_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Registrador de datos de bajo coste y acceso remoto (Datalogger)

**Autor: Alfonso Carlos Larrea Broch**

**Tutor: Jorge Gosálbez Castillo**

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2015-2016

Valencia, 13 de Septiembre de 2016

## Resumen

El crecimiento de los sistemas embebidos o SBC (Single Board Computers) abre las posibilidades a sistemas que requieren cierta capacidad computacional y almacenamiento a un coste reducido. Esto permite la realización de multitud de proyectos con utilidades y capacidades muy diversas. En este proyecto, se pretende desarrollar un registrador de datos muticanal con una tasa de muestreo de 1kHz/canal, con capacidad de almacenamiento local y controlable remotamente mediante red para acceder a los datos registrados, consultar parámetros de configuración, reconfigurar el dispositivo... Alguna de estas plataformas de desarrollo son Arduino, Raspberry… las cuales ofrecen potencia y sencillez, así como diferentes shields para realizar diferentes aplicaciones. En este proyecto, se analizarán diferentes opciones (Arduino, Raspeberry, PIC…) y se implementará un datalogger basado en Arduino.

## Resum

El creiximent dels sistemes rebeguts o SBC (Single Board Computers) obri les possibilitats a sistemes que requerixen certa capacitat computacional i almagasenament a un cost reduit. Aço permet la realització de multitut de proyectes en utilitats i capacitats molt diverses. En este proyecte, se pretèn desenvolupar un registrador de senyes multicanal en una taxa de mostratge de 1KHZ/canal, en capacitat d'almagasenament local i controlable remotament mediant xarxes per accedir a les senyes registrades, consultar paràmetros de configuració, reconfigurar el dispositiu... Alguna d'estes plataformes de desenvolupament son Arduino, Raspberry… les quals oferixen potencia i simplicitat, aixina com diferents shields per a realitzar diferents aplicacions. En este proyecte, s'analitzaran diferents opcions (Arduino, Raspberry, PIC…) i se implementarà un datalogger basat en Arduino.

## Abstract

The growth of the absorbed systems or SBC (Single Board Computers) opens the possibilities for systems that need certain computational capacity and storage to a trim cost. This allows the accomplishment of project multitude with usefulness and very diverse capacities. The objective of this project is to develop an information recorder that allows multiple channels with a sampling rate of 1kHz/channel, with aptitude of local and controllable storage remotely accessible by means of network to access the registered information, to consult parameters of configuration, to re-form the device... Some of these development platforms are Arduino, Raspberry … which offer power and simplicity, as well as different shields to realize different applications. In this project, they will be analyzed different options (Arduino, Raspberry, PIC, …) and a datalogger using Arduino will be implemented.

# Agradecimientos

Al principiar este proyecto guardaba una visión bastante abstracta en cuanto al resultado final y la derivación del mismo, ya que al comienzo se tienen las nociones básicas para emprender el camino, mientras que el continuarlo requiere de motivación por descubrir, aprender e indagar, y como no de un tema que a uno le resulte candente. Desde el primer momento que inicie dicho proyecto mi incipiente curiosidad no hizo más que desvanecer la idea de que lo que realmente estaba haciendo era trabajo, ya que el mero hecho de investigar y finalmente dilucidar los problemas que me iban surgiendo era suficiente para henchirme de satisfacción e incitarme a continuar avanzando en el proyecto.

Darle las gracias a Jorge Gosálbez por guiarme en este camino de la mejor forma posible brindándome su ayuda y mostrando el mismo interés que yo en la realización del proyecto. Mostrar también mi especial agradecimiento a mi madre, a mi padre y a mi hermana, los cuales, cada uno por separado, me han ayudado a vislumbrar el camino de la manera más adecuada ofreciéndome su apoyo incondicional en cualquier situación. Dar las gracias a mis abuelos, los cuales sé que puedo contar con ellos cuando los necesite y que han mostrado un gran interés en el proceso de mis estudios, lo cual agradezco muchísimo. También agradecer a mi primo Jose Luis y a mi tío Jehova el apoyo que me dan cuando lo necesito y que sé que siempre me darán sin importar las vicisitudes que me encuentre en el camino.

Ahora termino una pequeña etapa de mi vida, aunque en mi opinión, como una persona a la que considero sabia y le tengo mucho aprecio me dijo una vez, el tener una carrera te da suficiencia, no sapiencia. Por lo que todavía me queda un gran camino por delante y mucho que aprender.

# Contenido


**CAPÍTULO 1. INTRODUCCIÓN ..... 3**

1.1 OBJETIVOS Y REQUISITOS DEL DISPOSITIVO ..... 3
1.2 ESTRUCTURA DEL PROYECTO ..... 4

**CAPÍTULO 2. ARDUINO Y OTRAS ALTERNATIVAS DE IMPLEMENTACIÓN ..... 5**

2.1 ¿QUÉ ES ARDUINO Y CÓMO FUNCIONA? ..... 5
2.2 PLACAS DE ARDUINO ..... 6
*2.2.1 Arduino UNO ..... 6*
*2.2.2 Arduino MEGA 2560 ..... 7*
2.3 SHIELDS ..... 8
*2.3.1 Ethernet shield ..... 9*
*2.3.2 RTC DS1307 (Reloj en tiempo real) ..... 10*
2.4 ¿PORQUE ELEGIR ARDUINO? ..... 11
2.5 MICROCONTROLADORES ATMEL ..... 12
2.6 ALTERNATIVAS A ARDUINO ..... 13
*2.6.1 Tarjetas Raspberry Pi ..... 13*
*2.6.2 Microcontroladores PIC (Microchip Technology) ..... 14*

**CAPÍTULO 3. HARDWARE DEL PROYECTO ..... 16**

3.1 ARDUINO MEGA 2560 ..... 16
3.2 W5500 ETHERNET SHIELD 2 ..... 16
*3.2.1 Programación Ethernet Shield 2 – Ejemplo ..... 17*
3.3 3.3 RTC (RELOJ EN TIEMPO REAL) ..... 19
*3.3.1 Programación RTC DS1307– Ejemplo ..... 20*
3.4 TARJETA MICROSD ..... 21
*3.4.1 Programación tarjeta SD – Ejemplo ..... 22*
3.5 SENSOR DE TEMPERATURA TMP36 ..... 23
*3.5.1 Programación Sensor Temperatura – Ejemplo ..... 25*

**CAPÍTULO 4. DESARROLLO DEL SOFTWARE ..... 27**

4.1 INCLUSIÓN DE LIBRERÍAS EXTERNAS ..... 27
4.2 ESTRUCTURA PRINCIPAL DEL CÓDIGO ..... 28
*4.2.1 Consideraciones de memoria en Arduino ..... 30*
*4.2.2 Función Web Server ..... 31*
*4.2.3 Función adquisición de datos ..... 36*
*4.2.4 Función Archivo de configuración ..... 39*

**CAPÍTULO 5. PRESUPUESTO Y COMPARATIVA CON DATALOGGERS COMERCIALES ..... 40**

5.1 PRESUPUESTO DEL PROYECTO ..... 40
5.2 REGISTRADORES COMERCIALES ..... 40

**CAPÍTULO 6. CONCLUSIONES Y LÍNEAS FUTURAS ..... 42**

**BIBLIOGRAFÍA ..... 45**

# Capítulo 1. Introducción

Se desea implementar un dispositivo cuyas capacidades fundamentales sean el registro de datos de magnitudes físicas, en nuestro caso y como ejemplo de temperaturas, y su posterior gestión de forma remota para evitar la necesidad de operar físicamente con dicho dispositivo.

Para realizar esta tarea de forma óptima utilizaremos los registradores de datos, los cuales están pensados para aplicaciones donde se desean recoger gran cantidad de datos a lo largo del tiempo. Este tiempo varía dependiendo de las características de la aplicación, así pues, pueden existir aplicaciones en las que es necesario registrar un dato cada segundo y en otras cada hora, por ello este tiempo es ajustable en nuestros registradores. Para las diferentes aplicaciones disponemos de diferentes registradores, desde uno o dos canales hasta cientos. [1]

Un registrador de datos es técnicamente denominado un "Datalogger" el cual es un sistema que almacena los datos recogidos a través de un sensor (temperatura, sonido, luminosidad, humedad, etc) en un dispositivo de almacenamiento (tarjeta SD, USB, computadora, etc) para su posterior gestión de forma correcta. En este caso concreto tiene la función añadida de que se puede acceder de forma remota al dispositivo, sin necesidad de desplazarse físicamente.

## *1.1 Objetivos y requisitos del dispositivo*

Tal y como se ha descrito anteriormente, se quería implementar un dispositivo que fuera capaz de adquirir datos y almacenarlos de forma local para gestionarlos posteriormente de forma remota. Atendiendo a esto, se establecieron los siguientes requisitos que debía cumplir el dispositivo:

- ***Adquisición de datos:*** Se debe de poder almacenar los datos recogidos por el sensor de forma automática y ordenada.

- ***Configuración de la frecuencia y variables de muestreo***: El usuario debe de poder modificar el tiempo que se tarda en recoger una muestra y el número de muestras que habrá en cada fichero.

- ***Conexión a Internet vía Ethernet***: Para poder acceder y gestionar de forma remota al dispositivo necesitaremos poder conectar el dispositivo vía cable Ethernet a un router cercano.

- ***Funcionamiento a largo plazo del dispositivo:*** El dispositivo está pensado para que se quede en un lugar por un período de tiempo prolongado, por ello deberemos de poder conectarlo a una fuente de alimentación de forma continua.

- ***Posibilidad de gestionar los datos a través de Internet:*** El dispositivo debe de poder actuar como un servidor, proporcionando al usuario acceso a una página web específicamente diseñada para su gestión.

- ***Estructura y formato de los ficheros de datos:*** Los datos recogidos deberán almacenarse en ficheros con un formato ".CSV", que aunque es poco eficiente resulta

muy versátil para el manejo de la información. Además deberán de seguir una estructura donde se especifique la Temperatura (ºC), tiempo de adquisición, etc.

## *1.2 Estructura del proyecto*

Basándonos en los requerimientos establecidos para el funcionamiento del dispositivo, hemos decidido utilizar la plataforma Arduino y una placa SBC de Arduino para su realización debido a su buena relación entre eficiencia y coste. Además presenta ventajas adicionales como que Arduino tiene un carácter <<*open source*>> (código abierto), es decir, que el hardware y software disponible puede ser manejado por cualquier usuario. De ésta forma podemos crear nuestras propias librerías o utilizar las de otros usuarios que las hayan puesto a disposición de los demás, consiguiendo de ésta forma hacer nuestros programas más funcionales y compactos.

Las placas de Arduino pueden ser programadas utilizando diferentes lenguajes de programación (Python, C/C++, Arduino IDE), en nuestro caso utilizaremos el lenguaje C/C++ debido a que el propio lenguaje de Arduino, por ejemplo, está diseñado principalmente para programas pequeños mientras que C/C++ nos brinda mayores posibilidades de programación aparte de ser ampliamente conocido.

El presente trabajo se compone de cinco capítulos, siendo el primero el de introducción. En el segundo capítulo se estudiarán diferentes alternativas hardware haciendo especial hincapié en la tecnología seleccionada, Arduino.

En el capítulo 3 comentaremos en detalle el hardware específico que utilizaremos en conjunción con su funcionamiento en Arduino.

- Arduino Mega AT2560
- Módulo Ethernet y SD W5500
- Sensor de Temperatura LM35
- Módulo de reloj en tiempo real (RTC)
- Tarjeta micro SD

En el capítulo 4 comentaremos el apartado del software utilizado, desde las librerías utilizadas para cada pieza hardware y la programación necesaria para su funcionamiento hasta las diversas técnicas utilizadas.

- Librería para el módulo Ethernet
- Librería para el módulo SD
- Librería para el RTC
- Librería para el uso de pines de la placa base (SPI)

En el capítulo 5 explicaremos detalladamente la programación de las diferentes funciones realizadas para la adquisición de datos, funcionamiento del servidor web y la gestión de archivos.

En el capítulo 6 plantearemos las conclusiones en cuanto a resultado final, coste de la aplicación final y comparaciones con otras parecidas disponibles en el mercado, eficiencia (pocos/muchos errores) y programación a los que hemos llegado.

# Capítulo 2. Arduino y otras alternativas de implementación

## *2.1 ¿Qué es Arduino y cómo funciona?*

Arduino es una plataforma de prototipos electrónica de código abierto basada en hardware y software flexibles y fáciles de utilizar, ésta compañía se dedica a la realización de placas de desarrollo de hardware y software compuestas por un microcontrolador y un entorno de desarrollo (IDE) conjuntados en un circuito impreso. Puede ser utilizado para desarrollar una ingente cantidad de proyectos interactivos los cuales pueden ser ejecutados desde la tarjeta Arduino ó un entorno en un ordenador.

Está basado en el uso de un microcontrolador, los cuales se caracterizan por el control y adquisición de datos a través de entradas y salidas analógicas/digitales, contando además con una interfaz de usuario. Existen diferentes clases de microcontroladores de los cuales hablaremos posteriormente, que varían dependiendo de las necesidades del proyecto, además hay una gran variedad de fabricantes y versiones disponibles.

Los sistemas microcontrolados tienen una gran cantidad de entradas y salidas donde podemos conectar, por ejemplo, sensores para el control de magnitudes físicas. También cuentan con áreas de prototipado para conexiones de otros dispositivos y la posibilidad de conectar otros módulos (shields) que amplian las características de funcionamiento de la placa, como por ejemplo el de almacenamiento en tarjetas SD, conexión Ethernet, Wi-Fi, etc. [2]

El funcionamiento de Arduino se puede resumir en tres instancias:

- *Interfaz de entrada*: Se encuentra directamente unida a los periféricos ó conectada a ellos a través de puertos. Principalmente se encarga de llevar la información al microcontrolador.
- *Procesado de datos*: El microcontrolador se encarga de procesar los datos que se van adquiriendo.
- *Interfaz de salida*: Se encarga de llevar la información procesada a los periféricos encargados de hacer el uso final de dichos datos, ya sea en otra placa que volverá a procesar los datos ó en la misma sacando los datos por pantalla (reloj digital), un altavoz (reproductor de música), realizando el encendido de un array de LED's, etc.

La placa de Arduino está conformada por diferentes partes, en las que cada una de ellas desarrolla una función específica:

- *Entradas*: Se trata de aquellos pines que se utilizan para captar las lecturas, pueden ser analógicos o digitales.
- *Salidas*: Son aquellos pines que se encargan de transmitir las señales, solo pueden ser analógicos.
- *Pines adicionales*: Se trata de los pines necesarios para la alimentación (Vin), los de conexión a tierra (GND), el voltaje que deseamos dar a nuestra placa (5V/3,3V), los necesarios para la conexión SPI (ICSP), los necesarios para la conexión I2C/TWI (SDA, SCL).

La posición de algunos pines varía dependiendo de la placa de Arduino que se utilice.

Arduino lleva a cabo unas comunicaciones denominadas comunicaciones serie, las cuales son una de las herramientas más importantes de las que dispone un microcontrolador. Con ella puede dejar de ser un chip aislado permitiéndole interactuar con cualquier dispositivo que también soporte una comunicación serial: ordenadores, sensores con conexión serie, equipos controlados por MIDI, smartphones, servidores u otros microcontroladores.

Dentro de las comunicaciones series podemos encontrar los siguientes protocolos:

- *UART (Recepción – Transmisión Asincrona Universal):* Se trata de uno de los protocolos serie más utilizados. Utiliza una línea de datos para transmitir y otra para recibir los datos.
- *SPI*: Un maestro envía la señal de reloj, y tras cada pulso de reloj se envía un bit al esclavo y se recibe un bit de éste. Los nombres de las señales son SCK para el reloj, MOSI (Master Out Slave In) y MISO (Master In Slave Out). Para controlar más de un esclavo es preciso utilizar SS (Slave Select).
- *I2C*: Protocolo síncrono que utiliza solamente dos cables, uno para el reloj (SCL) y otro para el dato (SDA). Maestro y esclavo envían información por el mismo cable, el cual es controlado por el maestro, que crea la señal de reloj. I2C no utiliza SS sino direccionamiento.

## *2.2 Placas de Arduino*

Arduino ofrece multitud de tarjetas basadas en microcontroladores AVR capaces de brindar al usuario la capacidad de programas de forma versátil sus aplicaciones. A continuación podemos ver las placas principales y algunos shields (módulos) que podemos añadir a dichas tarjetas como extensión.

### *2.2.1 Arduino UNO*

Se trata de la versión de referencia de Arduino, que actualmente ha evolucionado en nuevas versiones. Sin lugar a duda es una de las mejores placas para comenzar en la electrónica y programación.

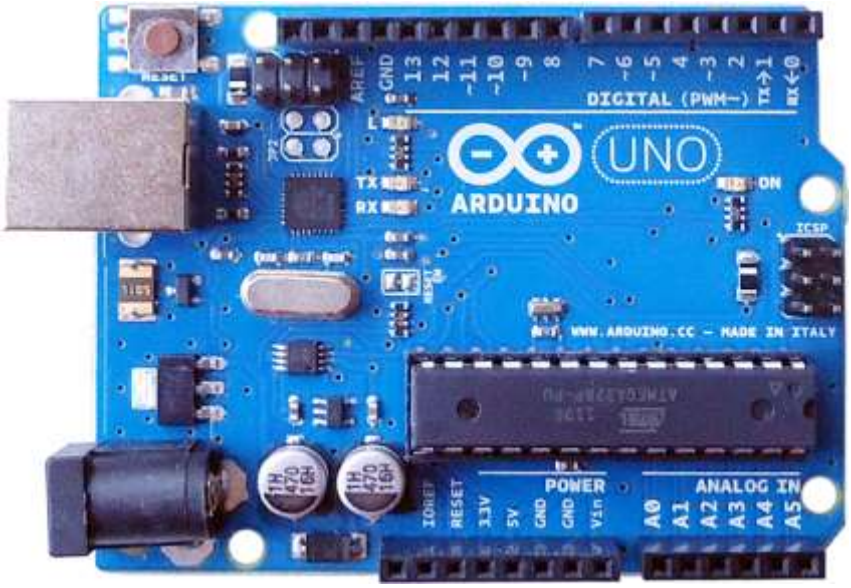


*Figura 1. Figura de Arduino 1*

Características técnicas [3]:

- Microcontrolador: ATMEL ATmega328P
- Voltaje necesario: 5V
- Voltaje de entrada: 7-12V
- Voltaje de entrada (límite): 6-20V
- Pines I/O Digitales: 14
- Pines PWM I/O Digitales: 6
- Pines Analógicos de entrada: 6
- Corriente DC por I/O pin: 20 mA
- Corriente DC por 3.3V pin: 50 mA
- Memoria Flash: 32 kB
- SRAM: 2 kB
- EEPROM: 1 kB
- Velocidad del reloj: 16 Mhz
- Programación mediante conexión USB
- Soporta dos tipos de interrupciones
- Soporto protocolos SPI e I2C
- Longitud: 68.6 mm
- Anchura: 53.4 mm
- Peso: 25g

El precio de ésta placa es de aproximadamente 25€, lo que es significativamente económico para todas las posibilidades que ofrece.

### *2.2.2 Arduino MEGA 2560*

Se trata de una versión de Arduino más actual que la UNO pensada para la realización de proyectos más complejos que requieren de una mayor memoria SRAM y más pines I/O.

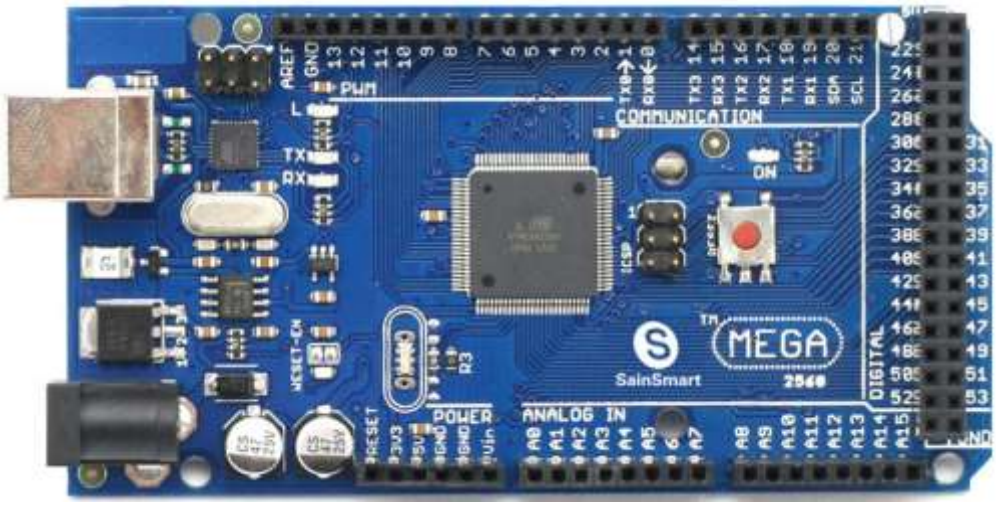


*Figura 2. Figura de Arduino MEGA*

Características técnicas [4]:

- Microcontrolador: ATMEL ATmega2560
- Voltaje necesario: 5V
- Voltaje de entrada: 7-12V
- Voltaje de entrada (límite): 6-20V
- Pines I/O Digitales: 54 (15 son salidas PWM)
- Pines Analógicos de entrada: 16
- Corriente DC por I/O pin: 20 mA
- Corriente DC por 3.3V pin: 50 mA
- Memoria Flash: 256 kB
- SRAM: 8 kB
- EEPROM: 4 kB
- Velocidad del reloj: 16 Mhz
- Programación mediante conexión USB
- Soporta seis tipos de interrupciones
- Soporto protocolos SPI e I2C
- Longitud: 101.52 mm
- Anchura: 53.3 mm
- Peso: 37g

El precio de ésta placa es de aproximadamente 35€ + VAT.

## *2.3 Shields*

Aparte del uso de las tarjetas, la plataforma Arduino también pone a nuestra disposición una serie de Shields (módulos) que podemos conectar a nuestra tarjeta para incrementar el número de funcionalidades disponibles.

A continuación podemos ver un ejemplo de conexión de un Shield el cual dota de conexión Ethernet y lector de tarjetas micro SD:

*Figura 3. Figura de un Shield para Arduino*

La conexión de los Shields se realiza en cascada verticalmente, de forma que encajan perfectamente con los pines de nuestra placa Arduino, quedando apilados uno encima del otro. Teóricamente perderíamos algunos pines de entrada/salida en nuestra tarjeta, pero esto no tendría mucha importancia ya que a cambio obtendríamos una gran cantidad de funcionalidades más. De todas formas, en la documentación del shield que adquiramos podemos comprobar si inhabilita algunas de las I/O ó si usa un bus y que requisitos tiene para su utilización.

La comunicación entre los Shields y la tarjeta SBC se consigue a través del protocolo SPI (Serial Peripheral Interface Bus). [5]

### *2.3.1 Ethernet shield*

El módulo de expansión Ethernet aporta a nuestro Arduino la funcionalidad de conectarse a una red local con unos sencillos comandos de configuración (IP, MAC). Además incluye una ranura que permite insertar una tarjeta SD para almacenaje para la escritura y lectura de datos.

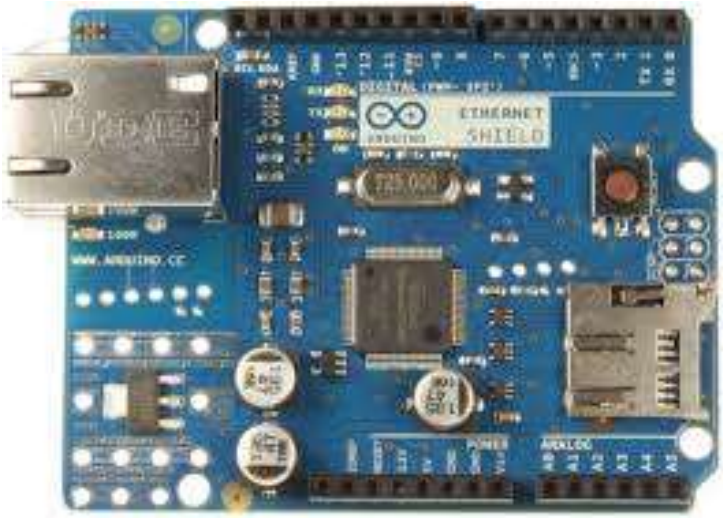


*Figura 4. Shield Ethernet*

Caraterísticas técnicas [6]:

- Requiere una placa de Arduino (UNO, Mega, etc)
- Voltaje de operación : 5V
- Utiliza el controlador W5100 con buffer interno de 16k
- Velocidad de conexión: 10/100Mb
- Conexión con el Arduino mediante los puertos SPI
- Zócalo para tarjeta micro SD
- Conector RJ45
- Opcion de alimentación mediante cable Ethernet (PoE)
- Soporta UDP y hasta cuatro conexiones en TCP

### *2.3.2 RTC DS1307 (Reloj en tiempo real)*

El RTC DS1307, es un reloj en tiempo real que utiliza la comunicación I2C. Una vez conectado a nuestro Arduino y establecida la hora y fecha inicial. El dispositivo mostrará el tiempo en segundos, minutos, horas, días, meses y años, y la fecha en formato de 24 horas o de 12 horas con modos AM/PM.

El dispositivo se alimenta utilizando una pila, la cual se inserta en un zócalo de forma que aunque haya un corte de electricidad el reloj cuenta con una batería por lo que continuará funcionando y no perderá la fecha y tiempo actual, todo gracias al chip DS1307 con el que funciona. [7]

Además, éste módulo posee una memoria AT24c32 que permite almacenar una cantidad mayor de datos.

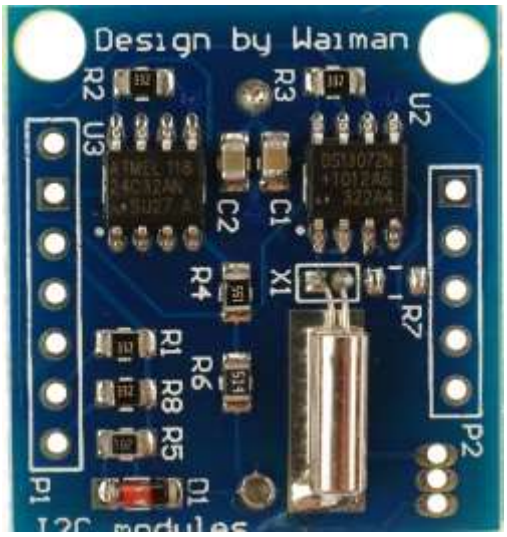


*Figura 5. Módulo RTC*

La conexión a módulo I2C se realiza conectando cables de colores del RTC a sus respectivas entradas en el Arduino. Hay que tener en cuenta que podemos escoger los cables del color que queramos, sin embargo por convenio podemos utilizar los siguientes:

- GND → Negro
- VCC → Rojo
- SDA → Naranja
- SCL → Azul

Características Técnicas:

- Voltaje de entrada (VCC): 4.5 ~ 5.5V
- El RTC cuenta el tiempo y la fecha hasta el año 2100
- 56-byte, utilización adicional de batería, RAM no volátil (NV) para almacenamiento
- Consume menos de 500nA en caso de utilización de batería
- Temperatura a la que funciona: -40°C - +85°C
- Dimensiones: 28 x 25 x 8.4 mm

## *2.4 ¿Porque elegir Arduino?*

En reconocimiento a la sencillez y gran accesibilidad que ofrece al usuario, Arduino se ha utilizado en miles de diferentes proyectos y aplicaciones. Como ya hemos comentado, el software de Arduino es de fácil uso para principiantes y lo suficientemente flexibles como para ser manejado por usuarios experimentados. Funciona tanto en Mac, Windows y Linux. Diseñadores y arquitectos pueden crear prototipos interactivos, profesores y alumnos pueden crear dispositivos científicos de bajo coste, para iniciarse en la programación y la robótica. Arduino es una herramienta clave para el aprendizaje de cosas nuevas relacionadas con la información, donde cualquier persona puede empezar a trastear con ello, desde niños hasta programadores.

Hay multitud de controladores más y plataformas de microcontroladores disponibles para la computación física. Parallaz Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, y muchos más que ofrecen un funcionamiento parecido. Todas estas herramientas cogen toda la complejidad de los microcontroladores y la convierten en un paquete sencillo de utilizar. Arduino también simplifica el proceso de trabajar con microcontroladores, pero ofrece algunas ventajas para profesores, estudiantes, y aficionados a la electrónica que estén interesados.

Existen numerosas razones por las que deberíamos escoger la plataforma Arduino para el desarrollo de nuestro proyecto.

En primer lugar, *Arduino es sencillo de programa,* ya que ésta plataforma nos ofrece un entorno de desarrollo integrado (IDE) con funciones preestablecidas que reducen la lógica a la lectura de entradas, control de tiempos y salidas de una manera intuitiva, convirtiéndolo en una herramienta perfecta para aquellos que quieran iniciarse en la electrónica y luego, probar cosas más difíciles.

En segundo lugar, se trata de su *carácter Open Source,* el cual nos permite acceder a todo aspecto del funcionamiento circuital y algorítmico de las placas además de que se nos proveen todos los archivos Eagle, diagramas y componentes para que nosotros mismos creemos nuestra propia versión de Arduino.

En tercer lugar, *la existencia de librerías*, las cuales facilitan enormemente el desarrollo del software ya que existen librerías para cualquier elemento externo que se pueda conectar a nuestro Arduino. La existencia de estas librerías supone un uso óptimo de los periféricos, pues vienen provistas de multitud de funciones para que la programación y la realización del proyecto sea más claro y sencillo.

En cuarto lugar, *encontramos multitud de información de ayuda en la red*, el mismo entorno de desarrollo de Arduino nos provee de una gran cantidad de ejemplos para realización de una gran cantidad de aplicaciones sencillas para conseguir entender el concepto de su desarrollo. Además podemos encontrar tutoriales e información a lo largo de la red.

En quinto lugar, *encontramos diferentes tipos de placas en base a nuestras necesidades*, dependiendo del proyecto que tengamos en mente podemos escoger una u otra placa.

También, la existencia de los Shields, que pueden ser acoplados a las tarjetas de forma sencilla, aumentando considerablemente el rango de aplicaciones disponibles. Además Arduino también es compatible con infinidad de periféricos de otras marcas como Xbee, Teclados, LCD, sensores digitales, tarjetas SD, etc. [8]

## *2.5 Microcontroladores ATMEL*

ATMEL es una compañía de semiconductores que fabrica microcontroladores, dispositivos de radiofrecuencia, memorias EEPROM y Flash, ASICs, WiMAX y muchas otras. En lo referente a la plataforma Arduino, cuyo núcleo es un microcontrolador de la línea ATMEL, explícitamente aquellos de la línea AVR diseñados con arquitectura Harvard tipo RISC (Reduced Instruction Set Computing) de 8 bits. Podemos dividir los AVR en los siguientes grupos:

- *ATxmega*: Procesadores muy potentes con 16 a 284 kB de memoria flash programable con un amplio conjunto de periféricos.
- *ATmega*: Microcontroladores AVR grandes con 4 a 256 kB de memoria flash programable con un set limitado de periféricos.

- *ATtiny*: Pequeños microcontroladores AVR con 0,5 a 8 kB de memoria flash programable y con un set limitado de periféricos.
- *AT90USB*: ATmega integrado con controlador USB.
- *AT90CAN*: ATmega con controlador de bus CAN.
- *AT90S*: Actualmente obsoletos.

Los microcontroladores AVR son relativamente rápidos en comparación con otros microcontroladores de 8-bit debido a que las instrucciones en la memoria de programa son ejecutadas con una segmentación de dos etapas. Mientras una instrucción está siendo ejecutada, la siguiente es pre-capturada de la memoria de programa, de forma que en un ciclo de reloj pueden leerse dos registros.

Estos microcontroladores cuentan con tres tipos de memoria diferente, en primer lugar tenemos la memoria SRAM la cual se utiliza para datos, en segundo lugar la memoria de programa Flash y finalmente la memoria EEPROM, todas On-chip. Por lo que al contrario que el típico microprocesador, contamos con un sistema de cómputo completo formado por una CPU junto con su memoria, sin embargo tendrá una mayor limitación a la hora de realizar las tareas debido a que se trata de un microcontrolador ya integrado.

El microcontrolador por ser un sistema digital programable, necesita de un código de programa o firmware que incluya las instrucciones necesarias para realizar el control del sistema embedido. En el caso de los AVR encontramos compiladores de lenguaje C/C++. Basic, dentro de los cuales podemos encontrar IDEs como AVRstudio y BASCOM.

En cuanto al interfaz de programación, dentro de la familia AVR, el chip dispone de un periférico específico para la programación de su memoria, el puerto ISP, el cual es un puerto serial formado por 3 pines del microcontrolador, son estos pines los que se conectan a un programador (por puerto paralelo, USB, serial, etc) y este a un puerto del pc, para realizar el "quemado" del chip. [9]

Algunas características adicionales son los circuitos internos de reloj, bloques que proveen la señal de sincronización, frecuencia o velocidad a la cual el microcontrolador ejecutará las instrucciones del programa.

## *2.6 Alternativas a Arduino*

### *2.6.1 Tarjetas Raspberry Pi*

Se trata de un ordenador de placa única (SBC) de bajo coste desarrollada por la Fundación Raspberry Pi. Esta plataforma ofrece funcionalidad avanzada superior a la de Arduino.

El diseño incluye un procesador Broadcom BCM2835 que contiene un procesador central ARM1176JZF-S a 700Mhz (puede llegar hasta 1 Ghz) y un procesador gráfico VideoCore IV y 512 MB de memoria RAM. El diseño no incluye un disco duro ni unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente. Tampoco incluye fuente de alimentación ni carcasa.

*Figura 6. Placa Raspberry Pi*

Las placas más modernas cuentan con hasta 4 puertos USB para conectar teclado y ratón, un conector HDMI y hasta una conexión Ethernet para acceso a la red vía cable. Entre los sistemas operativos disponibles se encuentran:

- Raspbian
- Arch Linux
- RaspBMC
- Pidora
- OpenELEC

En lo referente a precios Raspberry Pi es bastante económico. La última placa disponible (modelo B+) tiene un precio de 30.95€. Las placas anteriores son mucho más económicas e incluso podemos encontrar kits de trabajo por menos de 60€. [10]

En conclusión, podemos decir que se trata de una plataforma sencilla y a la vez potente cuyos equipos podemos conseguir por precios muy asequibles. Convirtiéndolo en una buena opción tanto para usuarios novatos como experimentados.

En cuanto a costes, las tarjetas de Raspberry Pi oscilan entre los 30€ - 90€. [11]

### *2.6.2 Microcontroladores PIC (Microchip Technology)*

La empresa Microchip Technology fabrica tarjetas basadas en microcontroladores PIC similares a las utilizadas en Arduino. Cabe decir que los microcontroladores PIC son la competencia directa de los ATMEL AVR debido a sus precios tan similares.


*Figura 7. Microcontrolador PIC [12]*

Algunas ventajas de los microcontroladores PIC son las siguientes:

- Eficiencia de código: Permiten una gran compactación de los programas
- Rapidez de ejecución: Utiliza una frecuencia de 20Mhz
- Seguridad en acceso por la separación de memoria de datos y programa
- Juego reducido de instrucciones y de fácil aprendizaje
- Compatibilidad de pines y código entre dispositivos de la misma familia o sin reducción de prestaciones internas.
- Gran variedad de versiones en distintos encapsulados.
- Herramientas de desarrollo software y hardware abundantes y de bajo coste

Todos ellos utilizan una arquitectura Harvard, lo que quiere decir que su memoria de programa está conectada a la CPU por más de 8 líneas. Hay microcontroladores de 12, 14 y 16 bits y desde 2007 de 32 bits, dependiendo del nivel de procesamiento de información que más se ajuste al proyecto. [12]

Las familias de controladores PIC:

- PIC10F20x: 4 dispositivos
- PIC12CXXX/16FXXX: 12/14 bits tenemos 8 dispositivos.
- PIC16C5X: 12 bits 9 dispositivos
- PIC16CXXX/16FXXX: 14 bits tenemos 74 dispositivos.
- PIC18CXXX/18FXXX: 16 bits 82 dispositivos

En lo relevante a la programación, utiliza ICSP o LVP para programar el PIC en el circuito destino. Para la ICSP se usan los pines RB6 y RB7 como reloj y datos y el MCLR para activar el modo programación aplicando un voltaje de 13 voltios.

Desde un punto de vista de estructura general los microcontroladores PIC y AVR son iguales, sin embargo, difieren en indicadores como: Lenguaje de programación, IDE, interfaces para la programación, reloj interno, voltaje de alimentación, potencia, costo, etc.

Al contrario que AVR, los diferentes IDE y compiladores C/C++ y Basic no son completamente gratuitas y tienen su costo, entre los más usados están: MPLAB, PICSIMULATOR, PICBASIC, y más.

En cuanto a costes, el microcontrolador PIC de la gama más alta (PIC32MZZ048ECM1) tiene un coste de unos 9.52$, mientras que el microcontrolador ATMEL AVR de mayor gama (ATMEGA32L-8AU) tiene un precio de 8.48€. Por lo que son precios muy parecidos. [13].

# Capítulo 3. Hardware del proyecto

En este capítulo focalizaremos los dispositivos físicos específicos que hemos utilizado para nuestro proyecto <<Registrador de datos remoto (Datalogger)>>, las funcionalidades de las que provee a nuestra aplicación y todo lo necesario para su correcto funcionamiento en el entorno de desarrollo de Arduino (IDE), como los pines I/O que hemos utilizado para cada dispositivo, etc.

En líneas generales, como ya se explicó en la introducción, nuestro proyecto requiere de las siguientes funcionalidades:

- Lectura de datos adquiridos por un sensor y su posterior guardado en un dispositivo de almacenamiento
- Conexión permanente a una red de área local
- Constancia del tiempo y fecha actuales

Para cumplimentar éstos requisitos utilizamos los siguientes dispositivos:

- Arduino Mega AT2670
- W5500 Ethernet Shield 2
- Sensor de temperatura LM35
- Groove-RTC clock
- Tarjeta micro SD
- Cable Ethernet

Si recordamos, en el capítulo 2 se han explicado de forma general algunos de éstos dispositivos, sin embargo, a continuación, se explicarán todos los dispositivos utilizados en mayor detalle.

## *3.1 Arduino Mega 2560*

Hemos utilizado éste modelo de Arduino ya que es el más potente que ofrece. Sin embargo, para el tipo de proyecto que estamos realizando no es estrictamente necesario usar éste modelo, con uno de características medias hubiera sido suficiente. Aunque el modelo UNO seguramente no hubiera sido viable debido a sus altas limitaciones de memoria para proyectos medianos y grandes.

## *3.2 W5500 Ethernet shield 2*

El W5500 permite dotar a nuestros proyectos de acceso a una conexión de internet, utilizando un chip que tiene embedido: TCP/IP apilados, 10/100 Ethernet MAC y PHY. El Shield también tiene dos conectores Groove y una ranura para una tarjeta microSD, de forma que pueda soportar proyectos que requieran una gran cantidad de datos.

*Figura 8. Módulo Ethernet Shield 2 [14]*

El Ethernet Shield 2 cuenta con las funcionalidades de conexión de red mediante Ethernet y de acceso a un dispositivo de almacenamiento como la tarjeta microSD, sin embargo ambas funcionalidades no pueden actuar simultáneamente.

El pin 10 del Shield es el CS (Chip Select) del W5500 que nos provee de una conexión Ethernet, mientras que el pin 4 es el CS correspondiente al lector de la tarjeta microSD, estos pines no pueden utilizarse como pines de entrada/salida.

El Arduino se comunica tanto con el W5500 como con la tarjeta SD utilizando el bus SPI (a través de la cabecera ICSP). Esto sucede en los pines digitales 10, 11, 12 y 13 en el Arduino UNO y en los pines 50, 51 y 52 en el Mega. En el Mega el pin 53 no se utiliza ni para seleccionar el W5500 ni la tarjeta SD, en cambio se guarda como una salida ya que de no ser así la interfaz SPI no funcionará. [14]

Características técnicas:

- Soporta IEEE802.3af
- Voltaje de entrada: 36V-57V
- Protección frente a sobrecarga
- Voltaje de salida: 12V
- Convertidor DC/DC de alta eficiencia
- Aislamiento de 1500V (de entrada a salida)

### *3.2.1 Programación Ethernet Shield 2 – Ejemplo*

Para que nuestro módulo Ethernet aporte a nuestro Arduino las funcionalidades deseadas debemos de configurar unos cuantos parámetros y escribir unos cuantos comandos. Un ejemplo de cómo configurar nuestro proyecto para conseguir que funciones como un servidor web sería:

```
#include <SPI.h>
#include <Ethernet2.h>

//Insertar la dirección IP de nuestro servidor a crear
//Insertar la MAC de nuestro módulo Ethernet
byte mac[] = {
 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 177);

// El puerto 80 es el utilizado para HTTP
EthernetServer server(80);

void setup() {
 // Iniciamos la comunicación serie y esperamos a que se inicie el
puerto
 Serial.begin(9600);
 while (!Serial) {
 ; //Esperamos a que el puerto se conecte, solo para placas Leonardo
 }


 // Iniciamos la conexión Ethernet
 Ethernet.begin(mac, ip);
 server.begin();

}

void loop() {

 //Funcionalidad WebServer

 // Escuchamos a clientes nuevos
 EthernetClient client = server.available();
 if (client) {
 Serial.println("new client");
 // una petición http finaliza con un espacio en blanco
 boolean currentLineIsBlank = true;
 while (client.connected()) {
 if (client.available()) {

 //Lo que queremos que ocurra en el servidor web

 }
 }
 // damos tiempo al explorador web para que reciba la información
 delay(1);
 // cerramos la conexión
 client.stop();
 Serial.println("client disconnected");
 }


}
```

Primero deberemos de incluir las librería <SPI.h> ya que necesitaremos tener activa la comunicación entre nuestra tarjeta de Arduino y nuestro Ethernet Shield. También será necesaria la librería <Ethernet2.h>, la cual es específica para aquellos módulos Ethernet que utilicen el chip W5500. Ésta librería aporta todas las funcionalidades de conexión de área local.

Como el ejemplo es de un servidor web, será necesario dotar a nuestro servidor de una dirección IP con el comando *IPAdress ip ()* que permite a los clientes realizar peticiones y una MAC con el comando *byte mac [] = {}* para su identificación. La dirección IP deberá de ser cualquiera dentro de nuestro rango de direcciones, el cual podemos comprobar en la consola de nuestro ordenador, escribiendo el comando *ipconfig* y chequeando la línea *Dirección de IPv4* y *Mascara de Subred.*

Dependiendo del producto, la MAC la podemos encontrar en una etiqueta pegada en la parte trasera de nuestro módulo. Es importante tener en cuenta que cada dispositivo debe tener una MAC única, es decir no podemos tener dos dispositivos con la misma MAC funcionando simultáneamente.

El resto de comandos están explicados mediante comentarios en el ejemplo.

## 3.3 *3.3 RTC (Reloj en tiempo real)*

El módulo RTC está basado en el chip de reloj DS1307, el cual soporta el protocolo I2C. Utiliza una batería de Litio (CR1225) de 3V. El reloj/calendario provee segundos, minutos, horas, dia, fecha, mes y año. La finalización de los meses se ajusta automáticamente para aquellos que meses que tienes menos de 31 días. El reloj trabaja tanto en el formato de 24 horas como el de 12 horas con indicadores AM/PM.


*Figura 9. Módulo RTC DS1307 [15]*

Características técnicas:

- Tamaño PCB: 2.0 cm * 4.0 cm
- Interfaz: 2.0 mm pitch pin header
- Estructura IO: SCL, SDA, VCC, GND
- ROHS: Sí
- VCC: 3.3 – 5.5V
- Logic High Level Input: 2.2~VCC+0.3V
- Logic Low Level Input:-0.3~+0.8V
- Voltaje de la batería: 2.0 ~ 3.5V

### *3.3.1 Programación RTC DS1307– Ejemplo*

Un ejemplo de cómo podemos iniciar nuestro reloj en tiempo real para que nos proporcione la fecha y hora correctas es el siguiente:

```
// Funcionalidad de ajuste de tiempo y fecha usando un DS1207 RTC
conectado via I2C y Wire lib

#include <Wire.h>
#include "RTClib.h"

RTC_DS1307 rtc;

void setup () {
 Serial.begin(9600);

 Wire.begin();
 rtc.begin();

 if (! rtc.isrunning()) {
 Serial.println("El RTC no esta funcionando!");
 // Ajustamos el tiempo y fecha a la de la ultima compilacion del
programa
 rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
 }
}

void loop ()
{
 DateTime now = rtc.now();

 //Sacamos por el monitor serie la fecha y tiempo actuales cada segundo
 Serial.print(now.year(), DEC);
 Serial.print('/');
 Serial.print(now.month(), DEC);
 Serial.print('/');
 Serial.print(now.day(), DEC);
 Serial.print(' ');
 Serial.print(now.hour(), DEC);
 Serial.print(':');
 Serial.print(now.minute(), DEC);
 Serial.print(':');
 Serial.print(now.second(), DEC);
 Serial.println();
```

```
}
```

En primer lugar debemos de incluir la librería <Wire.h> ya que es la que se encarga de controlar la comunicación I2C entre la placa de Arduino y el reloj. También deberemos de incluir la librería <RTClib.h> ya que es la que nos proporciona todas las funcionalidades del reloj y sin ella no funcionaría.

En segundo lugar, comprobaremos si el reloj ya está en marcha o hace falta iniciarse, mediante el comando *rtc.isrunning().* En caso de no estar en marcha lo inicia, y si ya está iniciado ajusta el tiempo y la fecha mediante el comando *rtc.adjust().*

En tercer lugar, definimos una variable a la que llamaremos *now* que contendrá todos los datos del tiempo y fecha, de forma que accediendo adecuadamente podremos conocer el año, mes, hora, minuto y segundo.

## *3.4 Tarjeta MicroSD*

Una vez hayamos leído los datos de temperatura del sensor, tendremos que guardarlos en algún dispositivo de almacenamiento para su posterior uso. Para nuestro proyecto hemos decidido utilizar una tarjeta micro SD, ya que el shield que hemos utilizado para la conexión Ethernet también permite insertar una tarjeta SD.

En cuanto a espacio de alamacenamiento la tarjeta micro SD cubre perfectamente nuestras necesidades. Hemos elegido la tarjeta micro SD Trascend de 2GB.


*Figura 10. Tarjeta MicroSD 2GB*

Una capacidad de 2GB suele ser suficiente para un registrador de datos. La memoria de nuestro dispositivo determinará el número de registros que pueden llevarse a cabo. Como esto puede convertirse en una limitación llegado un determinado momento, existen tarjetas con muchísima más capacidad, como por ejemplo las de 32GB.

Características técnicas:

- Velocidadad de hasta 10MB/s de transferencia de datos
- Velocidad de hasta 3MB/S en escritura de datos
- Tipo: Micro Secure Digital (MicroSD)

- Tipo de memoria interna: Flash
- Temperatura de funcionamiento: -25-85°C
- Capacidad: 2GB

### *3.4.1 Programación tarjeta SD – Ejemplo*

Para que nuestro Arduino reconozca la tarjeta SD y nos permita trabajar con ella, deberemos de escribir unos cuantos comandos. Un ejemplo de ello es el siguiente trozo de código que permite la escritura y lectura de datos dentro de la tarjeta SD:

```
//Ejemplo para leer y escribir datos en un archivo dentro de la tarjeta
SD

#define SD_CS_PIN SS
#include <SPI.h>
#include "SdFat.h"
SdFat SD;

File myFile;

void setup()
{
 // Open serial communications and wait for port to open:
 Serial.begin(9600);

 // Wait for USB Serial
 while (!Serial) {
 SysCall::yield();
 }

 Serial.print("Initializing SD card...");

 //Definimos el pin 53 como SS ya que usamos el Arduino Mega
 //Definimos el pin como una salida, de no ser asi las funciones de la
libreria SD no funcionarian
 pinMode(53, OUTPUT);

 //Iniciamos la tarjeta SD
 if (!SD.begin(SD_CS_PIN)) {
 Serial.println("initialization failed!");
 return;
 }
 Serial.println("initialization done.");

 //Abrimos un archivo que se encuentra dentro de la tarjeta SD
 myFile = SD.open("test.txt", FILE_WRITE);

 // Si el archivo se abrio correctamente, escribimos en el:
 if (myFile) {
 Serial.print("Escribiendo en el archivo test.txt...");
 myFile.println("Hola, esto es una prueba");
 // cerramos el archivo
 myFile.close();
 Serial.println("Archivo cerrado");
 } else {
```

```
 // Si el archivo no se abrio, saca un error por pantalla
 Serial.println("Error al abrir el archivo test.txt");
 }

 // Abrimos el archivo para leerlo:
 myFile = SD.open("test.txt");
 if (myFile) {
 Serial.println("test.txt:");

 // Leemos la información del archivo hasta que ya no quede más:
 while (myFile.available()) {
 Serial.write(myFile.read());
 }
 // cerramos el archivo:
 myFile.close();
 } else {
 // Si el archivo no se abrio, sacamos un error por pantalla:
 Serial.println("Error abriendo el archivo test.txt");
 }
}

void loop()
{
 // Despues del setup no ocurre nada
}
```

Es necesario incluir la librería <*SPI.h*> para activar la comunicación entre la tarjeta del Arduino y el módulo externo. También deberemos de incluir la librería <SdFat.h> la cual nos proporciona una gran cantidad de funciones para trabajar con la tarjeta SD. No se hace estrictamente necesario, pero dependiendo de la librería que nos descarguemos, podremos añadir las funcionalidades de la librería <SD.h> que nos aporta funciones para la tarjeta SD mediante el comando *SdFat SD* aparte de las que ya tengamos con la lib. SdFat.h.

La razón por la que utilizamos la librería SdFat y no exclusivamente la SD es porque nos provee de un mayor número de funciones, de forma que podemos sacarle un mayor rendimiento a lo que podemos realizar. Mientras que SD.h está diseñada para que sea fácil de entender y manejable, pensada para proyectos en los que no necesites un gran número de funciones y dirigido a usuarios que están empezando a dominar los conceptos básicos de programación.

Es importante no olvidarse de que no se puede tener un más de un archivo abierto simultáneamente, por lo que una vez se ha acabado de utilizar éste debe de cerrarse con el comando *NombreArchivo.close()* de forma que además de poder abrir un archivo nuevo se garantice que los cambios realizados en dicho archivo se han realizado con éxito.

Los comandos utilizados para el inicio de la tarjeta SD, lectura/escritura de ficheros, etc se explican en el ejemplo mediante comentarios.

## *3.5 Sensor de temperatura TMP36*

Para realizar la lectura de la temperatura necesitaremos un sensor que deberemos conectar a nuestro arduino. Para desempeñar esta tarea hemos decidido utilizar el sensor de temperatura TMP36.

El TMP36 es un sensor de temperatura de bajo voltaje, que proporciona una salida de voltaje que es linealmente proporcional a la temperatura Celsius (Centígrados). No requiere ningún tipo de calibración externa para conseguir unas precisiones de +-1C a una temperatura de +25C y de +-2C en un rango de temperaturas de -40C ~ +125c.

La baja impedancia de salida, la linealidad de de los datos de salida y su precisa calibración simplifican la interfaz con circuitos de control de temperatura y convertidores A/D. [16]

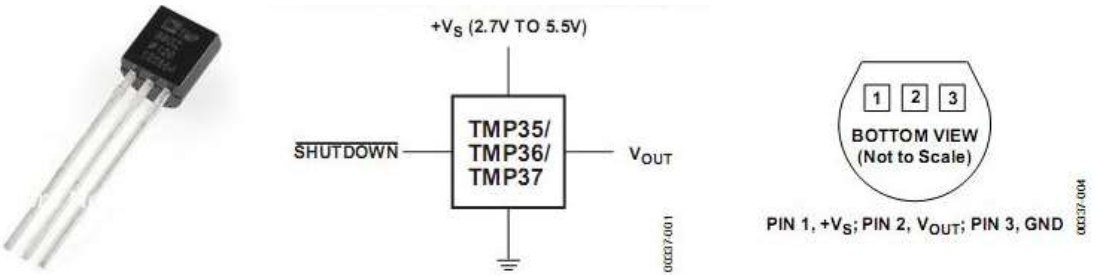


*Figura 11. Sensor TMP36*

En la figura 11 podemos observar la distribución de pines del sensor TMP36 y su aspecto físico. Para asegurarnos de que conectamos correctamente el sensor debemos fijarnos en los pines.

- Pin 1: Es el que debemos de conectar a la fuente de tensión, es decir al pin de 5 V o al de 3.3 V de nuestro Arduino.
- Pin 2: Es a través del cual se transmiten los datos recogidos por el sensor, luego lo deberemos de conectar a alguna de las entradas analógicas de nuestro Arduino, por ejemplo al pin A0.
- Pin 3: Es el que debemos de conectar a tierra, es decir, al pin GND de nuestro Arduino.

En nuestro proyecto escogeremos conectar el pin 1 al de 3.3 V del Arduino, ya que con esta tensión obtenemos unos resultados más precisos que con la de 5 V.

Características técnicas:

- Bajo voltaje de operación (2.7 V hasta 5.5 V)
- Calibrado directamente en grados Celsius
- Factor de escala de 10mV/C
- Precisión de +-2C
- Linealidad de +-0.5C
- Estable con elevadas cargas capacitivas
- Opera entre -40C ~ +125C

Aplicaciones:

- Sistemas de control ambientales
- Protección térmica
- Control de procesos industriales
- Alarmas de fuego

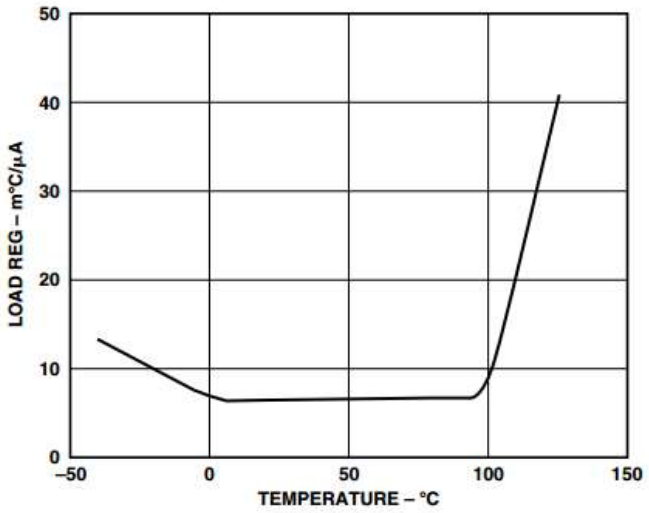


*Figura 12. Load Reg. Vs Temperature [16]*

En la figura 12 podemos observar la relación entre la carga registrada y la temperatura registrada para dicha carga.

En cuanto a las operaciones que debemos de incluir en nuestro código para conseguir obtener los valores númerico de temperatura tal y como los queremos, debemos de tener en cuenta las siguientes ecuaciones:

$$Voltaje = \frac{5}{1024}\ x\ lectura$$

$$Temperatura\ (C) = (Voltaje - 0.5)\ x\ 100$$

$$Temperatura\ (F) = \frac{9}{5}\ x\ Temperatura\ (C)\ + 32$$

En pocas palabras lo que está sucediendo es que estamos realizando una conversión de voltios a grados centígrados y farenheit, ya que la señal de salida del sensor es de tensión, y por tanto necesitaremos aplicarle una transformación.

### *3.5.1 Programación Sensor Temperatura – Ejemplo*

Para obtener la temperatura con valores correctos es necesario realizar una conversión de la señal de tensión (lectura del sensor) a grados Celsius como se muestra a continuación:

```
const int PinSensor= 0; //Pin del Arduino al que conectamos nuestro
sensor

void setup()
{
 Serial.begin(9600);
}

void loop()
{
 //medida sensor TMP36
 float voltaje, temp_med_C, temp_med_F;
```

```
 voltaje = analogRead(pinTemp) * 0.004882814; //señal de tensión
 temp_med_C = (voltaje - 0.5) * 100.0; //Grados celsius
 temp_med_F = (temp_med_C * (9.0 / 5.0)) + 32.0; //Grados Farenheit
}
```

La definición del pin que vamos a utilizar para capturar las lecturas de la temperatura es algo indispensable ya que sino el Arduino no estará registrando ningún dato. Podemos realizar dicha definición como en el ejemplo, *const int PinSensor = 0.* Los pines que podemos utilizar son todos aquellos que podemos ver en nuestra placa con los nombres Ai, i = 0,1,2, etc.

Aunque si es posible se recomienda no utilizar el pin 13 como entrada digital, ya que tiene un LED y una resistencia asociada soldados a la placa en la mayoría de ellas. Si activa la resistencia pull-up de 20k del interior provocará que el nivel de tensión en dicho pin se reduzca considerablemente por debajo de los 5V, por lo que no funcionará correctamente (siempre retornará bajo (LOW)). Una forma de solucionar esto sería utilizando una resistencia pull-up externa.

Es importante darse cuenta de que estamos trabajando con datos del tipo float, por lo que hay que cerciorarse de incluir el *.0* alfinal de cada número en la conversión, ya que sino las temperaturas que nos aparecerán serán totalmente diferentes a las esperadas.

# Capítulo 4. Desarrollo del Software

En este apartado entraremos más en detalle en los aspectos funcionales y la forma en la que hemos conseguido introducirlos mediante código de programación en nuestro registrador de datos.

En especial hablaremos sobre las librerías utilizadas, las funciones concretas que permiten la adquisición de datos y el diseño de la página web; que funcionará como una especia de software monitor basado en una solución TCP/IP LAN y pequeños detalles de programación que deberemos tener en cuenta para evitar problemas inesperados en el futuro.

Se hace necesario comentar, que en la elaboración de un proyecto como tal, conviene conocer los detalles más pequeños de programación de las diferentes piezas de hardware que utilicemos. Pues una vez conocidas las formas y funcionalidades básicas, resulta más inteligíble combinarlas hasta conseguir una aplicación final que funcione tal y como deseamos.

## *4.1 Inclusión de librerías externas*

El entorno de desarrollo de Arduino ya lleva incluido multitud de librerías para permitir un uso básico al usuario. Sin embargo podemos descargarnos librerías para unos usos específicos de nuestro dispositivo, en nuestro proyecto hemos utilizado las siguientes:

- SdFat.h
- SD.h
- RTClib.h
- Ethernet2.h

La forma de incluir correctamente dichas librerías de modo que el IDE las reconozca y nos permita utilizarlas es siguiendo los siguientes pasos [17]:

1. Ubicar y descargar la librería de Arduino que requerimos. Normalmente nos encontraremos con archivos en formato ZIP.
2. Abrir el IDE de Arduino y hacer click en *Sketch* > *Import Library…* > *Add Library* en la barra de menú.
3. Se mostrará la pantalla donde debemos indicar la ruta en la que se encuentra el archivo .ZIP.
4. Ahora la librería ya debería de estar incluida y debería haber salido una notificación de éxito, de todas formas podemos comprobarlo haciendo nuevamente click en *Sketch* > *Import Library* y deberíamos ver la librería en la lista.

Otra opción es la instalación manual, la cual podemos fácilmente realizar en unos pocos pasos:

1. Descargar la librería deseada.
2. Localizar la ruta donde está instalado Arduino y dentro de ella la carpeta *Librerías*, por ejemplo, C:\Users\Documents\Arduino\libraries.
3. Finalmente copiar la carpeta de la librería que se desea instalar en la ruta mostrada. En caso de que la hayamos descargado en formato .ZIP, deberemos descomprimir y tan solo extraer la carpeta que contenga los archivos *.cpp* y *.h*.

4. Comprobamos yéndonos a la barra de menú en el IDE de Arduino y clicando en *Sketch > Import Libraries* y comprobando que la nueva librería está en la lista.

Encontramos infinidad de librerías externas, muchas de ellas creadas por aficionados a la plataforma Arduino que nos pueden resultar de gran ayuda.

## *4.2 Estructura principal del código*

El programa principal se basa fundamentalmente en la definición al principio del código de las librerías, las variables globales y de dos funciones que están presentes en cualquier proyecto realizado con Arduino. En primer lugar la función *setup()* y en segundo lugar la función *loop()*.

La función *setup()* es la parte encargada de recoger la configuración. Se invoca una sola vez cuando el programa empieza y se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Asi mismo se puede utilizar para establecer el estado inicial de las salidas de la placa.

La función *loop()* contiene el código que se ejecutara continuamente. Después de llamar a *setup()* se ejecutará de manera cíclica permitiendo que el programa esté respondiendo continuamente ante los eventos que se produzcan en la placa.

La estructura de nuestro proyecto sería de la siguiente forma:

- *Librerías*: Incluimos todas las librerías de los elementos que vayamos a utilizar (reloj, tarjeta SD, Ethernet, etc)

- *Definición de variables globales*: Definimos los pines, buffers, y prácticamente todo parámetro que vayamos a utilizar

- *Funciones varias*: Definimos todas las funciones que conformarán el programa, de forma que luego podamos llamarlas y ejecutarlas

- *Función Setup()*: Inicialización de cualquier elemento externo al Arduino: módulo Ethernet y SD, el RTC y todo lo que conlleva, como el arranque del servidor, el ajuste de tiempo y fecha, inicio de la tarjeta SD.

- *Función loop()*: Ejecución de todas las funciones realizadas, de forma que se arranca el servidor y el proceso de adquisición de datos.

```
void setup() {

 // RTC ajustado a la hora & fecha en que se compilo el programa
 Wire.begin();
 rtc.begin(DateTime(F(__DATE__), F(__TIME__)));

 Serial.begin(9600); //Iniciamos el monitor serie

 pinMode(pinTemp, INPUT); //sensor temp.
 pinMode(chipSelect_ethernet, OUTPUT); // set the SS pin as an output
 digitalWrite(chipSelect_ethernet, HIGH); // but turn off the W5500
chip!
```

```
 // Inicializacion SD

 // Iniciamos el servidor tras haber iniciado la SD
 Ethernet.begin(mac, ip);
 server.begin();
}

void loop()
{

 char clientline[BUFSIZ];
 int index = 0;

 //Comprobamos de forma periodica si el archivo de configuracion se ha
editado
 if(SD.exists("Config.txt"))
 {
 SdConfig(); //Actualizamos los datos de muestreo
 }
 crear_DirSubdir(); // Comprobar y/o crear dir/subdir
 incremento_fichero(); // Actualizacion nombre fichero
 dataFile = SD.open(nombre_fichero, FILE_WRITE);
 Servduino_Datalogger(); //Webserver y Adquisicion de datos

 }
```

Como podemos ver en la función *setup()* se han iniciado los pines para la tarjeta SD y módulo Ethernet. También se ha iniciado la tarjeta SD y el servidor web.

Mientras que en la función *loop()* se realizan llamadas a múltiples funciones, consiguiendo un código sencillo y fácil de comprender.

- *crear_DirSubdir():* Comprueba que en la tarjeta SD exista una carpeta con el año actual y dentro de ésta otra con el mes actual en el que se ha encendido el dispositivo, en caso contrario las crea.
- *incremento_fichero():* Genera el nombre del fichero nuevo en función al tiempo actual, ya que dicho nombre es de la forma *DiaHoraMinutoSegundo.CSV*.
- *Servduino_Datalogger()*: Se encarga de hacer funcionar el servidor web respondiendo a las peticiones de los clientes en cualquier momento y al mismo tiempo de la adquisición de los datos que registra el sensor, todo de forma automática.
- *Listar_Archivos()*: Se encarga de representar en formato de hipervínculo en el explorador web todas las carpetas y archivos existentes en la tarjeta SD, de forma que podamos realizar multitud de cosas con ellos si los pulsamos (lectura, descarga, eliminación, etc).
- *SdConfig()*: Comprueba el archivo de configuración cada cierto tiempo y compara los datos de dicho fichero con los reales del programa (Tiempo_fichero y tiempo_muestreo), en caso de detectar algún cambio por parte del usuario en dichos parámetros del archivo, los aplica a las variables reales.

### *4.2.1 Consideraciones de memoria en Arduino*

Toda placa de Arduino cuenta con tres tipos de memoria que tienen propósitos ligeramente diferentes.

- *Memoria Flash (program space)*: Es donde se almacena el sketch del programa
- *Memoria SRAM (Static Random Access Memory)*: Es donde el sketch crea y manipula variables cuando se ejecuta
- *Memoria EEPROM*: Se trata de un espacio de memoria el cual los programadores utilizan para guardar información a largo plazo

La memoria Flash y EEPROM es no-volátil, de forma que se guardará incluso cuando el Arduino no tenga alimentación. Sin embargo, la memoria SRAM es volátil y sí que se pierde en cada apagado. En nuestro Arduino Mega tenemos las siguientes capacidades de memoria:

- Flash 256k byte (de los cuales 8k se usan para el bootloader)
- SRAM 8k bytes
- EEPROM 4k byte

La falta de memoria es uno de los principales problemas que nos pueden surgir en la elaboración de una aplicación. Debemos tener en cuenta que si nos quedamos sin memoria SRAM, el programa puede fallar de forma impredecible; aparentemente se compilará y subirá correctamente a nuestra tarjeta Arduino, pero no funcionará o lo hará con fallos.

Para comprobar si nuestra placa está falta de SRAM podemos probar a comentar ó acortar los strings u otro tipo de estructura de datos en nuestro sketch (sin cambiar el código). Si tras esta comprobación el programa funciona correctamente, lo más seguro es que dicha memoria esté llena. En este caso podemos probar las siguientes soluciones [18]:

- Si el sketch se comunica con un programa que está ejecutándose en un ordenador, podemos probar a cambiar algunos datos o cálculos al ordenador, reduciendo la carga en nuestro Arduino.
- Si nuestro sketch contiene arrays de un tamaño grande, podemos plantearnos el utilizar el tipo de dato óptimo para almacenar los diferentes tipos de datos, consiguiendo una mayor eficiencia. Por ejemplo, no utilizar un *int* que utiliza dos bytes de memoria, cuando basta con utilizar un *byte*, el cual utiliza solamente uno.

| Data Types | Size in Bytes | Can contain: |
|---|---|---|
| boolean | 1 | true (1) or false (0) |
| char | 1 | ASCII character or signed value between -128 and 127 |
| unsigned char, byte, uint8_t | 1 | ASCII character or unsigned value between 0 and 255 |
| int, short | 2 | signed value between -32,768 and 32,767 |
| unsigned int, word, uint16_t | 2 | unsigned value between 0 and 65,535 |
| long | 4 | signed value between -2,147,483,648 and 2,147,483,647 |
| unsigned long, uint32_t | 4 | unsigned value between 0 and 4,294,967,295 |
| float, double | 4 | floating point value between -3.4028235E+38 and 3.4028235E+38 (Note that double is the same as a float on this platform.) |

*Figura 13. Tabla de los tipos de datos [18]*

- Podemos almacenar los strings en memoria Flash en vez de SRAM. Existen diferentes variaciones en el código que podemos hacer, aunque muy sencillas. Por ejemplo, podemos utilizar la palabra clave *PROGMEM* al inicio de la línea de código que deseamos almacenar en Flash. Para poder utilizarla deberemos de incluir la librería *avr/pgmspace.h*.

### *4.2.2 Función Web Server*

Para el diseño de nuestra página web hemos creado la función *Servduino_Datalogger* la cual además nos permite gestionar los archivos existentes en la tarjeta SD. Aparte de esto como ya se había comentado, esta función también se ocupa de la adquisición de datos en la SD, pero en este apartado nos centraremos exclusivamente en la funcionalidad web.

En primer lugar para la creación de nuestra página web y de sus elementos, como pueden ser: links, botones web, colores, fuentes, etc hemos utilizado código *HTML*. Existen diferentes maneras de introducir código HTML en un proyecto de Arduino. En nuestro caso, nos hemos decidido a utilizar el código *in-line*, es decir, escribiendo el código en el propio IDE de Arduino junto al resto del programa, eso sí, utilizando un formato especial. Otra forma de la que hubiéramos podido manejar dicho código hubiera sido a través de la tarjeta SD, ya que el diseño

de una página web no es más que la conjunción de varios archivos. En este caso hubiéramos necesitado:

- *Archivo JavaScript*: Se encarga de la manipulación de imágenes, cambios dinámicos del contenido, ejecución de funciones, etc.
- *Archivo CSS*: Se encarga los aspectos decorativos de la web, como pueden ser los colores de fondo, de los botones, inserción de imágenes de fondo, etc.
- *Archivo HTML*: Es el que recoge la estructura de la página web, en el que se encuentra embedido el código CSS, JavaScript, etc.

De forma que en nuestro código de Arduino solo deberemos de hacer una llamada a dichos archivos para abrirlos al iniciar el servidor. La ventaja de este método es que ahorra mucha memoria SRAM del Arduino y supone utilizar menos código en nuestro programa. Sin embargo, como nuestro proyecto no requiere una página web con muchos elementos, el utilizar el código in-line no ocupa mucha memoria y tampoco hace el código muy complejo. Aparte de que utilizaremos técnicas en el código, las cuales se explicarán más adelante, que nos permitan utilizar la memoria Flash y reservar la memoria SRAM que es más preciada.

A continuación podemos ver un ejemplo de código HTML in-line:

```
// Petición HTTP
client.println(F("HTTP/1.1 200 OK"));
client.println(F("Content-Type: text/html;;charset=utf-8"));
 client.println();

 client.println(F("<td width=200px><font color=\"#00979d\" size=\"6\"
face=\"Verdana\"><strong>  Registrador  De  Temperaturas  Remoto
(Datalogger)</strong></font></td><td> </td></tr></table><br>"))
;
```

Básicamente el Arduino se encargará de generar una petición HTTP diseñada por nosotros, en la que se manda la información que el usuario desea incluir ó ejecutar en el servidor. En este caso, dentro de la petición se le está comunicando al servidor que cree un título en la página web denominado: <<*Registrador de Temperaturas Remoto (Datalogger)*>> con una determinada fuente, tamaño y color, atribuyéndole a su vez la propiedad de que se trata de texto (*Content-Type: text/html*), el cual puede contener cualquier caracter aceptado por el formato de codificación *utf–8*.

El comando *client.println()* se utiliza para indicar a la tarjeta Arduino que la información debe ser mandada al servidor. Como comentábamos antes, la memoria disponible es un factor crucial a la hora de desarrollar una aplicación, y como al usar el método in-line nos vemos obligados a utilizar parte de la memoria SRAM (la más importante), cambiamos el comando por *client.println(F()).* Esta pequeña variación nos permite almacenar dicha línea de código y todo lo que conlleva en la memoria Flash en vez de la SRAM, con lo cual deja de suponer problema alguno. Ya que específicamente en nuestro Arduino Mega disponemos de 253.952 bytes de memoria Flash y tan solo de 8.192 bytes de memoria SRAM.

En cuanto a la gestión de los archivos de la SD desde el servidor web, hemos diseñado varios botones web que al pulsarlos desempeñan las funciones básicas necesarias para poder gestionar de forma sencilla y eficiente los archivos.


*Figura 14. Botones web*

Los botones web diseñados los cuales podemos ver en la figura 13, son los siguientes:

- *Eliminar:ON*: Al pulsarlo el explorador web muestra una alerta explicando que se ha activado el modo de eliminación de archivos, lo que supone que al presionar sobre cualquier archivo este se eliminará.
- *Eliminar:OFF*: Al pulsarlo el explorador muestra una alerta en la que se comunica que el modo de eliminación se ha desactivado y se puede continuar con la lectura, descarga, etc de los archivos.
- *Descarga:ON*: Muestra una alerta comunicando que el modo descarga ha sido activado, por lo que al presionar cualquier archivo este inmediatamente se descargara en nuestro ordenador.
- *Descarga:OFF*: Modo descarga desactivado
- *EliminarSD*: Elimina todo el contenido de la tarjeta SD y además la formatea.
- *DescargarSD*: Descarga todo el contenido de la tarjeta SD.

En el caso de no pulsar ninguno de estos botones, al pulsar un archivo este mostrará todo el contenido que tiene.

Con el fin de cubrir un poco más las necesidades del usuario, también hemos incluido la opción de poder visualizar en tiempo real la temperatura, de forma que no es estrictamente necesario acceder al último fichero creado para imaginarnos cuál puede ser la temperatura ahora.

El diseño de la página web permite gestionar los archivos de forma versátil, rápida y sencilla. La disposición realizada de los archivos también contribuye, puesto que hemos decidido hacerla de forma estructurada, donde cada archivo se guarda dentro de la carpeta del mes en que fue creado, y a su vez esta se guarda dentro del año en el que se encuentra cuando fue creado. Por lo que una ruta típica de los archivos sería *2016/09/07154856.CSV*, donde 2016 es el año, 07 es el mes de Julio, y en cuanto al nombre del archivo este sería *Dia::Hora::Minuto::Segundo* de forma que se cumplimente la restricción de formato de nombres cortos en los que como máximo pueden utilizarse 8 caracteres de nombre y 3 caracteres de formato como ya habíamos comentado anteriormente.

La página web de nuestra aplicación tendría el siguiente aspecto:

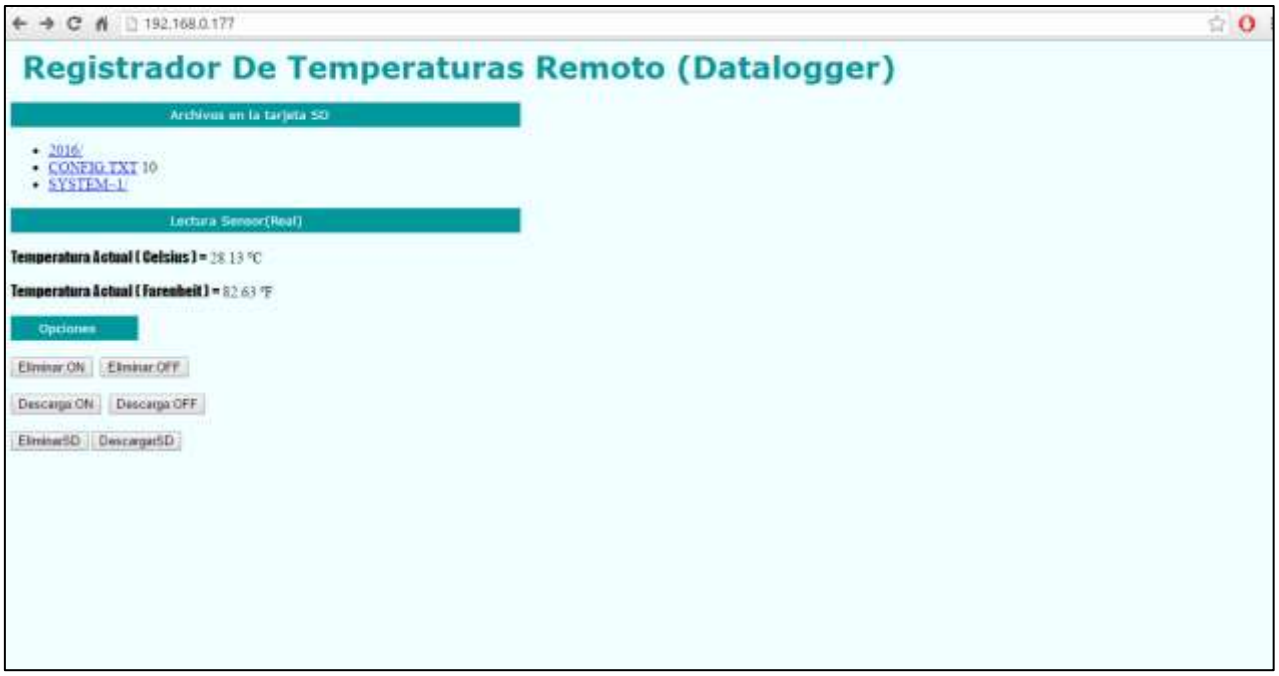


*Figura 15. Página Web principal*

En la figura 14 podemos ver la página principal de nuestra web. En ella podemos encontrar listados todos los archivos en el directorio principal *root* de la tarjeta SD insertada. En nuestro caso debido a la estructura que hemos diseñado, solo tendremos los directorios de los años y el archivo de configuración, en caso de que queramos cambiar el tiempo de muestreo y/o el tiempo de fichero de nuestro registrador.

A continuación tenemos la lectura de la temperatura en tiempo real en grados Celsius y en grados Farenheit los cuales se van refrescando cada cierto tiempo. Finalmente encontramos todas las opciones disponibles para la gestión de los archivos (eliminación y descarga).

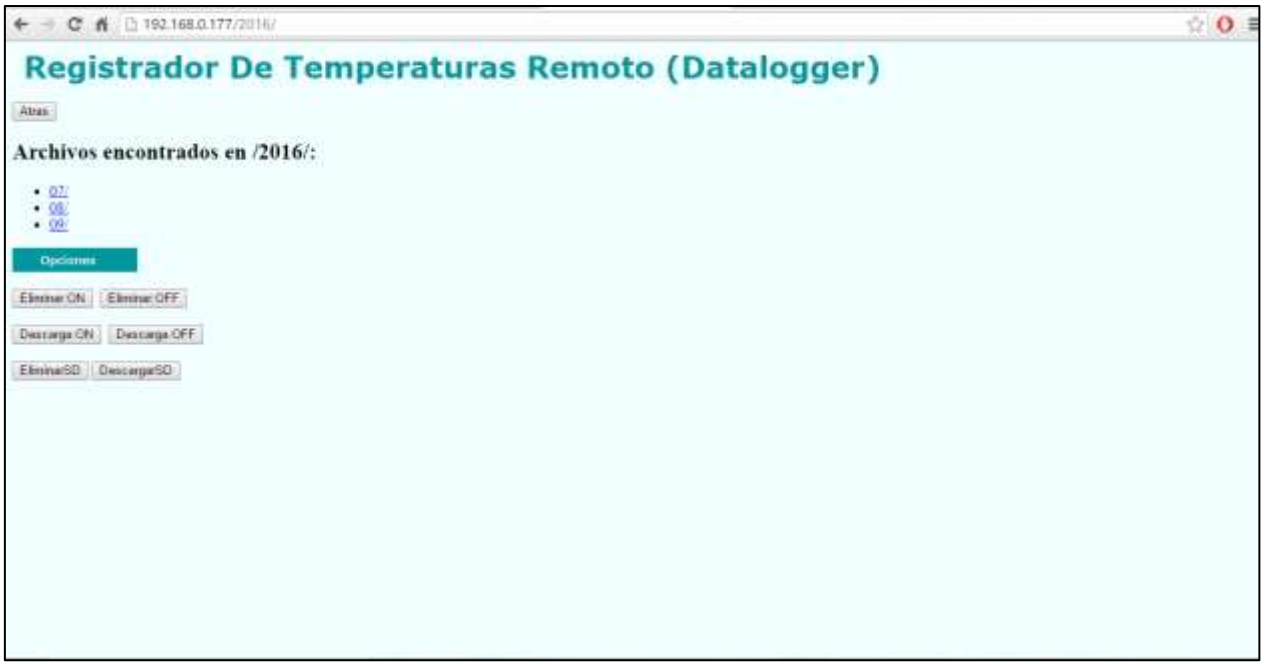


*Figura 16. Página web al hacer click en un directorio*

La siguiente imagen es el resultado de haber pulsado el hipervínculo del directorio */2016.* El explorador web nos mostrará los archivos contenidos en dicho directorio, que en este caso son subdirectorios que hacen referencia a los meses del año en los que se han registrado algún dato. Aparte también tenemos los botones para gestionar los archivos, para una mayor comodidad, y un botón para volver a la página anterior.

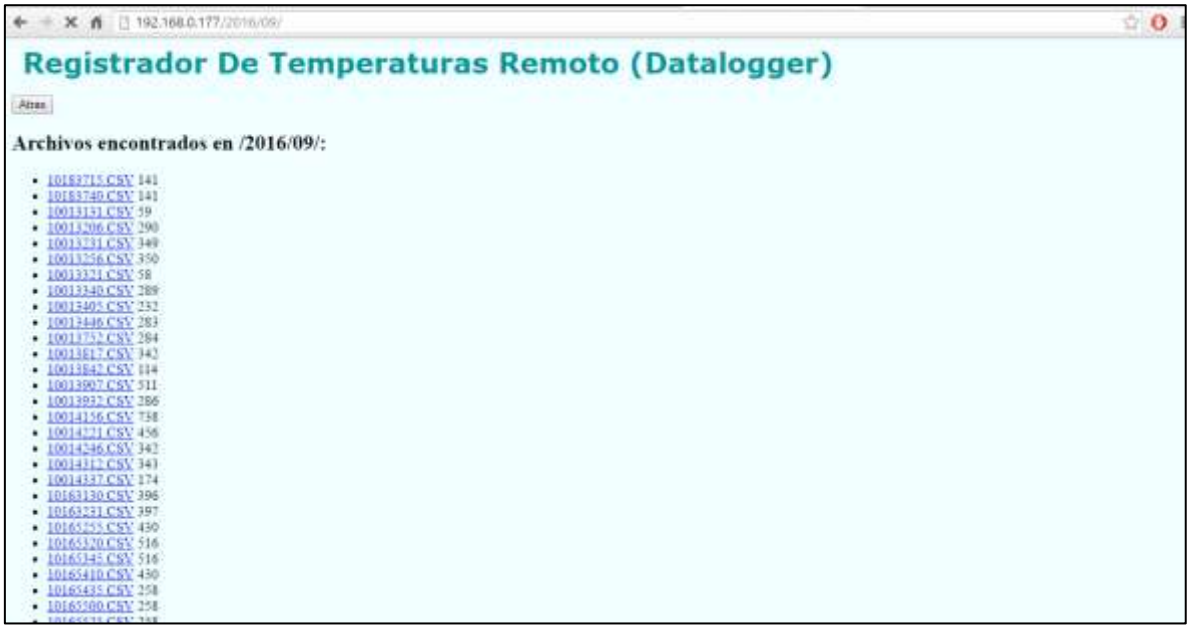


*Figura 17. Página web al hacer click en un subdirectorio*

Finalmente, Si pulsamos alguno de los subdirectorios referentes a los meses, se nos mostrarán todos los ficheros almacenados a lo largo de ese mes en forma de hipervínculo, de forma que al presionar podemos manipularlos de diferentes formas. Además también podemos ver el tamaño en bytes de cada fichero de registro, el cual aparece a la derecha de cada uno de los archivos. También volvemos a mostrar las diferentes opciones disponibles para una mayor comodidad.


*Figura 18. Página web al hacer click en un fichero de registro*

En la figura 17 podemos observar que al pulsar cualquier fichero de registro almacenado en la tarjeta micro SD el explorador web nos mostrará los datos contenidos en él conservando la misma estructura que el archivo original.

Si nos paramos a pensar, esta solución TCP/IP LAN como es la página web que hemos diseñado, junto con el número de funcionalidades que provee, da la gran ventaja al usuario de no tener que desplazarse físicamente al dispositivo y recoger la tarjeta SD, para luego manipular los archivos en el PC. Nos libra de la tediosa tarea que supone, ya que simplemente accediendo a la dirección IP del Arduino desde un PC o Smartphone podemos hacer todo tipo de cosas con los ficheros en tan solo cuestión de segundos.

### *4.2.3 Función adquisición de datos*

La implementación de ésta funcionalidad se desarrolla en la función *Servduino_Datalogger*, que como ya hemos dicho cubre tanto la funcionalidad web como la de recogida de datos, aunque en este apartado nos centraremos en la segunda. La adquisición de datos es el núcleo de la aplicación, por lo que hay que optimizarla lo máximo posible, es decir, tener en cuenta factores como el tiempo de muestreo, el tiempo de fichero, el número de lecturas por fichero, la información que contendrá el fichero, formato del fichero y que todo el proceso esté lógicamente automatizado.

*Tiempo de muestreo(s)* – Se define como la inversa de la frecuencia de muestreo (Hz) y se encarga de fijar cada cuanto tiempo se guarda una muestra del sensor.

*Tiempo de fichero (s)* – Define el tiempo máximo que un fichero permanece abierto para que se guarden muestras en él. Dicho tiempo cuenta desde que abrimos el fichero (*dataFile = SD.open(nombre_fichero, FILE_WRITE)*) hasta que lo cerramos (*file.close ()*).

*Nºmuestras/fichero* – Se trata de un parámetro que depende tanto del tiempo de fichero como del tiempo de muestreo de la siguiente forma:

$$Nºmuestras/fichero = \frac{Tiempo\ de\ fichero}{Tiempo\ de\ muestreo}$$

Básicamente la información que nos da es el número de datos que se han recogido en el sensor se incluirán en cada fichero.

*Información del fichero* – La cuestión ahora es que clase de información queremos que aparezca en nuestros ficheros. De forma imprescindible aparecerán las temperaturas en grados Celsius y de forma opcional añadiremos también el instante de tiempo en el que se recogió cada temperatura. Podríamos añadir múltiples datos más, por ejemplo sensores de humedad.

*Formato del fichero* – Hemos elegido el formato .CSV para los fichero ya que es el que ofrece una mayor facilidad para la lectura de datos, representación de los datos utilizando multitud de gráficas, etc. Este tipo de archivos no indica un juego de caracteres concreto, ni cómo van situados los bytes, ni el formato para el salto de línea, lo que lo convierte en un tipo de documento en formato abierto sencillo. Sin embargo, se trata de un formato que no resulta eficiente, por ejemplo en cuanto a la elección del nombre del archivo, ya que tan solo permite 8 caracteres de nombre junto a los 3 del formato. Aunque fijándonos en nuestro proyecto, es el formato más adecuado ya que se basa en la lectura y representación de datos.

*Automatización del proceso* – Para conseguir que la aplicación funcione de forma automática no basta con situar dicha función en el *loop()* para que se ejecute como un bucle. Debemos de fijar unas condiciones temporales de forma que la aplicación sepa cuando debe de abrir un fichero nuevo, adquirir un dato, cerrar el fichero, cambiar el nombre, etc. En la práctica nos hemos ayudado del reloj en tiempo real (RTC) utilizando la referencia temporal *unixtime()* que nos facilita la comparación entre unidades de tiempo, ya que se convierten en comparaciones de números enteros en vez de números sexagesimales, los cuales pueden ser más liosos.

Para poder ir comparando los tiempos de fichero, muestreo y el tiempo actual y final. Teóricamente podríamos describir el proceso de la siguiente forma:

- *Tiempo Actual*: Se trata del tiempo recogido por el RTC
- *Ultimo Tiempo*: El instante temporal en el que se guarda el dato, inicializado con valor 0 en el encendido.
- *t0*: Tiempo inicial en el que empieza el proceso, es decir, 0.
- *Tiempo de muestreo*
- *Tiempo de fichero*

La idea es básicamente conocer en qué momento el fichero tiene el número de datos que el usuario quiere, cuando se abre el próximo fichero, etc. Para ello basta con crear uno o dos bucles de la siguiente forma:

```
while((tiempo_actual == t0 + Tiempo_fichero) || (tiempo_actual < t0 +
Tiempo_fichero) )
{
 if((tiempo_actual-ultimo_tiempo      >      tiempo_muestreo)      ||
(tiempo_actual-ultimo_tiempo == tiempo_muestreo)
{
```

```
 //Adquisición de datos
}
]
```

En este ejemplo hemos hecho uso de dos bucles, el primero es un *while()* cuya función es comprobar si ha pasado el tiempo en el que el fichero tiene que estar abierto, de modo que si no ha pasado pasa al siguiente bucle, y en caso contrario se cerraría el fichero y se abriría uno nuevo.

En el caso de que no haya pasado el tiempo de fichero, pasamos a un bucle *if()* el cual se encarga de determinar cuándo se debe de guardar un dato en el fichero que tenemos abierto. El dato se guardará cuando la diferencia entre el tiempo actual y el último tiempo en el que se guardó la última muestra sea mayor al tiempo de muestreo, lo que significa que ha pasado el tiempo que hemos determinado entre muestra y muestra y por tanto se puede proceder a la adquisición.

A continuación podemos verlo de una forma más visual:

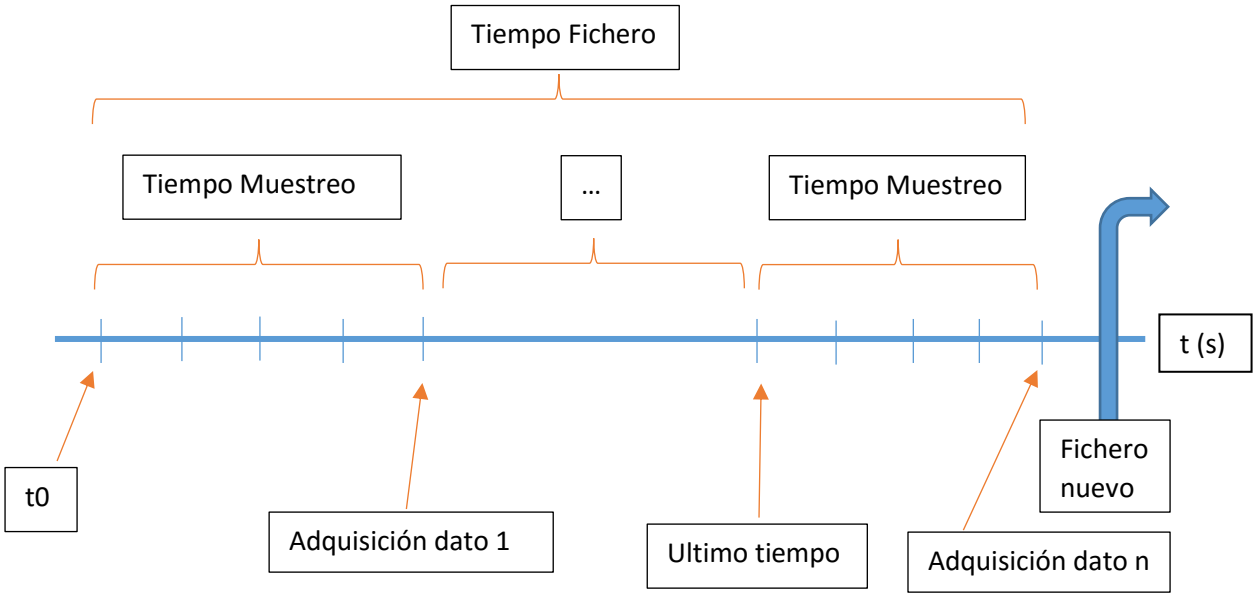


*Figura 19. Esquema teórico del proceso de adquisición de datos*

Debemos recordar que el tiempo de muestreo y el de fichero lo podemos variar editando el archivo de configuración que se encuentra dentro de la tarjeta SD y que es visible entrando en la página web. Ya que dependiendo del tipo de datos que estemos recogiendo y de la localización del sensor nos convendrá más un período de muestreo que otro.

### *4.2.4 Función Archivo de configuración*

Para dotar a nuestro registrador de una mayor versatilidad y eficacia se hace necesario la inclusión de un fichero que se encargue de transportar información vital para la adquisición de datos: *Tiempo de fichero* y *tiempo de muestreo*. Se trata del fichero de configuración, al que hemos denominado "**Config.txt"** haciendo referencia al uso que provee. El archivo tiene una estructura muy sencila, simplemente se trata de una línea con dos números enteros, cada uno de ellos representado al tiempo de fichero y el de muestreo.


*Figura 20. Contenido fichero configuración*

Como podemos ver en la figura 19, el valor 25000 hace referencia al Tiempo de fichero, mientras que el valor 5000 hace referencia al tiempo de muestreo, ambos en milisegundos.

La función que hemos desarrollado para que se encargue de la comprobación y actualización del fichero de configuración, para detectar si este ha sido modificado y en caso de ser así llevar a cabo el reajuste en los valores de tiempo de muestreo y de fichero es ***SdConfig().***

En resumidas cuentas, se trata de una función que comprueba periódicamente el contenido del fichero de configuración (Config.txt) almacenado en la tarjeta SD y simultáneamente actualiza dicho contenido con el ya existente. En el caso de ser el mismo no ocurrirá nada, sin embargo en caso de ser diferente los cambios se realizarán con efecto inmediato, todo esto debido a que hemos situado la función en el loop().

La forma en la que hemos codificado dicha función obliga al usuario a utilizar físicamente la tarjeta SD en caso de que quiera cambiar los parámetros de configuración del registrador. De forma que el usuario realizará los cambios modificando el archivo en el PC y posteriormente volverá a introducir la tarjeta SD en el Arduino. En cuanto a comprobación sí que podemos utilizar la página web para descargar el contenido del archivo de configuración, eliminarlo o visualizarlo, pero no subir un archivo modificado directamente desde nuestro terminal al Arduino. Aunque sí que es algo que queda como línea futura.

# Capítulo 5. Presupuesto y comparativa con dataloggers comerciales

En este capítulo vamos a realizar el presupuesto de nuestro registrador de datos, de forma que podamos compararlo con la multitud de registradores existentes que están actualmente a la venta. Nos podremos dar cuenta de la ventaja que supone la aplicación desarrollada por nosotros en comparación con los productos a la venta que puedan a primera vista resultar la solución óptima, tanto a nivel de precio como de funcionalidad.

## *5.1 Presupuesto del Proyecto*

A continuación haremos un resumen de todas las piezas de hardware que hemos utilizado a lo largo del proyecto, de forma que podamos realizar de forma esquemática un presupuesto final de nuestro registrador de datos remoto de bajo coste. Esto formará parte de los factores que nos ayudarán a comparar nuestro dispositivo con otros similares disponibles en el mercado.

| **Pieza de Hardware** | **Coste** |
|---|---|
| Arduino Mega 2560 | 47,20 € |
| W5500 Ethernet Shield 2 | 19,98 € |
| RTC Groove | 6,78 € |
| Tarjeta Micro SD (2GB) | 7,79 € |
| Sensor de Temperatura TMP36 | 3,40 € |
| Cableado (Ethernet) | 3,00 € |
| **TOTAL:** | **88,15 €** |

*Tabla 1. Presupuesto del proyecto*

El presupuesto final es de 88.15€, teniendo en cuenta el IVA (21%). Lo que en comparación con muchos registradores de similares características supone un coste bastante bajo, ya que prácticamente todos ellos superan los 100€ de coste.

## *5.2 Registradores comerciales*

Podemos encontrar una gran cantidad de registradores de datos a la venta, en este apartado trataremos de mostrar algunos de estos registradores que tengan aspectos parecidos a nuestro registrador, ya que es difícil que ofrezcan exactamente las mismas soluciones.

A la hora de compararlos con nuestra aplicación nos fijaremos en la funcionalidad de acceso remoto, la capacidad de medir magnitudes, la capacidad de manipular los archivos a distancia, la cantidad de registros que pueden producirse etc.

| **Producto** | **Características** | **Coste** |
|---|---|---|
| S300-ex-rj45 seguimiento de forma remota Data Logger IP temperatura y humedad (Data Logger Ethernet termómetro) [19] | 1. Pantalla LCD muestra la temperatura y humedad<br>3. Seleccionable ° C o ° F<br>4. USB/RS232/RS485/RJ45<br>5. 43,000/86,000 lecturas de registro<br>6. Software de configuración y la representación gráfica<br>7. Dimensiones: 120mm * 110mm * 33mm<br>8.Monitorea de forma remota la temperatura y humedad atendiendo a una solución TCP/IP LAN | 248.36€ |
| R90-EX-W WIFI Wireless Temperatura y Humedad [20] | 1. 65,000 lecturas<br>2. Rango Temp sensor 1.: 20C a +70C<br>3. Rango Temp. Sensor 2: -40C a +125C<br>4. Rango humedad: 0 a 100% RH<br>5. Precisión Temp.: De -0.3C a +0.3C<br>6. Precisión Humedad.: De -3% a +3%<br>7. Ciclo medida: 1s ~ 24h<br>8. Software monitor: Sí | 149.86€ |
| OM-EL-WIFI-TP-PLUS [21] | 1. Data logger wifi de temperature y humedad.<br>2. WiFi y display integrado<br>3. Conectividad inalámbrica para PC<br>4. Larga memoria de almacenamiento de datos<br>5. Lecturas máximas y mínimas<br>6. Set-Up usando el software para PC | 191€ |
| Registrador de temperatura WiFi con sonda interna [22] | 1. Monitorización de temperatura automatizada con sensor integrado<br>2.Sistema de registro de datos flexible y con instalación<br>3. Acceso a los datos de forma remota<br>4. Precisión Temp: De -0.5C a +0.5C<br>5. Rango Temp.: -30C a +50C<br>6. Temp Funcionamiento: -30C a +50C<br>7. Memoria: 10.000 valores/canal<br>8. Intervalo Medición: 15min (Basic) o 1min-24h (Advanced) | 190€ |

*Tabla 2. Comparativa dataloggers comerciales*

# Capítulo 6. Conclusiones y Líneas futuras

En el apartado anteriormente se han comparado las prestaciones entre nuestro registrador de datos y algunos que se encuentran en el mercado. A continuación se muestran las principales características de nuestra aplicación, de forma que se comentarán las diferencias con las de los otros registradores.

| Producto | Características | Precio |
|---|---|---|
| Registrador de Datos remoto Ethernet | Sensor: Lm35<br>Rango Temp.: -55C ~ +110C<br>Precisión: -0.5C ~ +0.5C<br>Número Canales: 1<br>Memoria: Micro SD 2GB<br>Software Monitor: Sí<br>Control remoto: Ethernet | 88.2€ |

*Tabla 3. Tabla de características del proyecto*

Como podemos comprobar, nuestro registrador de Arduino tiene características muy similares a los registradores comerciales mostrados anteriormente, sin embargo, el punto fuerte de nuestro registrador es que permite interactuar en tiempo real con la información recogida por el dispositivo, ya que actúa como una especie de servidor FTP, haciéndolo más versátil y efectivo que los registradores comerciales comentados. Ciertamente, la mayoría de ellos lleva un software monitor también, pero la gran mayoría de ellos no permite las mismas funcionalidades que nuestro registrador, dándole al usuario el lujo de no tener que manipular la tarjeta SD físicamente para trabajar y manipular los archivos.

En cuanto a la memoria, nuestro registrador tiene una capacidad de 2GB lo que resulta suficiente para guardar unos miles de registros, aunque evidentemente dependerá del número de canales y la frecuencia de muestreo. Además, en el caso de querer aumentar éste parámetro solo deberíamos de comprar una tarjeta SD de mayor capacidad e intercambiarlas, lo cual no tiene ninguna complicación.

El archivo de configuración disponible posibilita el hecho de que los tiempos de muestreo sean completamente ajustables en nuestro dispositivo, permitiendo establecer como mínimo del orden de milisegundos y cómo máximo el tipo de variable que hayamos utilizado para ello. En nuestro caso se trata de un tipo de dato <<*unsigned long*>>, lo que equivale a un tamaño de 4 bytes permitiéndonos contar hasta 2.147.483.647 segundos, es decir, establecer tiempos de registro de años. Si comparamos esta característica con las de los demás registradores, nos daremos cuenta de que los que más tiempos de medición admiten son de 1 mili segundo hasta 24 horas.

El número de canales es un parámetro en el que nuestro dispositivo puede encontrarse en desventaja respecto a otros registradores ya que el conversor A/D es compartido para todas las

entradas analógicas que presenta ARDUINO y por tanto, la frecuencia de muestreo máxima permitida se divide por el número de canales que queramos capturar más un factor de penalización debido al tiempo de conmutación del multiplexor. En el caso de querer aumentarlo y conseguir muchísimos canales más que los ofrecidos por la mayoría de registradores sería mediante la utilización de multiplexores a la entrada de los diferentes canales de entrada, los cuales podemos encontrar por un precio de 0.60€, como en el caso de los multiplexores MC14051BCP (8 canales a 1).

El sensor de temperatura utilizado en nuestro Arduino tiene una buena exactitud de temperatura y un buen rango de temperaturas admisibles, que en comparación con el resto de registradores resulta aceptable a la vez que similar.

Hay algunos de los modelos de registradores que hemos comentado anteriormente, que utilizan una conectividad WiFi, lo que toma en ventaja a la conexión Ethernet desde el punto de vista de compartición de información con el resto de usuarios. Sin embargo, hay que tener en cuenta que en él fondo, la finalidad de un registrador de datos puede ser tanto de uso personal como global. En cuanto a la calidad de servicio, la conexión WiFi introduce mucho ruido comparada con la Ethernet, con lo que la gestión de dicha información puede ser muy lenta o prácticamente imposible en algunas ocasiones. En el caso de que decidiésemos implantar una conectividad WiFi en nuestro registrador, deberíamos de cambiar nuestro Shield Ethernet por uno WiFi, el cual tiene un precio actual de 84.58€, e incluir las correspondientes librerías.

Se hace necesario comentar que una de las desventajas que tiene nuestro registrador es que el acceso al servidor web solo es accesible si nos encontramos conectados a la misma red LAN que el dispositivo. Por lo que técnicamente no tendría una conectividad a internet, más bien sería para un uso personal en una red privada accediendo mediante VPN. Sin embargo en el caso de que quisiéramos dotar a la aplicación de dicha conectividad para que se pueda acceder desde cualquier parte sería necesario realizar los siguientes cambios:

- ***Habilitar Portforwarding en el router***: Se trata de una aplicación de NAT que redirige toda la información que llega a una dirección y puerto específicos hacia otra, de forma que permite a ordenadores a lo largo de internet conectarse a un ordenador específico que se encuentra en una LAN.

- ***Contratar servicio de IP estática***: Es necesario contratar un servicio de ip estática a nuestro operador telefónico (normalmente caro), ya que los operadores cambian nuestra IP pública de forma temporal (normalmente cada semana) y necesitamos que nuestro servidor siempre tenga la misma.

- ***Acceso mediante DNS***: Una alternativa a la opción de arriba, es mantener la dirección IP dinámica de forma que teniendo un código en un PC que esté conectado al router se encargue de chequear periódicamente la IP y actualice la DNS de forma que la red de nuestro Arduino siempre sea visible (excepto unos minutos que será cuando se actualice). Por lo que accederíamos al servidor mediante el nombre de dominio, p.e: *www.RegistradorDatos.es*.

Debemos comentar el hecho de que existen multitud de registradores de datos Arduino con precios similares a los expuestos que no permiten una conectividad de red de ningún tipo, ni la gestión remota de sus archivos. Sin embargo, tienen unas mejores características respecto al número de canales, memoria, sensores, etc. Como hemos visto esto son características realizables

en nuestro registrador, y teniendo en cuenta el presupuesto alcanzado, sería viable la opción de mejorar dichas características.

En conclusión, vemos que es una buena opción la adquisición de nuestro registrador de datos, ya que con un presupuesto de 88.2€ consigue tener características similares a las de muchos registradores de datos comerciales y en muchos casos también superiores.

# Bibliografía

## ***Capitulo 1***

[1] “Registradores de datos - Dataloggers”, ), Sensing S.L., (http://www.sensores-de-medida.es/sensing_sl/EQUIPOS-DE-ADQUISICI%C3%93N-DE-DATOS_32/Registradores-de-datos---Dataloggers_229/).

## ***Capítulo 2***

[2] Enrique Crespo, “Aprendiendo Arduino: Comunicaciones con Arduino”, aprendiendoarduino, (https://aprendiendoarduino.wordpress.com/2014/11/18/tema-6-comunicaciones-con-arduino-4/ ), 18/11/14.

[3] “Arduino UNO”, Arduino Webpage, (https://www.arduino.cc/en/Main/ArduinoBoardUno).

[4] “Arduino Mega 2560”, Arduino Webpage, (https://www.arduino.cc/en/Main/ArduinoBoardMega2560).

[5] “Shields para Arduino”, aprendiendoarduino, (https://aprendiendoarduino.wordpress.com/2015/03/23/shields-para-arduino/).

[6] “Interfaces Físicas: Comunicación Serial”, dtic, (http://www.dtic.upf.edu/~jlozano/interfaces/interfaces6.html).

[7] Ivan, “RTC DS1307, Real Time Clock”, Patagoniatec, (http://saber.patagoniatec.com/rtc-ds1307-real-time-clock-fecha-hora-i2c-real-time-clock-eeprom-arduino-pic-ptec/).

[8]”Introducción Arduino”, Arduino Website, (https://www.arduino.cc/en/Guide/Introduction).

[9] Omar Otoniel Flores, “Batalla de microcontroladores ¿ATMEL o PIC?”, Boletín facultad de informática y ciencias aplicadas, (https://microcontroladores2utec.files.wordpress.com/2009/11/180909_articulo_colaboracion_boletin_fica_omar_otoniel_flores.pdf), UTEC Septiembre de 2009.

[10] “Raspberry Pi”, Wikipedia, (https://es.wikipedia.org/wiki/Raspberry_Pi).

[11] “Raspberry Pi: Qué es, características y precios”, Culturación, (http://culturacion.com/raspberry-pi-que-es-caracteristicas-y-precios/).

[12] “Productos de microcontroladores PIC”, Microchip, (http://www.microchip.com/ParamChartSearch/chart.aspx?branchID=211&mid=10&lang=en&pageId=74), 1998-2016.

[13] “Microcontroladores PIC y sus variedades”, Microntroladoressesv, (https://microcontroladoressesv.wordpress.com/microcontroladores-pic-y-sus-variedades/).

## ***Capítulo 3***

[14] “Arduino Ethernet”, Arduino Website, (http://www.arduino.org/products/shields/arduino-ethernet-shield-2).

[15] “Grove RTC”, SeedWiki, ( http://wiki.seeedstudio.com/wiki/Grove_-_RTC), last modified 15 February 2016.

[16] “Low Voltage Temperature Sensors TMP35/TMP36/TMP37”, Analog Devices Datasheet Arduino, (https://www.arduino.cc/en/uploads/Main/TemperatureSensor.pdf).

***Capítulo 4***

[17] “Installing Arduino Additional Libraries”, Arduino Webpage, (https://www.arduino.cc/en/Guide/Libraries).

[18] “Memories of an Arduino”, Adafruit Webpage, (https://learn.adafruit.com/assets/31818).

***Capítulo 5***

[19] “S300-EX-RJ45 Data Logger Termómetro Temperatura y humedad Data Logger Ethernet IP”, Aliexpress , (http://es.aliexpress.com/store/product/S300-EX-RJ45-Remotely-Monitoring-Data-logger-IP-Temperature-and-Humidity-Data-Logger-Ethernet-Thermometer/1943381_32540402856.html).

[20] “R90-EX-W WIFI Wirelss Temperatura y Humedad Data Logger”, Aliexpress,

[21] “OM-EL-WIFI-TP-PLUS: Data logger wifi para temperatura y humedad”, OMEGA , (http://es.omega.com/googlebase/product.html?pn=OM-EL-WIFI-TP-PLUS&gclid=Cj0KEQjw3ZS-BRD1xu3qw8uS2s4BEiQA2bcfMwIklMg5AQTdHjzrf1ImZ2IyKYVEL_R8C_EwBiYHSlUaAgf_8P8HAQ).

[22] “Registrador de temperatura WiFi con sonda interna y licencia para nube", TermoMed, (http://www.termomed.net/registrador-temperatura-wifi-sonda-interna-licencia-para-nube-p-1148.html?gclid=Cj0KEQjw3ZS-BRD1xu3qw8uS2s4BEiQA2bcfM8bH_JR2DrjddBea5KFsDqA7pW7hcdsxx4yIrM3l5N4aAupg8P8HAQ).