flag Externa 1
167 }
168

```

Figura 31 - Definición de la función rec\_DMA.

- **Play():** Permite reproducir la grabación almacenada en el array de muestras haciendo uso del DAC y el DMA.

```

169 void play()
170 {
171     NVIC_DisableIRQ(ADC_IRQn);
172     init_DMA_DAC();
173     LPC_SC->EXTINT=(1<<2); // Borrar flag Externa 2
174 }
175

```

Figura 32 - Definición de la función play.



## PWM.c

Este archivo contiene la función de configuración del pulse Width Modulator empleado para configurar el ciclo de trabajo del ventilador de refrigeración del sistema en función de la temperatura del sensor digital de temperatura *DS1621*. Contiene las siguientes funciones:

- **Init\_PWM():** Configura la PWM1.1 con un periodo de 20 ms y habilita la salida de la señal en el pin P1.18 para poder conectar el ventilador de refrigeración.

```
4 void init_PWM(void)
5 {
6     //Se ha seleccionado PWM1.1 en el P1.18 en el archivo init.c
7     LPC_SC->PCONP |= (1<<6); //PWM1 ON
8     LPC_PWM1->MR0 = F_pclk*Tpwm; //Periodo de la señal PWM de 20ms
9     LPC_PWM1->MCR |= (1<<1); //Reset del TC cuando se alcanza el valor de MR0
10    LPC_PWM1->PCR |= (1<<9); //configurado el ENAL (PWM1.1 salida por pin P1.18 ó P2.0)
11    LPC_PWM1->TCR |= (1<<0) | (1<<3); //PWMTTC y PWMPC habilitados para contar, modo PWM habilitado
12 }
13
```

Figura 33 - Definición de la función init\_PWM.

- **Set\_ciclo\_trabajo\_PWM():** Configura el ciclo de trabajo de la PWM en función del parámetro que se le pasa como argumento. Se emplea para configurar la velocidad del ventilador encargado de refrigerar el sistema.

```
16 void set_ciclo_trabajo_PWM ( float temperatura)
17 {
18     LPC_PWM1->MR1 = (LPC_PWM1->MR0*temperatura/100); //Ciclo de trabajo
19     LPC_PWM1->LER |= (1 << 1) | (1<<0); //Se aplican los cambios
20 }
21
```

Figura 34 - Definición de la función set\_ciclo\_trabajo\_PWM.

## DS1621.c

Este archivo contiene las funciones de configuración para la conversión continua de muestras y de lectura de los datos obtenidos en el sensor *DS1621*.

- **Config\_DS1621():** Configura el sensor para que realice la conversión de temperatura de manera continua.

```
6 void config_DS1621(void)
7 {
8     I2C_SendAddr(0x48,0); //Dir.Slave 0x48 (A2=A1=A0=0) + escritura
9     I2C_SendByte(0xAC); //Acceso al registro de configuración
10    I2C_SendByte(0x02); //Modo conversión continua
11    I2C_SendStop();
12    I2C_Delay();
13    I2C_SendAddr(0x48,0); //Dir.Slave 0x48 (A2=A1=A0=0) + lectura
14    I2C_SendByte(0xEE); //Inicio conversión, continua, de la temperatura
15    I2C_SendStop();
16 }
```

Figura 35 - Definición de la función config\_DS1621.

- **Leer\_DS1621():** Realiza la lectura de los dos bytes de temperatura (parte real y parte decimal de esta) y realiza la conversión a float de la temperatura.

```
18 float leer_DS1621()
19 {
20     signed char entero,decimal;
21     float f_entero=0.0,f_decimal=0.0;
22     I2C_SendAddr(0x48,0); //Dir.Slave 0x48 (A2=A1=A0=0) + escritura
23     I2C_SendByte(0xAA); //Leemos temperatura
24     I2C_SendAddr(0x48,1);
25     entero = I2C_GetByte('0'); //Leemos parte entera de la temperatura (8b de mayor peso)
26     decimal = (I2C_GetByte('1') >> 7); //Leemos parte decimal de la temperatura (8*bit de los ocho bits de menor peso)
27     I2C_SendStop(); //Bus en reposo
28     f_entero=(float)entero;
29     f_decimal =(float)decimal;
30     if(decimal==1){
31         if(f_entero>0)
32             {f_entero=f_entero+0.5;}
33         else
34             {f_entero=f_entero-0.5;}
35     }
36     return (f_entero); //Convertimos a float
37 }
```

Figura 36 - Definición de la función leer\_Ds1621.

## BMP180.c

Este archivo contiene las funciones necesarias para calibrar correctamente el sensor, medir la presión y la temperatura dadas por este y corregidas con los pertinentes factores de calibración y, además, calcular la altitud absoluta con respecto al nivel del mar a partir de la presión obtenida.

- **Read\_calibration\_data():** Realiza la lectura de las 11 variables de calibración de la memoria E<sup>2</sup>PROM. Dado que el código de esta función es muy extenso no se ha incluido en la memoria, para más información véase el archivo BMP180.c del proyecto adjunto.
- **Read\_uncompensated\_temp():** Realiza la lectura de la medida de la temperatura sin calibrar.

```
177 void read_uncompensated_temp (void)
178 {
179     I2CSendAddr(0x77,0);           //Dirección slave + escritura
180     I2CSendByte(0xF4);             //Seleccionamos reg.medidas
181     I2CSendByte(0x2E);             //Configuramos para temperatura
182     I2Cdelay();                    //Esperamos 4.5ms
183     I2CSendAddr(0x77,0);           //Dirección slave + escritura
184     I2CSendByte(0xF6);             //Leemos MSB Temp
185     I2Cdelay();
186     I2CSendAddr(0x77,1);
187     UT = I2CGetByte('1') << 8;    //MSB
188     I2CSendStop();
189     I2CSendAddr(0x77,0);           //Dirección slave + lectura
190     I2CSendByte(0xF7);
191     I2Cdelay();
192     I2CSendAddr(0x77,1);
193     UT += I2CGetByte('1');         //LSB
194 }
```

Figura 37 - Definición de la función read\_uncompensated\_temp.

- **Calculate\_temp():** Calcula la temperatura real en grados centígrados aplicando las variables de calibración.

```
219 float calculate_temp (void)       //T* calibrada
220 {
221     X1 = (UT - AC6) * AC5/32768;
222     X2 = MC * 2048/(X1 + MD);
223     B5 = X1 + X2;
224     t_bmp = (B5 + 8)/64;
225     return (t_bmp*0.1);           //Temperatura en °C
226 }
```

Figura 38 - Definición de la función calculate\_temp.

- **Read\_uncompensated\_press():** Realiza la lectura de la medida de la presión sin calibrar<sup>10</sup>.

```

196 void read_uncompensated_press (void)
197 {
198     I2C_SendAddr(0x77,0);           //Dirección slave + escritura
199     I2C_SendByte(0xF4);             //Seleccionamos req.medidas
200     I2C_SendByte(0x74);             //Configuramos para presión (oss=1 -> Modo std, 7.5ms de conversión, 2muestras)
201     I2C_Delay();
202     I2C_Delay();                     //Esperamos 7.5ms en total
203     I2C_SendAddr(0x77,0);           //Dirección slave + escritura
204     I2C_SendByte(0xF6);             //Leemos MSB Temp
205     I2C_Delay();
206     I2C_SendAddr(0x77,1);
207     UP = I2C_GetByte('1') << 16;    //MSB
208     I2C_SendStop();
209     I2C_SendAddr(0x77,0);           //Dirección slave + lectura
210     I2C_SendByte(0xF7);
211     I2C_Delay();
212     I2C_SendAddr(0x77,1);
213     UP += I2C_GetByte('1') << 8;     //LSB. Nota: Podríamos leer el XLSB si empleásemos el modo de conversión "Ultra high resolution"
214     I2C_SendStop();
215     UP = UP >> (8-1);                //Desplazamos 8-oss (datasheet requirement).
216 }

```

Figura 39 - Definición de la función read\_uncompensated\_press.

- **Calculate\_press():** Calcula el valor de la presión en hecto pascales aplicando las variables de calibración.

```

228 float calculate_press (void)         //Presión calibrada
229 {
230     B6 = B5 - 4000;
231     X1 = (B2 * (B6*B6/4096))/2048;
232     X2 = AC2 * B6/2048;
233     X3 = X1 + X2;
234     B3 = ((AC1 * 4 + X3)<< 1)+2)/4;
235     X1 = AC3 * B6/8192;
236     X2 = (B1*(B6*B6/4096))/65536;
237     X3 = (X1 + X2)+2)/4;
238     B4 = AC4*(unsigned int) (X3+32768)/32768;
239     B7 = ((unsigned int)UP - B3)*(5000 >> 1);
240
241     if (B7 < 0x80000000)
242         p_bmp = (B7 * 2)/B4;
243     else
244         p_bmp = (B7/B4)+2;
245
246     X1 = (p_bmp/256) * (p_bmp/256);
247     X1 = (X1 * 3038)/65536;
248     X2 = (-7357*p_bmp)/65536;
249     p_bmp = p_bmp + (X1+X2+3791)/16;
250     return ((SEA_LVL_PRESS-p_bmp)*0.01);           //Presión en hPa
251 }

```

Figura 40 - Definición de la función calculate\_press.

- **Calculate\_altitude():** Calcula la altitud absoluta respecto al nivel del mar.

```

253 float calculate_altitude (void)       //Altitud absoluta respecto al nivel del mar en metros
254 {
255     float altitude = 0.0;
256     float pressure = SEA_LVL_PRESS-p_bmp;
257     altitude = 44330*(1-pow((pressure/SEA_LVL_PRESS), (1/5.255)));
258     return altitude;
259 }

```

Figura 41 - Definición de la función calculate\_altitude.

- **I2CRestart():** Invoca las funciones I2C\_SendStop() e I2C\_Delay() de la librería i2c\_lpcxx, antes mencionada, para garantizar un tiempo mínimo entre la condición de STOP y la condición START.

<sup>10</sup> Para conseguir más precisión se ha utilizado el el parámetro oss = 1, de manera que toman dos muestras, el tiempo de conversión es de 7.5 ms y se reduce el ruido en la medición.



## Main.c

Es el archivo principal del sistema y contiene la función main(), puesto que se tomó como referencia para realizar el proyecto, el ejemplo "http\_demo" y se ha construido a partir de este el proyecto, este fichero tiene el nombre "http\_demo.c". Incluye numerosas funciones necesarias para el correcto funcionamiento del servidor web así como del resto de elementos. Las funciones más destacables son:

- **Timer\_pool():** Configura el SysTick en modo sondeo. Genera un tick cada 100 ms necesario para para la comunicación TCP/IP y el correcto funcionamiento del servidor web.

```
75 |
76 | /*----- timer_poll -----*/
77 |
78 | static void timer_poll () {
79 |     /* System tick timer running in poll mode */
80 |
81 |     if (SysTick->CTRL & 0x10000) {
82 |         /* Timer tick every 100 ms */
83 |         timer_tick ();
84 |         tick = __TRUE;
85 |     }
86 | }
```

Figura 42 - Definición de la función timer\_pool().

- **Dhck\_check():** Se encarga de monitorizar la asignación de la dirección web del servidor mediante DHCP.

```
136 | static void dhcp_check () {
137 |     /* Monitor DHCP IP address assignment. */
138 |
139 |     if (tick == __FALSE || dhcp_tout == 0) {
140 |         return;
141 |     }
142 |
143 |     if (mem_test (&MY_IP, 0, IP_ADRLen) == __FALSE && !(dhcp_tout & 0x80000000)) {
144 |         /* Success, DHCP has already got the IP address. */
145 |         dhcp_tout = 0;
146 |         sprintf((char *)lcd_text[0], " IP address:");
147 |         sprintf((char *)lcd_text[1], " %d.%d.%d.%d", MY_IP[0], MY_IP[1],
148 |                                     MY_IP[2], MY_IP[3]);
149 |         LCDupdate = __TRUE;
150 |         return;
151 |     }
152 |     if (--dhcp_tout == 0) {
153 |         /* A timeout, disable DHCP and use static IP address. */
154 |         dhcp_disable ();
155 |         sprintf((char *)lcd_text[1], " DHCP failed  ");
156 |         LCDupdate = __TRUE;
157 |         dhcp_tout = 30 | 0x80000000;
158 |         return;
159 |     }
160 |     if (dhcp_tout == 0x80000000) {
161 |         dhcp_tout = 0;
162 |         sprintf((char *)lcd_text[0], " IP address:");
163 |         sprintf((char *)lcd_text[1], " %d.%d.%d.%d", MY_IP[0], MY_IP[1],
164 |                                     MY_IP[2], MY_IP[3]);
165 |         LCDupdate = __TRUE;
166 |     }
167 | }
```

Figura 43 - Definición de la función DHCP\_check.

- **Init():** Esta función se invoca desde la función main(). Configura las prioridades de los pulsadores utilizados para grabar y reproducir audio, la prioridad del

TIMERO y configura el SysTick para que interrumpa cada 100ms. Además, invoca numerosas funciones de inicialización de distintos elementos como puedan ser los GPIOs, el PWM, el ADC, el sensor de temperatura DS1621, el display LCD y el protocolo TCP.

```
38
39
40 /*----- init -----*/
41
42 static void init ()
43 {
44     //Funciones de atención a la interrupción
45     NVIC_SetPriorityGrouping(3);
46     NVIC_SetPriority(UART0_IRQn, 0x0);
47     NVIC_SetPriority(TIMERO_IRQn, 0x1);
48     NVIC_SetPriority(EINT1_IRQn, 0);
49     NVIC_SetPriority(EINT2_IRQn, 0);
50     NVIC_EnableIRQ(EINT1_IRQn);
51     NVIC_EnableIRQ(EINT2_IRQn);
52
53     //Inicialización GPIO
54     init_GPIO();
55
56     //Inicialización PWM
57     init_PWM();
58
59     //Inicialización ADC para los sensores analógicos
60     init_ADC_sensores();
61
62     //Configuración DS1621
63     config_DS1621();
64
65     //Inicialización LCD y TCP
66     init_display ();
67     init_TcpNet ();
68
69     /* Setup and enable the SysTick timer for 100ms. */
70     SysTick->LOAD = (SystemCoreClock / 10) - 1;
71     SysTick->CTRL = 0x05;
72 }
73
```

Figura 44 - Definición de la función init.

- **TIMERO\_IRQHandler():** Se trata de la función de atención a la interrupción del TIMERO que se ha configurado para interrumpir cada segundo, es una de las funciones de mayor importancia del programa desarrollado. Dentro de esta función se llevan a cabo numerosas tareas:
  - **Se transmiten los datos de los sensores** de temperatura, de presión, de humedad y la velocidad del viento medida por el anemómetro **a través de la UART0**
  - Se emplea como **referencia de inicio de conversión del ADC** para los sensores analógicos (cada 1s).
  - **Permite refrescar el LCD cada 5 segundos.**
  - Se **activa el modo BURST del ADC** para que realice el muestreo los sensores analógicos de modo continuo.

PROYECTO FINAL ESTACIÓN METEOROLÓGICA CON CONEXIÓN A INTERNET

```

187 void TIMER0_IRQHandler(void) // Interrumpe cada segundo
188 {
189     //Contadores para enviar datos por UART y LCD
190     static U32 contador_uart, contador_disp=0;
191
192     //Activación del modo BURST - ADC sensores analógicos
193     LPC_ADC->ADCR |= (1<<16); // BURST=1 --> Cada 65536 ADC se toma una muestra de cada canal comenzando desde el más bajo (bit LSB de CR[0..7])
194
195     //Lectura velocidad anemómetro
196     vel_anemometro = LPC_TIM3->TC * 2 * pi * radio_spinner; // Medida en m/s
197     LPC_TIM3->TCR |= 1<<1; // Reset contador (Timer3)
198     LPC_TIM3->TCR |= ~(1<<1); // Out Reset contador (si no se mantiene reseteado)
199
200     //Lectura datos BMP
201     read_uncompensated_temp();
202     read_uncompensated_press();
203     temp_BMP180 = calculate_temp();
204     presion = calculate_press();
205     altitud = calculate_altitude();
206
207     //TX datos UART y LCD
208     if(tx_completa)
209     {
210         switch(contador_uart){
211             case 0:
212             {
213                 sprintf((char*)buff_env, "TEMPERATURAS\n LM35: %.1f C\nDS1621: %.1f C\nBMP180: %.1f C\nUmbral: %d C\n", temp_LM35, temp_DS1621, temp_BMP180, umbral_temp);
214                 tx_cadena_UART0(buff_env);
215                 contador_uart++;
216                 break;
217             }
218             case 1:
219             {
220                 sprintf((char*)buff_env, "PRESION: %.2f hPa\nHUMEDAD: %.2f %%\n", presion, humedad);
221                 tx_cadena_UART0(buff_env);
222                 contador_uart++;
223                 break;
224             }
225             case 2:
226             {
227                 sprintf((char*)buff_env, "VIENTO: %.2f m/s\nALTITUD: %.2f m\n", vel_anemometro, altitud);
228                 tx_cadena_UART0(buff_env);
229                 contador_uart=0;
230                 break;
231             }
232         }
233         if(contador_disp>5) //Actualizamos display cada 2.5s
234         {
235             contador_disp=0;
236             upd_display ();
237         }
238         contador_disp++;
239         LPC_TIM0->IR |= (1<<1); // Borrar flag interrupción
240     }
}

```

Figura 45 - Definición de la función TIMER0\_IRQHandler.

- **ADC\_IRQHandler():** Es la función de atención a la interrupción del ADC. Lee los registros del canal 2 y 4 para calcular la temperatura y la humedad que miden los sensores analógicos. Además, invoca la función que obtiene la medida de temperatura del sensor DS1621. **Esto último se ha realizado por no sobrecargar la función de atención a la interrupción del TIMER0.**

```

174 void ADC_IRQHandler(void)
175 {
176
177     LPC_ADC->ADCR |= ~(1<<16); // BURST=0 // Deshabilitamos el modo Ráfaga (ojo continua la conversión del siguiente canal)
178
179     //Almacenamos las muestras
180     temp_LM35 = (((LPC_ADC->ADDR2 >>4)&0xFFFF)*3.3/4095)*10; //Temperatura LM35 en °C
181     humedad = (((LPC_ADC->ADDR4 >>4)&0xFFFF)*3.3/4095)-0.772)/0.03; //Humedad relativa
182     temp_DS1621=leer_DS1621();
183 }

```

Figura 46 - Definición de la función ADC\_IRQHandler.

- **Init\_display():** Función de inicialización del display. Muestra la información de cada uno de los datos adquiridos por los diferentes sensores de la estación y los representa de manera permanente sobre el display LCD.

PROYECTO FINAL ESTACIÓN METEOROLÓGICA CON CONEXIÓN A INTERNET

```

117  /*----- init_display -----*/
118
119  static void init_display () {
120      /* LCD Module init */
121
122      LCD_Initialization();
123      LCD_Clear(Black);
124      GUI_Text(10,10, "ESTACION METEOROLOGICA", Green, Black);
125      GUI_Text(10,40, "Temperatura LM35: ", Red, Black);
126      GUI_Text(10,60, "Temperatura DS1621: ", Red, Black);
127      GUI_Text(10,80, "Temperatura BMP180: ", Red, Black);
128      GUI_Text(10,100, "Humedad: ", Red, Black);
129      GUI_Text(10,120, "Presion: ", Red, Black);
130      GUI_Text(10,140, "Umbral : ", Red, Black);
131      GUI_Text(10,160, "Viento : ", Red, Black);
132      GUI_Text(10,180, "Altitud: ", Red, Black);
133  }

```

Figura 47 - Definición de la función init\_display.

- **Upd\_display():** Se encarga de refrescar la información de las medidas realizadas por los diferentes elementos del proyecto y que se muestran sobre el display.

```

89  /*----- upd_display -----*/
90
91  static void upd_display () {
92      /* Update GLCD Module display text. */
93
94      sprintf((char*)text_disp, "%.1f C", temp_LM35);
95      GUI_Text(150,40, text_disp, White, Black);
96      sprintf((char*)text_disp, "%.1f C", temp_DS1621);
97      GUI_Text(170,60, text_disp, White, Black);
98      sprintf((char*)text_disp, "%.1f C", temp_BMP180);
99      GUI_Text(170,80, text_disp, White, Black);
100      sprintf((char*)text_disp, "%.1f %%", humedad);
101      GUI_Text(80,100, text_disp, White, Black);
102      sprintf((char*)text_disp, "%.2f hPa", presion);
103      GUI_Text(80,120, text_disp, White, Black);
104      sprintf((char*)text_disp, "%d C", umbral_temp);
105      GUI_Text(80,140, text_disp, White, Black);
106      sprintf((char*)text_disp, "%.2f m/s", vel_anemometro);
107      GUI_Text(80,160, text_disp, White, Black);
108      sprintf((char*)text_disp, "%.2f m", altitud);
109      GUI_Text(80,180, text_disp, White, Black);
110      GUI_Text(60,240, lcd_text[0], White, Red);
111      GUI_Text(52,260, lcd_text[1], White, Red);
112  }

```

Figura 48 - Definición de la función upd\_display.

- **Main():** Función principal. Se encarga de invocar las funciones de inicialización de todos los elementos necesarios para el correcto funcionamiento del proyecto, comprobar la temperatura del sistema y mantener el servidor operativo.

PROYECTO FINAL ESTACIÓN METEOROLÓGICA CON CONEXIÓN A INTERNET

```
243 /*----- MAIN -----*/
244
245 int main (void) {
246     //Inicialización general
247     init();
248     //Baudios UART
249     uart0_init(9600);
250     tx_cadena_UART0("----ESTACION METEOROLOGICA---\n\r");
251
252     //Inicialización TIMER principales T0-adq y tx datos, T3-Anemómetro
253     init_TIMER3();
254     init_TIMER0();
255
256     //Lectura datos calibración BMP180
257     check_communication();
258     read_calibration_data();
259
260     dhcp_tout = DHCP_TOUT;
261
262     while (1)
263     {
264         timer_poll();
265         main_TcpNet();
266         dhcp_check();
267         if (temp_DS1621 > (float)umbral_temp) //Se comprueba el umbral de temperatura para
268             set_ciclo_trabajo_PWM(temp_DS1621); //configurar la velocidad del ventilador.
269     }
270 }
271
272
```

Figura 49- Definición de la función main.

- **EINT1\_IRQHandler() y EINT2\_IRQHandler():** Funciones de atención a la interrupción externa configuradas para los pulsadores KEY1 y KEY2 respectivamente encargados de grabar y reproducir la señal de alarma.

```
169 /* -----
170 void EINT1_IRQHandler() {rec_ADC();}
171 void EINT2_IRQHandler() {play();}
172
```

Figura 50 - Definición de la función EINTX\_IRQHandler.

## HTTP\_CGI.c

Este archivo resulta muy importante ya que contiene las funciones que se emplean para actualizar la página web y recibir información desde ésta, es decir, controla las acciones del servidor http. Contiene las siguientes funciones:

- **Cgi\_func():** Se encarga de actualizar la información de las diferentes variables dinámicas de la página web. Las variables que se actualizan se configuran a través de los comandos que interpreta el archivo **.cgi** que contiene toda la información de la página web.

```

86  /*----- cgi_func -----
87
88  U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
89
90      U32 len = 0;
91      switch (env[0])
92      {
93          case 't': //Se actualizan los campos correspondientes a las temperatura
94          {
95              switch (env[2])
96              {
97                  case '1': //Temp sensor analógico LM35
98                      len= sprintf((char*)buf, (const char*)&env[4], temp_LM35);
99                      break;
100
101                  case '2': //Temp sensor DS1621
102                      len= sprintf((char*)buf, (const char*)&env[4], temp_DS1621);
103                      break;
104
105                  case '3': //Temp sensor BMP180
106                      len= sprintf((char*)buf, (const char*)&env[4], temp_BMP180);
107                      break;
108
109                  case '4': //Umbral
110                      len= sprintf((char*)buf, (const char*)&env[4], umbral_temp);
111                      break;
112              }
113          }
114          break;
115
116          case 'h': //Se actualiza la humedad
117              len= sprintf((char*)buf, (const char*)&env[2], humedad);
118              break;
119
120          case 'p': //Se actualiza la presión
121              len= sprintf((char*)buf, (const char*)&env[2], presion);
122              break;
123
124          case 'v': //Se actualiza la velocidad del anemómetro
125              len= sprintf((char*)buf, (const char*)&env[2], vel_anemometro);
126              break;
127
128          case 'a': //Se actualiza la altura
129              len= sprintf((char*)buf, (const char*)&env[2], altitud);
130              break;
131      }
132      return ((U16)len);
133  }
134

```

Figura 51 - Definición de la función **cgi\_func**.

- **Cgi\_progress\_var ():** Función que se encarga de tratar la información procedente de la página web a través del método **GET**. En este caso se utiliza para permitir al usuario modificar el valor del umbral de temperatura.

PROYECTO FINAL ESTACIÓN METEOROLÓGICA CON CONEXIÓN A INTERNET

```
55  /*----- cgi_process_var -----*/
56
57  void cgi_process_var (U8 *qs) {
58      /* This function is called by HTTP server to process the Query_String */
59      /* for the CGI Form GET method. It is called on SUBMIT from the browser. */
60      /* The Query_String is SPACE terminated. */
61      U8 *var;
62      var = (U8 *)alloc_mem (40);
63      do
64      {
65          /* Loop through all the parameters. */
66          qs = http_get_var (qs, var, 40);
67          /* Check the returned string, 'qs' now points to the next. */
68          if (var[0] != 0)
69          {
70              if (str_scomp (var, "umbral_t=") == __TRUE)
71                  sscanf ((const char *)&var[9], "%d", &umbral_temp);
72          }
73      } while (qs);
74
75      free_mem ((OS_FRAME *)var);
76  }
```

Figura 52 - Definición de la función cgi\_process\_var.

## *Index.cgi*

Contiene toda la información que se mostrará en la página web. Permite introducir una serie de comandos que caracterizan la forma de interpretar la línea que encabezan y dotan a la página web de un carácter dinámico que permite visualizar la evolución de las distintas medidas. Los comandos que admite y su función son los siguientes:

Command	Description
<b>i</b>	Commands the script interpreter to <b>include a file</b> from the virtual file system and to output the content on the web browser.
<b>t</b>	Commands that the <b>line of text</b> that follows is to be output to the browser.
<b>c</b>	Calls a C function from the <b>HTTP_CGI.c</b> file. The function may be followed by the line of text which is passed to <b>cgi_func()</b> as a pointer to an environment variable.
<b>#</b>	This is a comment line and is ignored by the interpreter.
<b>.</b>	Denotes the last script line.

Figura 53 - Información de comando para archivos .cgi.

Es importante destacar que las líneas que comiencen con el comando **c** serán tratadas por la función **cgi\_func()** del archivo HTTP\_CGI. Mediante este comando conseguimos que se actualicen los valores medidos por los diferentes sensores.

## *Web.inp*

En este archivo se enumeran los archivos que forman la página web , el directorio en el que se encuentran y el archivo de destino. Mediante **farm** se genera el archivo .c que es el contiene toda la información de la página web.

```
index.cgi, mono.gif to Web.c nopr root (Web)
```

Figura 54 - Contenido del archivo web.inp.



## Resultados obtenidos

Para acabar, se muestran los resultados obtenidos del funcionamiento del sistema final diseñado.

### *Página web*

Se ha generado una página web con credenciales de usuario y contraseña en la que se muestran los valores de cada uno de los sensores en una tabla de contenidos. Asimismo, se permite al usuario establecer el valor del umbral de la temperatura para la activación de el ventilador de refrigeración mediante un formulario que funciona mediante el método GET. Puesto que se trata de una página que en su mayor parte se emplea para visualizar datos, se ha optado por un diseño minimalista evitando fondos cargados e información disgregada.

ESTACIÓN METEOROLÓGICA

TEMPERATURA

LM35

DS1621

BMP180

Umbral

HUMEDAD

HIH4000

PRESIÓN

BMP180

VELOCIDAD DEL VIENTO

Anemómetro

ALTITUD

BMP180

18.5 C

25.5 C

3.2 C

20 C

74.7 %

911.5 hPa

0.00 m/s

883.44 m

Umbral de temperatura:

Enviar




Figura 55 - Aspecto de la página web.

## Display

Se adjuntas sendas capturas con la información mostrada en el LCD en los distintos estados del sistema. En la imagen de la izquierda se puede observar el sistema inicializándose (está tomando las primeras medidas tras un RESET o una desconexión) y en la imagen de la derecha, se puede apreciar la representación de dichas medidas una vez han sido tomadas.

Como ya se ha mencionado en apartados anteriores, el display tiene una frecuencia de actualización de 5s, no se ha considerado un frecuencia inferior puesto que las magnitudes medidas, como la temperatura o la presión, no tienden a variar de manera brusca en pequeños intervalos de tiempo.

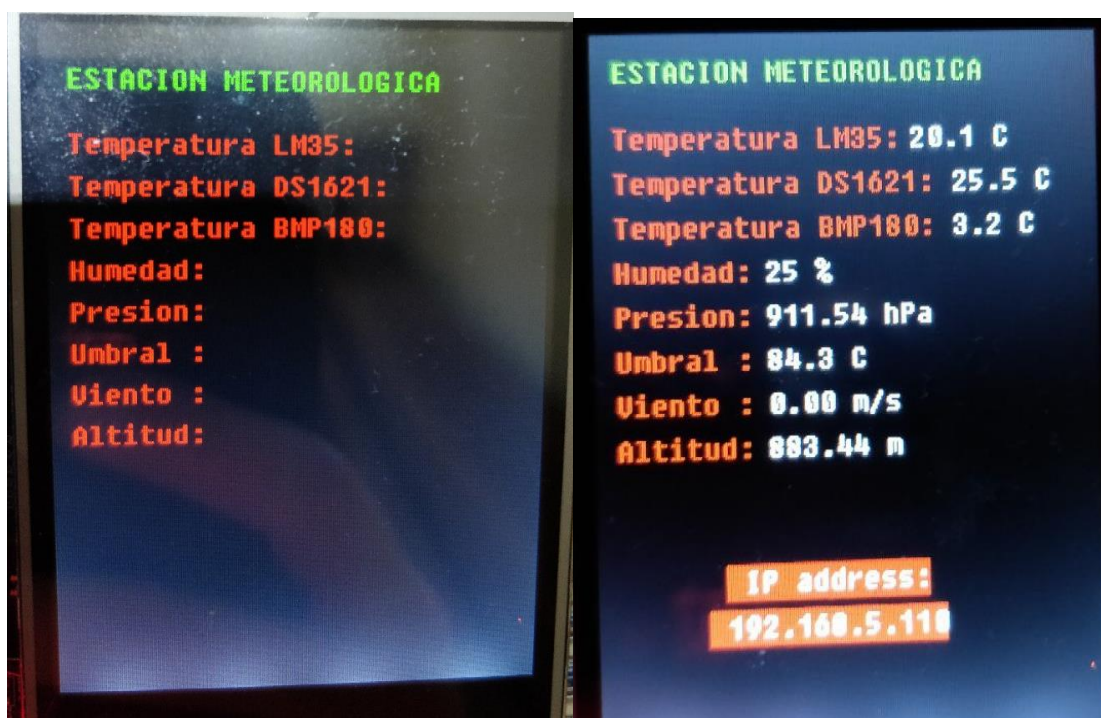


Figura 56 - Aspecto del display LCD.

## UART

Al igual que en los apartados anteriores, se procede a mostrar la representación de los datos medidos en el interfaz serie asíncrono:

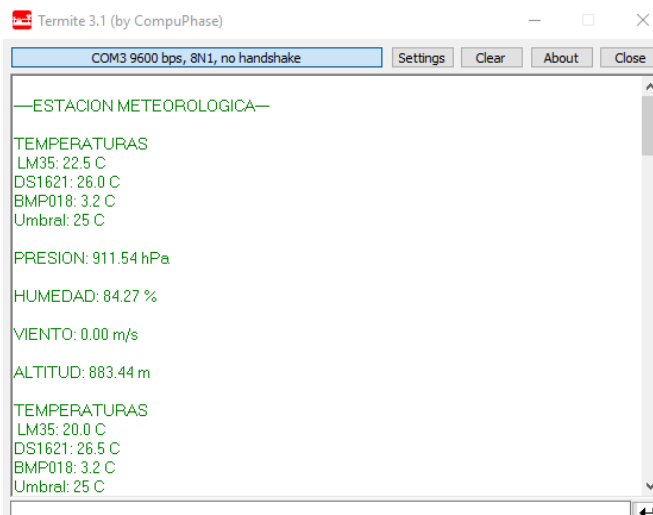


Figura 57 - Representación de datos por la UART.

## Conclusiones

El desarrollo de este proyecto ha sido muy ilustrativo y dinámico, ha permitido a los autores aprender un amplio rango de nuevos conocimientos aplicados a la electrónica discreta y ha generado una gran curiosidad por parte de estos sobre este “mundillo”.

Al igual que todo proyecto desarrollado a lo largo de un tiempo relativamente extenso, han surgido distintos problemas que han permitido aprender de los errores, mejorar la codificación y los sistemas de acondicionado y, además, obtener nuevas lecturas de lo que en principio se creía correcto. Por otro lado, se cree que es posible mejorar este proyecto añadiendo algunos módulos adicionales que permitiesen un mejor y más fácil control del sistema por parte del usuario, como puedan ser:

- **Módulo Wi-Fi.**
- **Módulo Bluetooth.**
- **Sistema RTO:** para controlar las distintas tareas de manera eficiente.
- **Módulo para tarjeta SD:** Para aumentar la memoria del dispositivo y permitir realizar páginas web con mayor contenido, almacenar grabaciones de mayor duración y calidad y, generar una pequeña BBDD con las medidas obtenidas por el sistema a lo largo del tiempo.