ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

*Proyecto Fin de Carrera*

# REGISTRADOR DE DATOS DE BAJO COSTE
(LOW COST DATALOGGER)

Para acceder al Titulo de

**INGENIERO TÉCNICO DE TELECOMUNICACIÓN**

Autor: Javier Pérez Gutiérrez

Septiembre - 2013

UNIVERSIDAD DE CANTABRIA

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

**INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN**

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

**Realizado por: Javier Pérez Gutiérrez**
**Director del PFC: Esteban Stafford**
**Título:** "Registrador de datos de bajo coste"
**Title:** "Low cost Datalogger"

**Presentado a examen el día:**

para acceder al Título de

**INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SISTEMAS ELECTRÓNICOS**

Composición del Tribunal:
Presidente (Apellidos, Nombre):
Secretario (Apellidos, Nombre):
Vocal (Apellidos, Nombre):

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

# Agradecimientos

Quiero agradecer todo el esfuerzo y apoyo recibido de mi tutor Esteban Stafford, ya que me ha dedicado todo el tiempo necesario y ha valorado mi trabajo desde el primer día que empezamos a trabajar en este proyecto.

También tengo que acordarme del incondicional apoyo de mi familia, la cual siempre ha estado ahí, en los momentos fáciles y sobre todo en los difíciles. Sin ellos, habría sido imposible recorrer todo este camino.

Javier

# Tabla de contenidos

# Prólogo

La creciente proliferación de los sistemas SBC (Single Board Computers) que se vive en estos tiempos, abre un gran abanico de posibilidades computacionales de moderado rendimiento que posibilitan la realización de multitud de proyectos de bajo coste. La versatilidad de este tipo de sistemas, permite que usuarios sin conocimientos avanzados en programación y en electrónica, puedan implementar de forma compacta multitud de aplicaciones que se puedan imaginar.

La plataforma sobre la cual se desarrolla este proyecto, *Arduino Board*, es uno de los ejemplos de sistemas basados en microcontroladores que ofrecen la potencia y sencillez necesaria para realizar aplicaciones realmente útiles para los usuarios. Todo esto unido a su carácter *"Open Source"* o código abierto, hace de los sistemas SBC una opción económica para satisfacer las inquietudes de programadores, artistas, diseñadores y aficionados a la electrónica.

Como ejemplos de aplicaciones capaces de ser desarrolladas en estas plataformas, podemos encontrar cosas tan interesantes como web servers accesibles desde red local e internet, control de aplicaciones mediante Wireless, aplicaciones domóticas en viviendas, registradores de datos *(dataloggers),* robots, control de regadío, iluminación de diseño mediante matrices de leds y un sin fin de proyectos que cualquier persona pueda imaginar.

En nuestro caso, vamos a desarrollar un proyecto de control de temperaturas aplicable a cualquier situación que lo precise. Así mismo, la monitorización de las temperaturas deseadas podrá ser configurable desde un archivo de configuración editado por el usuario. Para hacer la aplicación más compacta, se van a registrar en una memoria micro SD las medidas realizadas obteniendo así un sistema registrador de datos *(datalogger)* completamente autónomo, funcional, configurable y muy económico.

# Capítulo 1

# Introducción

Se desea implementar un dispositivo capaz de registrar y almacenar datos de magnitudes físicas como puede ser la temperatura de diferentes zonas simultáneamente. Concretamente, se debe registrar la temperatura de diferentes estancias de una vivienda, para así poder conocer las variaciones registradas durante largos periodos de tiempo. Con estos datos, podríamos regular o economizar los gastos en calefacción, regulando los niveles de calor en las distintas estancias.

Estos sistemas conocidos ampliamente como *Registradores de datos* o *"Data Loggers"*, son sistemas muy versátiles ya que permiten el registro de muchas magnitudes físicas y la actuación en consecuencia. En este caso trabajaremos con temperaturas, pero tenemos la posibilidad de registrar tensiones, corrientes, humedad y luminosidad entre muchas otras.

## 1.1 Requisitos del proyecto

Además de conocer las temperaturas de las estancias, el dispositivo deberá ofrecer una serie de prestaciones que permitan al usuario poder conocer y configurar su sistema de monitorización de forma rápida y sencilla. Los principales requisitos del sistema se describen como:

- *Completamente configurable desde un fichero externo.* Se debe permitir que el usuario del registrador, pueda modificar el número de puntos de medida, los tiempos de muestreo y otros factores mediante la edición de un fichero de configuración.

- *Funcionamiento autónomo durante largos periodos de tiempo*. La idea es que el sistema registrador este alimentado y funcionando de forma autónoma mediante una fuente de alimentación, aunque también cabria la posibilidad de alimentarlo mediante el cable Ethernet.

- *Monitorización de magnitudes analógicas y digitales*. El sistema registrador de datos, deberá contar con una serie de entradas analógicas y digitales que reciban información a través de los sensores más adecuados.

- *Posibilidad de conexión de varios sensores simultáneos*. Al contar con varias entradas analógicas, se permite la monitorización de temperatura de forma simultánea en distintos lugares de la vivienda.

- *Periodo de muestreo variable*. En un sistema registrador de datos, debemos permitir al usuario que el periodo de recogida de datos se produzca a intervalos de tiempo deseados, desde segundos a días.

- *Posibilidad de preprocesar datos.* Las magnitudes recogidas podrán y deberán ser tratadas antes de ser mostradas. Los datos recibidos de los sensores, serán leídos comúnmente en forma de mili voltios lo cual no tiene mucho significado para el usuario. Internamente el sistema realizara las conversiones necesarias para mostrar las temperaturas en grados.

- *Capacidad de representación posterior de los datos obtenidos.* Con los datos obtenidos regularmente, el usuario necesitará algún software externo que los represente gráficamente, para así tener una visión más global de las variaciones de temperatura.

- *Diseño de un Pcb compatible y adaptable con la plataforma*. Se debe realizar un diseño sobre un Pcb, que contenga todos los elementos necesarios de nuestro sistema. De esta forma, obtendremos un sistema compacto y funcional sobre un circuito impreso.

Teniendo en cuenta los requisitos expuestos anteriormente, se ha pensado en la utilización de la plataforma *"Arduino"* para llevar a cabo dicho proyecto. La plataforma Arduino consiste en una tarjeta basada en un microcontrolador de 8 bits. Este microcontrolador tiene acceso a multitud de puertos de entrada/salida de tipo analógico y digital, lo cual lo hace una plataforma excelente para recoger señales mediante sensores, como en este caso la temperatura.

Otra de las grandes motivaciones para escoger la plataforma Arduino es sin duda su carácter *"Open Source"* o código abierto, lo que significa que el hardware y el software es de libre utilización por cualquier usuario. Esto implica la posibilidad de modificar el hardware y fabricar tarjetas especificas basadas en Arduino si fuese necesario.

También nos brinda la posibilidad de escribir nuestras propias librerías y de utilizar las librerías disponibles en las comunidades de Arduino para hacer los programas más compactos y funcionales. Con un lenguaje basado en C/C++ el cual es ampliamente conocido, las posibilidades de programación se agrandan hasta estar delante de una plataforma muy flexible en todos los aspectos.[1]

## 1.2 Estructura del proyecto

En este proyecto se propone una solución de bajo coste basada en una SBC *(Single Board Computer*) provista por la plataforma Arduino, que conjuntamente con elementos de medida externos y otros componentes discretos, cumpla con todas las especificaciones expuestas anteriormente.

En el capítulo 2, se reúne la información acerca de la SBC de Arduino utilizada, destacando las principales características de esta y su adaptación a las necesidades

descritas. También se hará una comparación con otras de las plataformas disponibles actualmente que se deben tener en cuenta como otra opción de diseño.

En el capítulo 3, se hace un estudio del hardware externo necesario y de su funcionamiento conjunto con Arduino. Entre estos componentes se encuentran los sensores de temperatura disponibles, multiplexores para incrementar los puntos de medida, tarjeta microSD para el almacenamiento de datos y un reloj de tiempo real que se encargara de proveer de un registro horario ininterrumpido para nuestro registrador de datos.

En el capítulo 4, se explica toda la parte software del proyecto, donde encontraremos las estrategias de programación de Arduino que se ponen a nuestra disposición mediante todas las librerías Open Source disponibles. Se abordará la conexión Ethernet, la tarjeta microSD, el preprocesado de datos, la obtención de datos y el registro horario de eventos.

En el capítulo 5, se realizara un diseño de Pcb para poder disponer de una placa totalmente compatible con nuestra tarjeta Arduino. También se presentara una forma de analizar los datos almacenados en la tarjeta SD.

En el capítulo 6, se exponen las conclusiones del proyecto así como el presupuesto previsto para implementar el sistema registrador de datos. Además se hará una comparación con algunos de los registradores datos comerciales disponibles en el mercado.

# Capítulo 2

# Single Board Computers (SBC)

Cuando hablamos de un Single Board Computer (SBC), nos referimos a una plataforma de computo basada en un microprocesador o microcontrolador que esta presentada en una sola tarjeta, provista de todo lo necesario para su funcionamiento autónomo, como la memoria de programa, la memoria de datos y diversos dispositivos de entrada/salida.

Inicialmente se utilizaban como sistemas de desarrollo, sistemas de pruebas, demostraciones y para fines académicos. Más adelante, se comenzaron a utilizar como sistemas de computo empotrados presentes en lugares con necesidades de menor potencia, mayor versatilidad y menor volumen. El ejemplo más directo de uso, es el desarrollo inmediato de aplicaciones por parte del programador, incluso el no experimentado en algunos casos, debido a que la tarjeta es completamente funcional y autónoma.

Se pueden apreciar algunas diferencias entre la SBC basadas en microprocesadores y las basadas en microcontroladores. Las ultimas carecen de una interfaz de usuario de propósito general y de dispositivos de almacenamiento masivo, aunque esto último ya está actualmente resuelto. Las tarjetas basadas en microcontroladores enfatizan el control y la adquisición de datos o medidas a través de entradas y salidas analógicas/digitales, mientras que el sistema con microprocesadores suele apoyarse más en la implementación interna o de ejecución, más que en las funciones externas.

Los sistemas microcontrolados poseen generalmente multitud de entradas y salidas para el control de magnitudes físicas, áreas de prototipado para conexiones de otros dispositivos y la posibilidad de conectar otras tarjetas de expansión mediante protocolos de comunicación configurables por el usuario. Algunas de las expansiones más interesantes de estas tarjetas son la conexión de red ethernet, el almacenamiento en tarjetas de memoria SD, etc.

Los microcontroladores utilizados poseen ciertas ventajas frente a los procesadores, como la integración de la memoria de programa y de datos en el propio chip, lo que reduce el coste y el espacio ocupado en las tarjetas. Debido a esto, directamente se aumenta la cantidad de entradas/salidas disponibles para propósitos generales. Algunas de las arquitecturas más utilizadas en estas tarjetas son: Intel 8048, Atmel AVR, PIC y ARM.

Hoy en día, los sistemas microcontrolados son baratos y de relativa facilidad de uso para el público en general, gracias en buena parte a diversos proyectos Open Source donde el usuario puede acudir a consultar la documentación disponible sin ningún

coste. También hay que destacar que los lenguajes de programación de alto nivel facilitan enormemente la abstracción del hardware por parte del usuario, llegando a ser transparente para poder centrar sus esfuerzos en plasmar sus ideas.

Todas estas características mencionadas contribuyen a que este tipo de tarjetas se estén convirtiendo en unas plataformas muy atractivas para los desarrolladores, atrayendo incluso a colectivos ajenos a la programación como artistas, diseñadores y entusiastas.[2] [3]

## 2.1. Microcontroladores AVR

Los microcontroladores AVR, creados por la empresa Atmel en 1996, están basados en arquitectura Harvard tipo RISC *(Reduced Instruction Set Computing)* de 8 bits. Se cree que inicialmente este MCU *(Microcontroller Unit)* fue diseñado en Noruega por unos estudiantes de la empresa Nordic VLSI y que más adelante fue vendida la tecnología a la empresa Atmel. Hoy en día Atmel cuenta con 6 grupos de microcontroladores desarrollados desde el inicial:

- TinyAVR
- MegaAVR
- XMega
- Application-specific AVR
- FPSLIC (AVR with FPGA)
- AVR32

La principal característica de la arquitectura Harvard *(RISC)* consiste en un sistema con memorias físicas separadas, tanto para instrucciones como para datos. Los AVR fueron unos de los primeros microcontroladores que utilizaron la tecnología de memoria Flash como memoria de programa (instrucciones). Con este tipo de memoria no volátil y reprogramable miles de veces, se dejo atrás a los demás controladores basados en memoria Rom, Eprom y EEprom, programables una sola vez en el primer caso y reprogramables/borrables en los siguientes.

Estos microcontroladores cuentan, aparte de la memoria de programa Flash, con memoria Sram para datos y con memoria EEprom, todas On-chip. De esta forma contamos con un sistema de computo completo formado por una Cpu junto con su memoria al contrario que un microprocesador convencional. De este hecho se desprende que el uso y la complejidad de la tarea a realizar será más limitada para un microcontrolador al presentarse ya integrado.[4]

Los primeros controladores AVR creados, como el AT90S8515 con encapsulado DIP *(Dual In-line)* de 40 pines *(figura1.Izda)*, estaban basados en la micro arquitectura CISC del Intel MCS-51 o más conocido como 8051 *(figura1.Dcha)*, solamente diferenciado de este por la polaridad de aserción del Reset. Esto se debió a que era una versión RISC basada en el chip CISC original de Intel. Hoy en día sigue siendo compatible pin a pin con multitud de chips basados en el 8051 y ofrece un modelo de programación RISC más sencillo.[5]

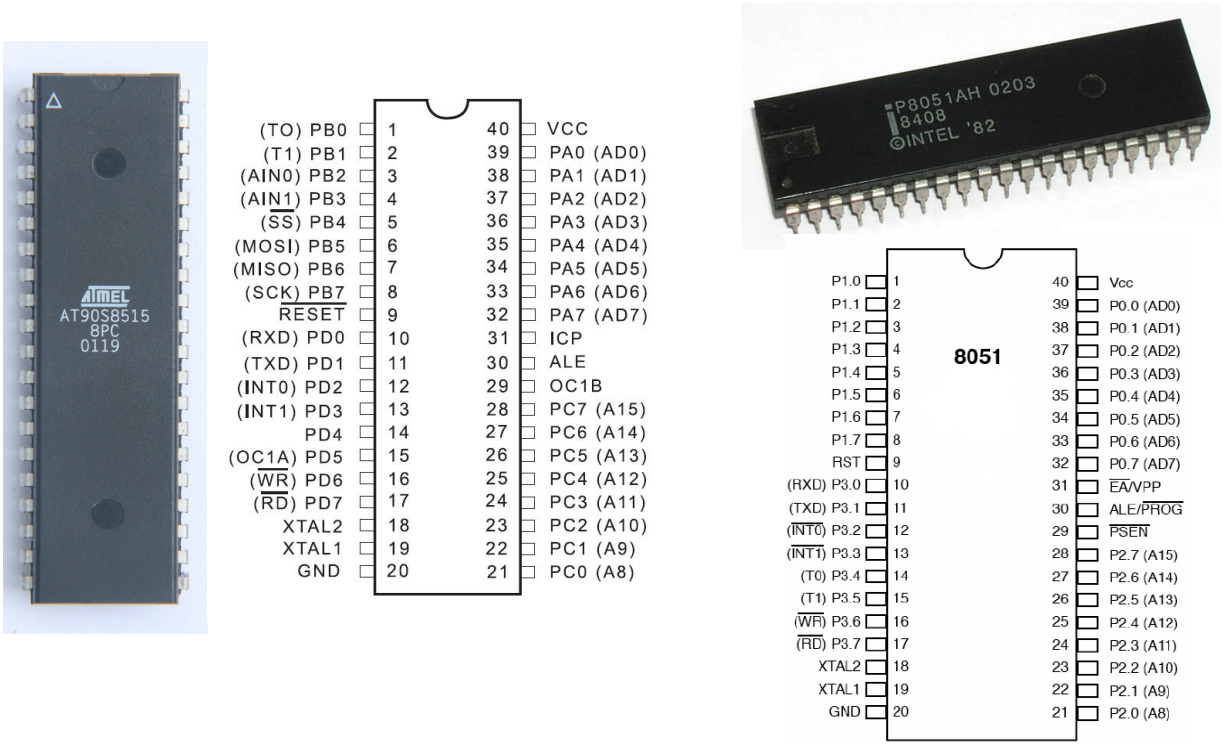



Figura 1. (Izda)Microcontrolador Atmel AT90S8515. (Dcha)Microcontrolador Intel 8051

A continuación se citan algunas de las características principales de los microcontroladores AVR:[4]

- Compatibilidad con compiladores C optimizados para esta arquitectura (HLL - High Level Language)
- Entradas/salidas bidireccionales y configurables
- Memoria de programa Flash interna y programable
- Programable in-situ mediante conexión serie
- Memoria Sram interna
- Memoria EEprom interna
- Temporizadores de 8 y 16 bits
- Conversor Analógico/digital de 10 a 12 bits
- Conversor Digital/Analógico de 12 bits

## 2.2. Qué es Arduino

Arduino es una herramienta de origen Italiano para el desarrollo de aplicaciones electrónicas de tipo Open-Source o código abierto. Principalmente, se basa en la facilidad de uso de una tarjeta hardware y de un entorno de programación, para leer y controlar multitud de magnitudes físicas de nuestro alrededor. Debido a la filosofía del proyecto, está dirigido a todo tipo de usuarios, desde programadores experimentados, hasta artistas, diseñadores y entusiastas que no hayan programado nunca.

Arduino puede ser usado para desarrollar proyectos interactivos, tomando medidas desde Switches y sensores, controlando motores, arrays de leds y un sin fin de elementos electrónicos. Además, todos los proyectos pueden ser ejecutados desde la tarjeta Arduino o también ser controlados desde un entorno en un ordenador, haciendo las aplicaciones configurables e interactivas.

Arduino ofrece multitud de tarjetas y módulos de expansión, que permiten al usuario incluir funcionalidad avanzada a sus proyectos, desde conexión de red Ethernet, hasta almacenamiento en tarjetas de memoria SD. Estas ampliaciones pueden ser adquiridas listas para funcionar, pero debido al carácter open-source del proyecto, uno mismo podría crear su propio prototipo para desarrollar una aplicación concreta.

Arduino está controlado por un microcontrolador que gobierna el sistema, este se programa en un lenguaje propio de Arduino, el cual está basado en Wiring, el cual a su vez es una modificación del lenguaje C/C++ ampliamente conocido. De esta forma se ofrece al usuario una API (Application program interface) de muy alto nivel en términos de programación. Por supuesto el entorno de programación de Arduino se puede obtener de forma gratuita.[6]

## 2.3. Por qué elegir Arduino

Existen en el mercado muchas otras plataformas basadas en microcontroladores como tarjetas PIC, Raspberry Pi, Parallax Basic Stamp, etc, que ofrecen prestaciones similares a las ofrecidas por Arduino. Sin embargo, Arduino simplifica el proceso de abstracción del microcontrolador de modo que ofrece una ventaja añadida para los usuarios menos experimentados, capacitándoles para desarrollar proyectos multidisciplinares.

Algunas de las ventajas más relevantes de Arduino son las siguientes: [6]

- *Coste reducido*. Las tarjetas Arduino son relativamente baratas comparadas con otras plataformas basadas en microcontroladores. Los módulos de expansión pueden ser incluso diseñados y montados a mano, o ser adquiridos ya montados por menos de 40€.

- *Multiplataforma (Cross-Platform)*. El software de Arduino puede ejecutarse en Windows, Linux y Mac. Otras plataformas están limitadas a ser programadas desde Windows.

- *Entorno de desarrollo sencillo y claro*. El Arduino IDE, es accesible para programadores no experimentados y suficientemente flexible y potente para usuarios avanzados.

- *Software de código abierto*. El entorno de programación se distribuye de forma gratuita y está disponible el código fuente para ser extendido por usuarios

avanzados. El lenguaje de programación puede ser expandido a través de librerías propias de C/C++ e incluso se puede hacer uso directo de las librerías estándar de AVR LibC (librerías de C para ser compiladas con GCC en controladores AVR).

- *Hardware de código abierto*. Las tarjetas Arduino están basadas en los microcontroladores Atmel Mega. Estas están licenciadas de forma pública de forma que los diseñadores electrónicos más experimentados pueden crear sus propias versiones, extendiendo sus capacidades y mejorando sus características. Incluso usuarios relativamente inexpertos, podrían construir sus diseños en una placa de prototipado sin soldadura para comprobar cómo funcionan las Arduino Boards.

## 2.4. Oferta de Arduino

El proyecto Arduino ofrece multitud de tarjetas basadas en microcontroladores AVR capaces de brindar al usuario la capacidad de programar de forma extremadamente versátil sus aplicaciones. A continuación se recoge un resumen con las principales tarjetas SBC y algunas posibles expansiones desarrolladas por Arduino:

### 2.4.1. Tarjeta Arduino UNO rev3

En primer lugar tenemos la ***Arduino UNO Rev3***. Este es el diseño de referencia de Arduino, es una tarjeta de propósito general a la que podremos acoplar sin ninguna dificultad cualquiera de las expansiones disponibles.[7]

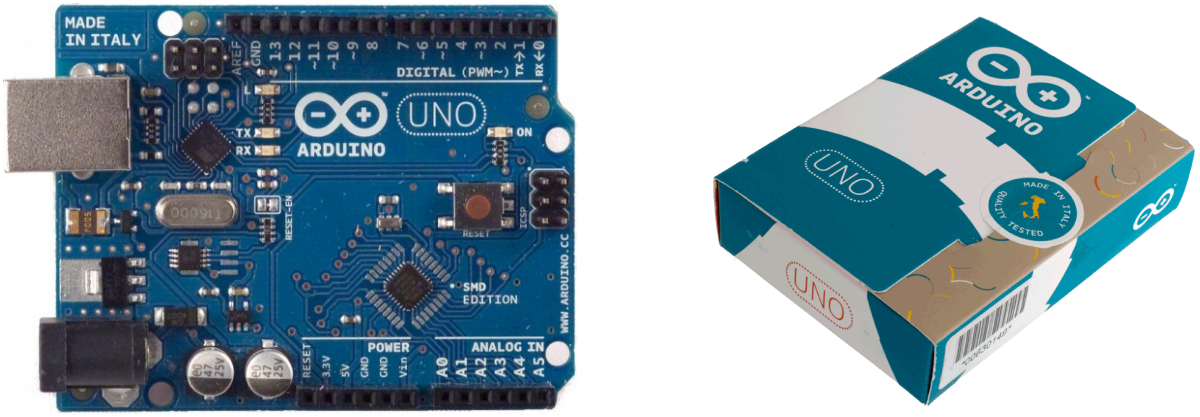



Figura 2. Vista frontal Arduino UNO board edición SMD.

Sus características son:

- SBC basada en microcontrolador AVR AtmelMega328.
- Alimentación entre 7-12V, suministrado desde Usb o desde fuente externa.
- Voltaje de operación de 5V.
- 14 Pines I/O digitales.

- 6 pines digitales con salida PWM opcional (Pulse Width Modulation).
- 6 pines I/O analógicos.
- Memoria Flash de 32KB.
- Memoria Sram de 2KB.
- Memoria EEprom de 1KB.
- Velocidad de reloj de 16MHz.
- Programación mediante conexión USB.
- Soporta 2 tipos de interrupciones.
- Soporta protocolos SPI e I2C.

### 2.4.2. Tarjeta Arduino MEGA 2560

En segundo lugar mencionar una tarjeta más potente, ***Arduino MEGA 2560***. Esta es una tarjeta más completa que la de referencia, ofreciendo mayores pines de entrada/salida y una capacidad de memoria más alta. [7]

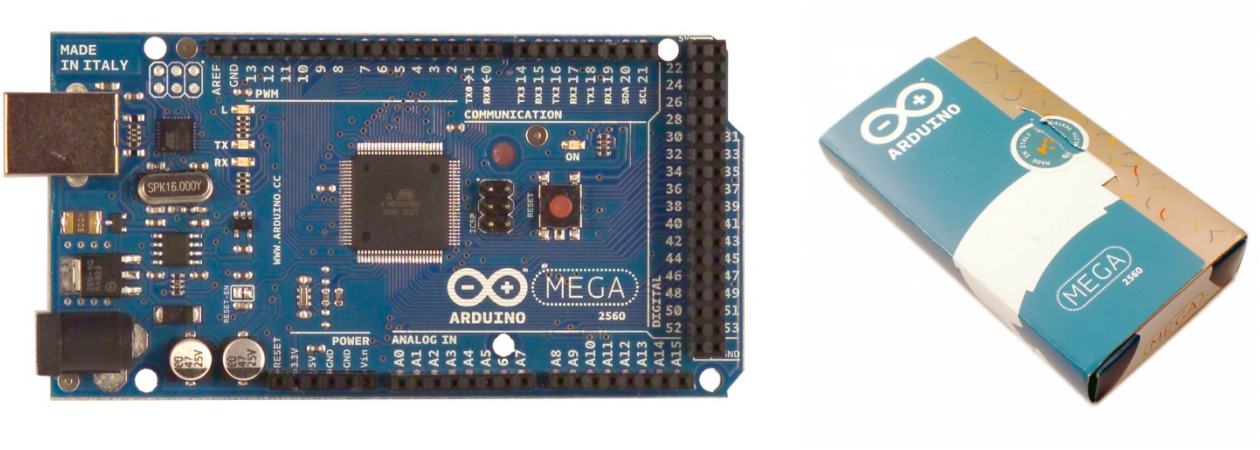



Figura 3. Vista frontal Arduino MEGA 2560 board.

Sus características son:

- SBC basada en microcontrolador AVR AtmelMega2560.
- Alimentación entre 7-12V, suministrado desde Usb o desde fuente externa.
- Voltaje de operación de 5V.
- 54 Pines I/O digitales.
- 15 pines digitales con salida PWM opcional (Pulse Width Modulation).
- 16 pines I/O analógicos.
- Memoria Flash de 256KB.
- Memoria Sram de 8KB.
- Memoria EEprom de 4KB.
- Velocidad de reloj de 16MHz.
- Programación mediante conexión USB.
- Soporta 6 tipos de interrupciones.
- Soporta protocolos SPI e I2C.

Como vemos, Arduino ofrece una conjunto de tarjetas de propósito general como las mencionadas aquí y otras muchas. Para incrementar la potencia de estas, además se ofertan varios módulos o expansiones (conocidas en la comunidad Arduino como ***Shields***) que pueden ser conectadas a las tarjetas base. Estos Shields se conectan en cascada verticalmente hacia arriba *(figura 4)*, de forma que la distribución de pines encaja correctamente con la tarjeta base.

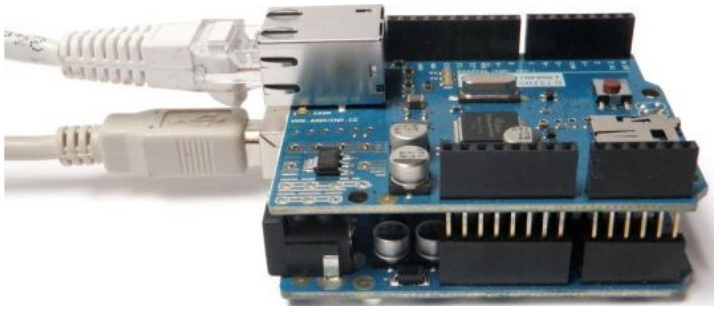


Figura 4. Shield o expansión de Arduino sobre la tarjeta de referencia.

Estas ampliaciones de la tarjeta base, se comunican con esta mediante el protocolo SPI *(Serial Peripheral Interface Bus)* y por lo tanto se produce una disminución de los pines de entrada/salida disponibles para el usuario ya que estarán ocupados para la comunicación. Sin embargo, dependiendo de la tarjeta base adquirida, la perdida de pines I/O no es especialmente relevante ya que agrega otro tipo de funciones avanzadas.

El diseñador de proyectos deberá hacer un estudio previo para conocer de antemano las necesidades totales de recursos y así poder adquirir el sistema más optimo para él. En capítulos posteriores se planteara una solución externa para ampliar las líneas de entrada/salida utilizando multiplexores.

### 2.4.3. Ethernet Shield de Arduino

Una de las expansiones más interesantes de la plataforma Arduino es la ***Ethernet Shield***. Esta Shield permite conectar nuestra tarjeta Arduino en red local o a internet con unos sencillos comandos de configuración. Además cuenta con un zócalo para utilizar una tarjeta de memoria micro SD para tener espacio de almacenamiento adicional. [7]

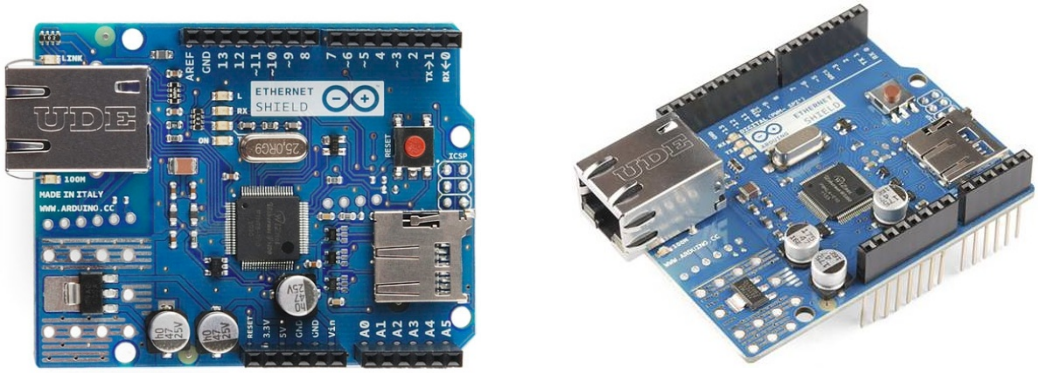



Figura 5. Ethernet shield de Arduino.

Sus características principales son:

- Compatible con tarjetas Arduino UNO y MEGA.
- Basado en el controlador ethernet W5100.
- Velocidad de conexión de 10/100 Mb/s.
- Soporta 4 conexión simultaneas en modo TCP.
- Soporta comunicación UDP.
- Voltaje de operación de 5V .
- Comunicación con tarjetas Arduino mediante bus SPI.
- Cuenta con conector RJ45 (ethernet) y slot para micro SD.
- Posibilidad de añadir accesorio PoE (Power over ethernet - Alimentación mediante el cable de red ethernet).

### 2.4.4. Utilización de un reloj en tiempo real

Otra expansión muy interesante del sistema Arduino es la inclusión de un ***reloj en tiempo real (Real Time Clock, RTC)*** en nuestro proyecto. Con él, podemos registrar perfectamente los eventos de nuestro proyecto con la fecha y la hora en la que se produzcan. Es una excelente opción para realizar un sistemas registrador de datos *(datalogger)* en el cual es de vital importancia controlar los tiempos.

El RTC más famoso usado con Arduino es el basado en el chip DS1307 *(figura 6.Dcha)*. Este se comunica con la tarjeta principal mediante el protocolo I2C *(Inter-Integrated Circuit)* soportado por Arduino. Debido a que es un complemento externo conectado por I2C, esto nos limitara el numero de pines analógicos disponibles en la tarjeta. Esto se produce porque la conexión I2C requiere del uso dedicado de 2 líneas de comunicación.

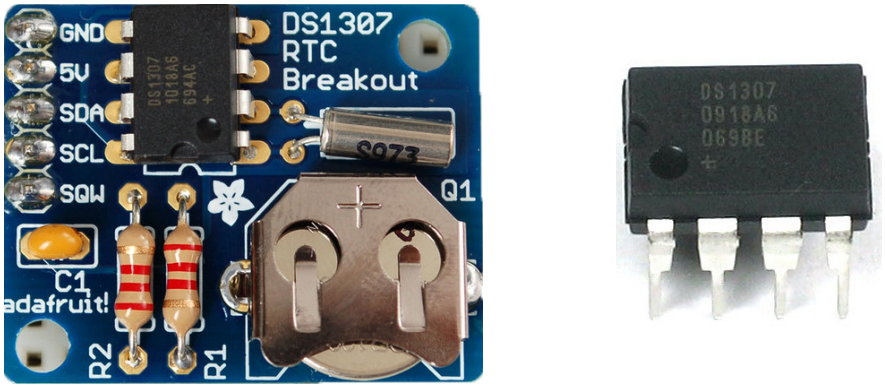



Figura 6. (Izda)Pcb distribuido pre-soldado comercialmente. (Dcha)Chip DS1307

Hay que resaltar que este modulo se puede obtener por piezas para ser soldado por el usuario, o por el contrario adquirir un Shield que lo incorpora *(figura 6.Izda)*. En el primer caso el usuario lo colocara al lado de la tarjeta pero sin la opción de incrustarlo verticalmente en ella. También se puede hacer el montaje en una placa de prototipado sin soldadura para comprobar su funcionalidad. En el caso de adquirir el Shield, este se acoplara de forma más compacta pero igual de funcional. [8]

Algunas de las características más importantes son las siguientes:

- Basado en el RTC DS1307 de bajo coste y contaje de años.
- Contaje ininterrumpido con una pila de 3.3V de respaldo.
- DS1307 requiere pocos componentes discretos para su montaje y operación.
- Soporte de librerías de programación mediante Arduino.

## 2.5. Otras plataformas

Como se ha mencionado antes, hoy en día en el mercado podemos encontrar multitud de plataformas con propósitos parecidos a los de Arduino, en muchos casos más avanzados. En este apartado, se intentara hacer una breve descripción de las características que podemos encontrar en la competencia, teniendo en cuenta unas características similares en potencia y precio.

### 2.5.1. Tarjetas MicroChip

La primera plataforma a mencionar es la basada en microcontroladores PIC. Diseñados por la empresa ***MicroChip***, podemos encontrar a través de la red una gran variedad de tarjetas similares a las ofrecidas por Arduino. Los microcontroladores PIC son rivales directos de los Atmel AVR debido a sus características y precios que pueden rondar en ambos casos por debajo de los 10€ en su gama más alta.

En la web Modtronix Engineering podemos adquirir por menos de 50€ tarjetas basadas en PICs con funcionalidades como conexión Ethernet, manejo de LCDs, comunicación serie, etc. Un ejemplo concreto puede ser la tarjeta ***SBC44UC***. [9]

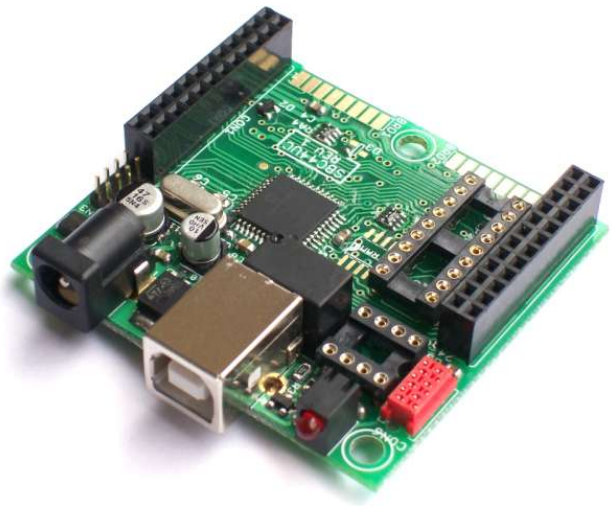


Figura 7. Tarjeta basada en microcontrolador PIC, SBC44UC

Las principales características son las siguientes:

- SBC basada en el PIC 18F4550 de gama alta.
- Interfaz de programación USB.
- Alimentación entre 9-12V, suministrado desde USB o con fuente externa.
- Entorno gratuito de programación Microchip Mplab.
- 33 entradas/salidas programables de propósito general.
- 13 entradas programables con conversión analógico-digital de 10 bits.
- 2 entradas programables con salida PWM.
- Soporte para protocolos I2C y SPI.
- Memoria Flash de 32KB.
- Memoria Sram de 2KB.
- Memoria EEprom de 256 Bytes.
- Posibilidad de acoplar tarjetas de expansión similar a Arduino.

Como se observa, la tarjeta descrita arriba no cuenta con el hardware necesario para la conexión Ethernet. Eso es porque es una tarjeta mas de propósito general en cuanto al desarrollo de aplicaciones. Por supuesto, hay una serie de tarjetas que incluyen estas funciones y se ofertan en la web mencionada arriba.

La serie ethernet cuenta con los modelos SBC65EC, SBC66EC y SBC68EC, todos con hardware ethernet, entradas/salidas analógicas y digitales, RTC y web server pre configurado.

Figura 8. Tarjeta basada en microcontrolador PIC, con conexión Ethernet, SBC65EC

A continuación se describen la principales características del modelo ***SBC65EC***: [10]

- SBC basada en el PIC 18F6627 de gama alta.
- Alimentación entre 7-35V.
- Conexión ethernet 10/100 Mbps.
- Acepta comandos HTTP y UDP.
- 32 entradas/salidas digitales.
- 12 entradas programables con conversión analógico-digital de 10 bits.
- 4 entradas programables con salida PWM.
- Soporte del protocolo I2C.

- Programado con la librería SBC65EC Modtronix web server sobre ethernet.
- Control y configuración desde web server.
- Memoria Flash de 98KB.
- Memoria Sram de 4KB.
- Memoria EEprom de 64KB.
- Posibilidad de acoplar tarjetas de expansión similar a Arduino.

Una expansión muy común de las tarjetas de MicroChip es la tarjeta de prototipado sin soldadura. Con ella disponemos de una zona amplia de conexiones para utilizar los dispositivos externos que necesitemos. [9]

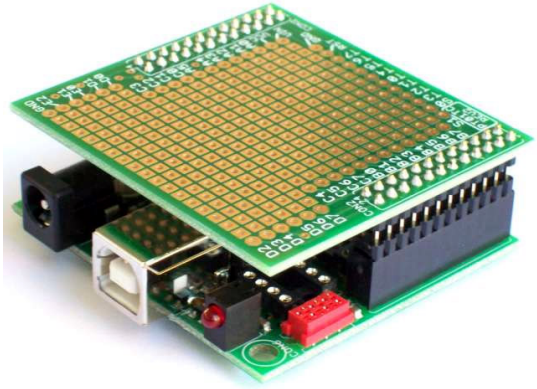


Figura 9. Expansión de prototipado compatible con las tarjetas SBC44UC

### 2.5.2. Tarjetas Raspberry Pi

Para continuar describiendo las plataformas alternativas a Arduino, una parada obligada es la tarjeta ***Raspberry Pi***. Hay que destacar que esta plataforma ofrece funcionalidad avanzada superior a la Arduino pero conviene mencionarla por su creciente popularidad en el ámbito de las SBCs.

Raspberry Pi es una minicomputadora del tamaño de una tarjeta de crédito capaz de conectar con una televisión digital y con un teclado. Se podría ver como un pequeño PC que puede ser usado para muchas de las aplicaciones que nuestro PC de escritorio puede hacer. Podemos utilizar hojas de cálculo, procesadores de texto, videojuegos y ver video de alta definición. [11]

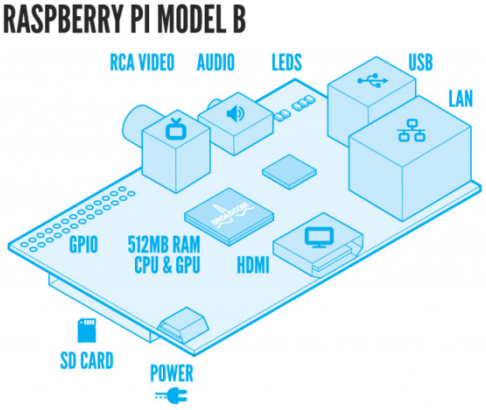



Figura 10. Detalle de puertos de la tarjeta Raspberry Pi modelo B

Se ofertan dos modelos, Modelo A y B con un precio máximo alrededor de 35€. Los demás elementos como la fuente de alimentación o la tarjeta SD deben ser comprados por separado. Algunas de las características generales son las siguientes:

- Modelo A tiene 256MB de RAM, 1 puerto Usb y no tiene Ethernet.
- Modelo B tiene 512MB de RAM, 2 puertos Usb y ethernet.
- Las medidas de la tarjeta son 85.60mm x 56mm x 21mm y su peso de 45 gramos.

El procesador encargado de gobernar estas tarjetas es el ***Broadcom BCM2835***. Este procesador, es idóneo para tareas multimedia de gran rendimiento en aplicaciones incrustadas y móviles como la tarjeta Raspberry Pi. Esta diseñado y optimizado para tener un bajo coste y una eficiencia energética importante. Las principales características son las siguientes: [12]

- Basado en el procesador ARM1176JZ-F de bajo consumo
- Basado en el procesador multimedia VideoCore IV de doble núcleo
- Reproducción a 1080p, compresión H264

Los procesadores ARM *(Advanced RISC machine)* son una arquitectura tipo RISC *(reduced instruction set computing)* de 32 bits desarrollada por ARM Holdings. Su relativa simplicidad los hace ideales para aplicaciones de baja potencia y por ello se han hecho fuertes en el mercado de la electrónica móvil e integrada.

En 2005, alrededor del 98% de los más de mil millones de teléfonos móviles vendidos cada año utilizan al menos un procesador ARM. Desde 2009, los procesadores ARM son aproximadamente el 90% de todos los procesadores RISC de 32 bits integrados y se utilizan ampliamente en la electrónica de consumo, incluyendo PDA, tabletas, Teléfono inteligente, teléfonos móviles, videoconsolas portátiles, calculadoras, reproductores digitales de música y medios (fotos, vídeos, etc.), y periféricos de ordenador como discos duros y routers. [13]

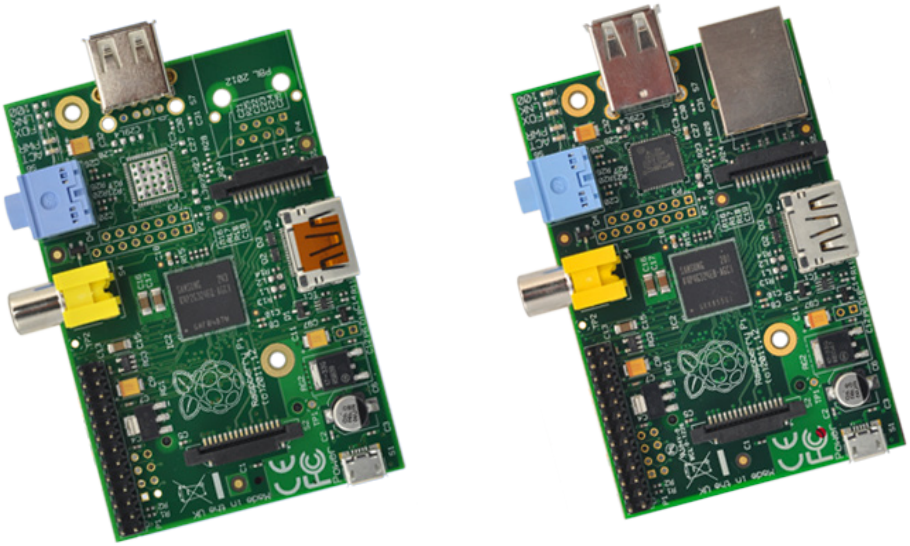


Figura 11. (Izda)Tarjeta Raspberry Pi modelo A. (Dcha)Tarjeta Raspberry Pi modelo B

A continuación se presenta una tabla con las principales características de los 2 modelos ofertados por Raspberry Pi, modelo A y B: [14]

| ***Características técnicas*** | ***Modelo A*** | ***Modelo B*** |
|---|---|---|
| Chip | Broadcom BCM2835 SoC full HD multimedia applications processor | Broadcom BCM2835 SoC full HD multimedia applications processor |
| CPU | 700 MHz Low Power ARM1176JZ-F Applications Processor | 700 MHz Low Power ARM1176JZ-F Applications Processor |
| GPU | Dual Core VideoCore IV® Multimedia Co-Processor | Dual Core VideoCore IV® Multimedia Co-Processor |
| Memoria | 256MB SDRAM | 512MB SDRAM |
| Ethernet | No tiene | onboard 10/100 Ethernet RJ45 jack |
| Usb 2.0 | Single USB Connector | Dual USB Connector |
| Salida de video | HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC) | HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC) |
| Salida de audio | 3.5mm jack, HDMI | 3.5mm jack, HDMI |
| Almacenamiento | SD, MMC, SDIO card slot | SD, MMC, SDIO card slot |
| Sistema operativo | Linux | Linux |
| Dimensiones tarjetas | 8.6cm x 5.4cm x 1.5cm | 8.6cm x 5.4cm x 1.7cm |

Tabla 1. Características de las tarjetas Raspberry Pi

Como se observa en el resumen de características, esta plataforma ofrece unas capacidades avanzadas de procesamiento debido a la potencia de su CPU y GPU. Cabe recordar que el uso de un procesador no es el más adecuado por el hecho de que no ofrece la capacidad de utilizar pines de entrada/salida como un microcontrolador. Sin embargo, es una plataforma lo suficientemente interesante como para nombrarla aquí.

# Capítulo 3

# Hardware utilizado

A partir de este capítulo nos centraremos en los detalles del proyecto, explicando en este apartado todo lo relacionado al hardware utilizado. Por hardware se entiende la tarjeta Arduino utilizada y todos los elementos externos que ampliaran la funcionalidad de esta.

Como se expuso en la introducción, la finalidad del proyecto es la de diseñar un sistema registrador de datos o datalogger mediante el uso de la plataforma Arduino. Teniendo en cuenta que las necesidades del proyecto no requieren de una gran potencia, la principal opción es utilizar la Arduino UNO rev3, la cual no brinda pines I/O suficientes para recoger las mediciones y para conectar los dispositivos externos que se necesiten.

Las principales carencias de la tarjeta UNO rev3 es la falta del hardware ethernet y del zócalo para la tarjeta de memoria micro SD. Por este motivo se hace necesario la adquisición del Ethernet Shield o del Arduino Ethernet Board rev3, que combina en una sola tarjeta la Arduino UNO con un Ethernet Shield ya integrado. Las dos opciones son perfectamente validas ya que nos proporcionan exactamente las mismas funcionalidades. Para optimizar al máximo el presupuesto del proyecto, la mejor opción es obtener directamente la *Arduino Ethernet Board rev3.*

## 3.1. Arduino Ethernet Board rev3

La ***Arduino Ethernet Board rev3*** es una tarjeta basada en el microcontrolador AVR ATMega328 que combina las características de la UNO rev3 y del ethernet Shield todo en la misma tarjeta. Cuenta con 14 pines digitales de entrada/salida, 6 pines analógicos, oscilador a 16MHz, conector ethernet RJ45, conector de alimentación para fuente externa, conector ICSP de programación y botón de reset manual.

Las especificaciones generales de la Ethernet Board rev3 son las siguientes:

- Basado en el microcontrolador ATMega328
- Conexión ethernet mediante WZ5100 controlador TCP/IP
- Slot para tarjeta micro SD totalmente compatible
- Voltaje de alimentación de 7-12V
- Voltaje de operación de 5V
- 14 pines I/O digitales
- 4 pines I/O digitales con PWM (pin3, pin5, pin6, pin9)
- 6 pines I/O analógicos

- Memoria Flash de 32KB
- Memoria Sram de 2KB
- Memoria EEprom de 1KB
- Velocidad oscilador de 16MHz
- Soporta comunicación mediante protocolo SPI e I2C

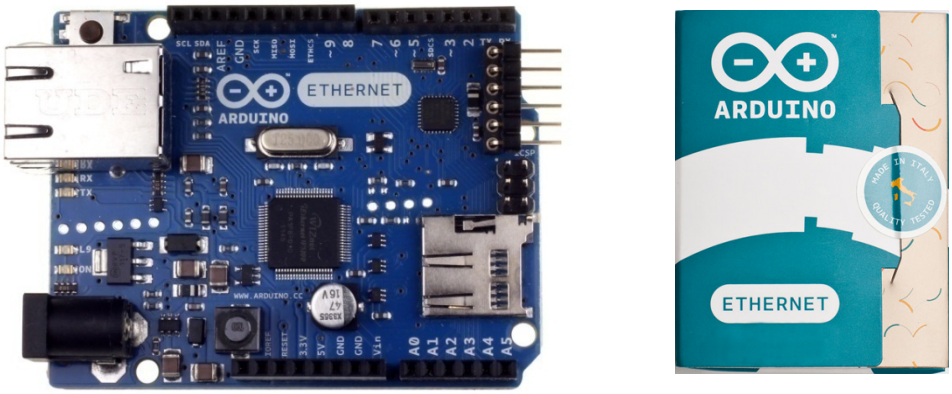



Figura 12. Tarjeta Arduino Ethernet Board rev3

El ethernet está gobernado por el chip *Wiznet WZ5100*, el cual controlara todas las conexiones de red de la tarjeta. Para poder activar el WZ5100, debemos hacer uso de un pin de selección *(slave select)*, que en esta tarjeta es el pin digital 10. Debido a la comunicación SPI entre la tarjeta y el WZ5100, quedaran reservados 4 de los pines I/O digitales, quedando así 10 pines I/O digitales.

Para el manejo del almacenamiento en la tarjeta de memoria micro SD, también se deberá hacer uso de un pin de selección *(slave select)*. En nuestra tarjeta es el pin digital 4, por lo que tampoco podremos usarlo para otro propósito quedándonos así 9 pines I/O digitales disponibles.

Recordar que la tarjeta micro SD y el ethernet están conectados al mismo bus SPI para enviar y recibir datos con el microcontrolador. En la revisión 3 de la tarjeta, esto no produce ningún choque entre ambos dispositivos debido al control del protocolo que hace Arduino, pudiendo funcionar a la vez sin problemas.

Los pines utilizados por la comunicación SPI son los siguientes:

- Pin digital 4 - tarjeta micro SD slave select
- Pin digital 10 - ethernet chip slave select
- Pin digital 11 - SPI MOSI (Master output, Slave input)
- Pin digital 12 - SPI MISO (Master input, Slave output)
- Pin digital 13 - SPI SCK (Serial clock)

La tarjeta podrá ser alimentada desde una fuente de alimentación externa, desde el modulo *Power over ethernet (PoE)* o desde el cable de programación *FTDI (Future Technology Devices International) (figura 13.Dcha).* La alimentación con fuente externa tendrá que poseer un conector con el positivo central de 2.1mm para poder conectarlo

al jack de la tarjeta. También se podrían usar una serie de pilas conectadas a los pines de alimentación Vin y GND de la tarjeta.

La alimentación sobre el cable ethernet es una característica de esta tarjeta que puede ser utilizada para no tener que comprar fuentes externas. El modulo PoE viene ensamblado en la tarjeta de forma permanente y es capaz de extraer la potencia del cable ethernet categoría 5 *(figura 13.Izda)*. En este proyecto no se utilizara la alimentación PoE.[15]

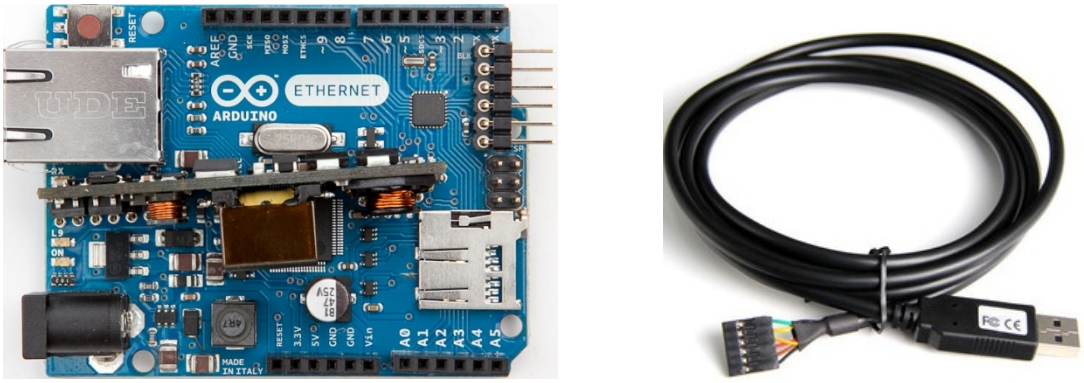



Figura 13. (a)Arduino Ethernet Board con modulo PoE. (b)Cable programación FTDI

El modo de programación de la Arduino Ethernet Board rev3 es diferente al usado por la UNO rev3. En nuestra tarjeta podemos cargar los programas utilizando el conector de 6 pines de tipo serie o podemos utilizar un programador externo. En nuestro caso utilizaremos el conector de 6 pines.

Este conector de programación serie de 6 pines es compatible con el cable FTDI USB 232R de 3.3V. Es un cable de conversión Usb a serie, que soporta el reset automático después de programar, lo cual evita que el usuario tenga que hacer un reset manual para iniciar el código cargada en la tarjeta. Por otra parte, este cable de conexión con el Pc nos permite alimentar la tarjeta sin tener que utilizar una fuente externa.

A continuación se describe la polaridad del cable FTDI USB para que sea correctamente conectado a nuestra Ethernet Board, ya que el conector podría ser físicamente conectado de dos formas posibles. La conexión es la siguiente: [16]

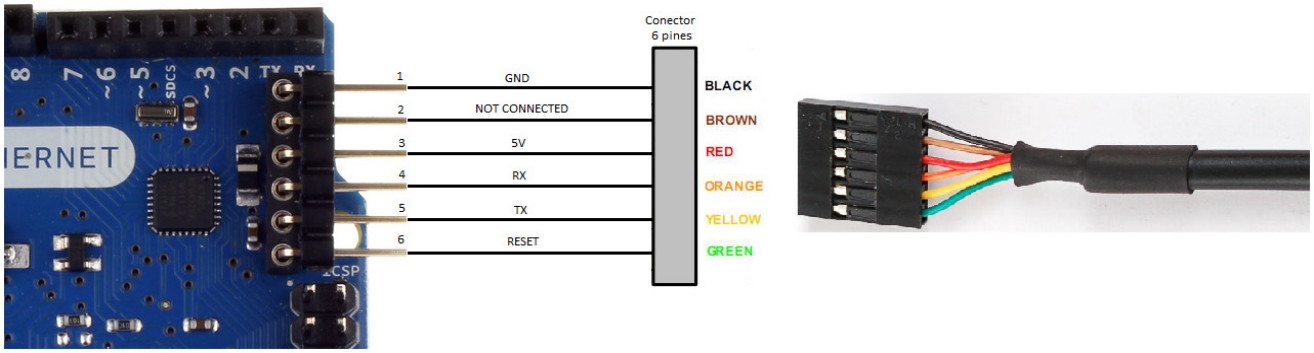



Figura 14. Detalle de la conexión del cable de programación FTDI con la Arduino Ethernet Board

## 3.2. Hardware externo necesario

Una vez escogida la tarjeta Arduino más conveniente para el proyecto, se va ha hacer una descripción detallada de los demás componentes que son necesarios. Para realizar un registrador de datos, en primer lugar necesitamos la recogida de alguna magnitud física a monitorizar. En nuestro caso monitorizamos la temperatura en distintos puntos, por lo que vamos a necesitar algún sensor de temperatura que sea fácilmente utilizable con la tarjeta Arduino.

Todos los registros de temperaturas que hagamos, deberán ser almacenados de forma compacta y segura en un dispositivo de almacenamiento masivo. Para ello, contamos con el zócalo para tarjeta micro SD, la cual nos permite almacenar una inmensa cantidad de datos dependiendo de la capacidad de esta. Hoy en día, por el precio que tienen las tarjetas SD, vamos a adquirir una de 8 GB de capacidad.

En un sistema registrador de datos, se puede decir que es vital saber cuándo se han producido las medidas, y de esta forma poder hacer análisis posteriores de máximos, mínimos, medias y tendencias en periodos concretos. Para ello, se hace imprescindible la utilización de un Real time clock (RTC) que nos va a permitir el registro exacto de fecha y hora de cada medida. Este modulo se va a comprar por piezas y será ensamblado por nosotros mismos.

Como se ha comentado en capítulos anteriores, las necesidades de la tarjeta Arduino hacen que varios pines I/O de propósito general queden reservados. Para solucionar este posible contratiempo con el número de entradas disponibles, se hace uso del multiplexado de las entradas I/O. Utilizando un multiplexor de propósito general junto con la tarjeta Arduino, elevamos la cantidad de entradas o puntos de medida del sistema.

En resumen, el hardware externo utilizado es:

- Sensores de temperatura integrados, para recoger las medidas de los puntos deseados.
- Tarjeta de memoria micro SD, para almacenar todas las medidas realizadas.
- Reloj de tiempo real, para documentar con la fecha y hora cada medida.
- Multiplexor, para ampliar el número de entradas I/O por si fuesen pocas.

## 3.3. Sensores de temperatura

Para realizar las mediciones de temperatura, se han utilizado sensores analógicos integrados de bajo coste, gran rango de medida y alta precisión, en concreto dos modelos, TMP36 y LM35. Estos sensores son muy parecidos en funcionamiento, no necesitan de calibración previa y solo hay que realizar una conversión mediante programa para convertir la señal analógica en una temperatura Celsius.

En primer lugar, el sensor ***TMP36GZ de Analog Devices*** utilizado, consta de 3 pines y tiene un encapsulado TO-92 *(transistor outline package case style 92)* de plástico. Produce una tensión de salida linealmente proporcional a la temperatura ambiente

Celsius. Requiere una tensión de alimentación de entre 2.7-5.5V, la cual se la podemos dar desde nuestra tarjeta Arduino sin ningún problema. [17]

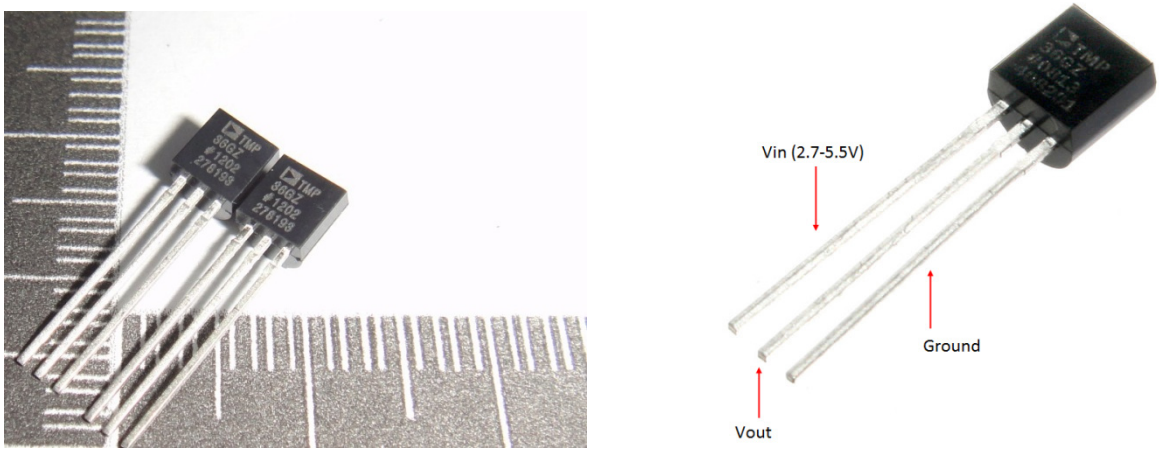


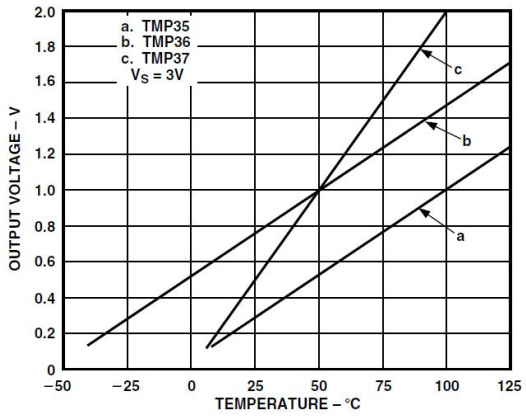

Figura 15. Sensor de temperatura integrado Tmp36

Sus principales características técnicas son las siguientes:

- Baja tensión de alimentación 2.7-5.5V
- Calibrado directamente en grados Celsius
- Factor de escala de 10mV/°C
- Precisión de -+2°C
- Rango de temperaturas de operación -40°C a +125°C
- Rango de tensión de salida 0.1V (-40°C) a 2V (+125°C)

Conectando el pin central del TMP36 en una entrada analógica de la tarjeta Arduino, recibimos la señal en mili voltios proporcional a la temperatura ambiente medida. Esta señal es independiente de la señal de alimentación. Desde la programación de Arduino, debemos realizar un pequeño calculo para convertir los mili voltios en grados Celsius.

$$Temp\_Tmp36\ (°C) = (Vout\ (mV) - 500) / 10$$

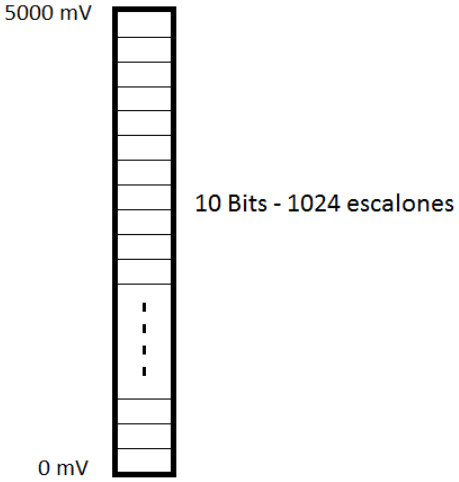



*TPC 1. Output Voltage vs. Temperature*

Figura 16. Grafica de relación entre tensión de salida y temperatura medida para el Tmp36

Cabe destacar que las entradas analógicas de la tarjeta Arduino, poseen un conversor ADC *(Analog to digital converter)* de 10 bits de precisión, lo cual es resolución suficiente para mediciones no demasiado exigentes como las de este proyecto. De este modo, resulta interesante indicar el grado de exactitud que ofrece una medida a través de una entrada analógica. [18]

En este caso alimentamos nuestro sensor con 5V y tenemos una precisión de 10 bits:




$$Tamaño\ escalon = \frac{5000}{1024} = 4.882\ \frac{mV}{escalon}$$

$$Teniendo\ por\ ejemplo\ lectura\ analogica = 150$$

$$Tension\ (mV) = 150\ esc \cdot 4.882 \frac{mV}{esc} = 732.3\ mV$$

$$Temp\_Tmp36(°C) = \frac{732.3-500}{10} = 23.23°C$$

Figura 17. Ejemplo de cálculo de una medida a través del conversor ADC de la tarjeta Arduino

En segundo lugar se van a utilizar los sensores ***LM35DZ de National Semiconductor***. Estos tienen características muy similares a las del TMP36 como el encapsulado TO-92 de plástico y 3 pines de conexión. También nos da a su salida una tensión en mili voltios linealmente proporcional a la temperatura medida. *Hay que indicar que para usarlo en mediciones de temperaturas negativas, debe tener alimentación simétrica (+Vs -Vs) y una resistencia externa adicional.* [19]

Sus características técnicas más importantes son las siguientes:

- Tensión de alimentación 4-30V
- Calibrado directamente en grados Celsius
- Factor de escala de 10mV/°C
- Precisión de 0.5°C (25°C)
- Rango de temperaturas de operación básica +2°C a +150°C (+Vs Gnd)
- Rango de temperaturas de operación avanzada -55°C a +150°C (+Vs -Vs)
- Rango de tensión de salida -0.55V (-55°C), 0.25V (25°C) y 1.5V (+150°C)

Una vez recibida la señal en mili voltios del sensor LM35 debemos hacer una conversión mediante programa para ajustar el valor a grados Celsius. Utilizaremos la siguiente fórmula: [18]

*Temp_Lm35 (°C) = Vout (mV) / 10*

Aquí se muestra el esquema de montaje y conexión de los sensores con la tarjeta:

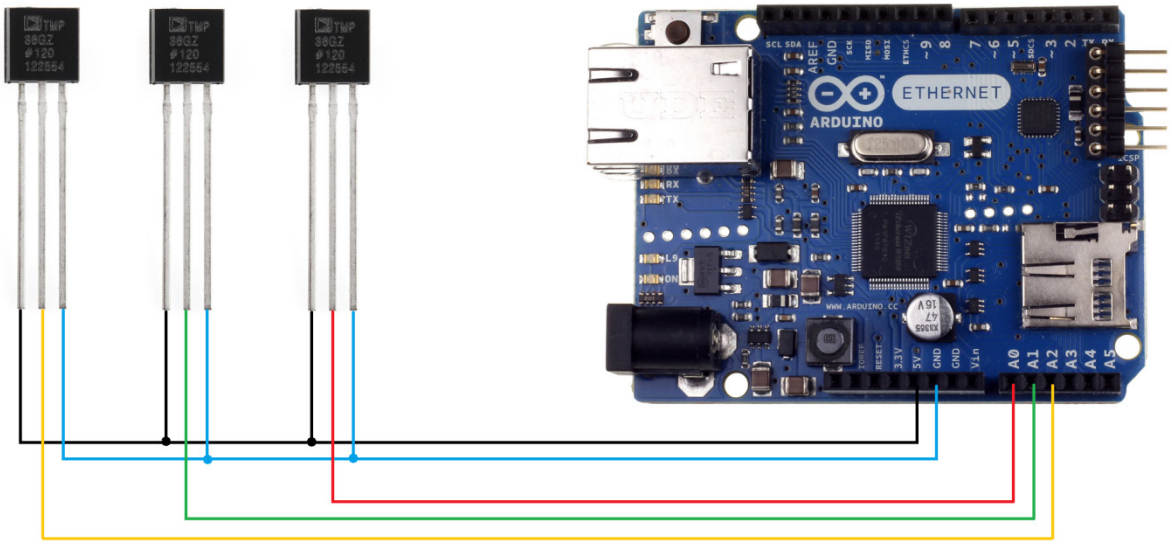



Figura 18. Montaje de 3 sensores Tmp36 con la tarjeta Arduino Ethernet Board

### 3.3.1. Ejemplo básico de programación

A continuación veremos cuál puede ser una programación sencilla para utilizar nuestros sensores de temperatura TMP36 y LM35. Antes de nada, se muestra cual es la estructura básica de un programa en la plataforma Arduino, la cual debe tener siempre dos funciones especificadas.

```
// Inclusión de librerías

void setup()
{
  // Inicialización de variables
  // Inicialización de protocolos
  // Comprobaciones iniciales
}

void loop()
{
  // Código principal
  // Llamadas a funciones
}
```

Como podemos ver, la estructura no precisa de la función main() típica y necesaria de cualquier programa en C. Esta se sustituye por dos funciones muy descriptivas llamadas *setup()* y *loop()*.

Como indican sus nombres, cada una tiene una función específica. En la función *setup()* debemos inicializar los protocolos utilizados como la comunicación serie, el I2C o la tarjeta de memoria micro SD.

Por otra parte, tenemos la función *loop()* que se ejecuta de forma indefinida, en ella debemos escribir el código principal del programa, que será ejecutado cíclicamente mientras la tarjeta este alimentada. [20]

Figura 19. Estructura básica de un Sketch de Arduino

Como primer paso para utilizar nuestros sensores de temperatura junto con la tarjeta Arduino, se muestra a continuación el código básico para leer uno de ellos:

```
void setup()
{
 Serial.begin(9600);
 pinMode(A0,INPUT);
}

void loop()
{
 int sensor;

 sensor = analogRead(A0);

 Serial.print("TMP36: ");
 Serial.println(sensor);

 delay(2000);
}
```

Como se observa, en la función setup() inicializamos la comunicación serie entre la tarjeta Arduino y el Pc. También establecemos el pin analógico A0 como entrada de señal. Si no hiciésemos esto la lectura seria errónea.

En la función loop(), utilizamos la función *analogRead()* para recoger la medida dentro de una variable tipo *int*. Más adelante se utiliza la función *print()* aplicada sobre la comunicación serie para mostrar por el terminal la medida.

Como la función loop() se ejecuta de forma cíclica, establecemos un delay que realice la medida cada 2 segundos. Esta medida no nos proporciona demasiada información, ya que solo es la conversión analógica-digital de 10 bits de Arduino.

Figura 20. Estructura básica de un Sketch de Arduino

En la figura 21 se observa las conversiones para el TMP36 y el LM35: [18]

```
//Código para Tmp36
void setup()
{
 Serial.begin(9600);
 pinMode(A0,INPUT);
}

void loop()
{
 int sensor;
 float mvolt, grados;

 sensor = analogRead(A0);
 mvolt = ((float)sensor * 5000) / 1024;
 grados = (mvolt - 500) / 10;

 Serial.print("TMP36-temp: ");
 Serial.println(grados);

 delay(2000);
}
```

```
//Código para Lm35
void setup()
{
 Serial.begin(9600);
 pinMode(A0,INPUT);
}

void loop()
{
 int sensor;
 float mvolt, grados;

 sensor = analogRead(A0);
 mvolt = ((float)sensor * 5000) / 1024;
 grados = mvolt / 10;

 Serial.print("LM35-temp: ");
 Serial.println(grados);

 delay(2000);
}
```

Figura 21. (Izda)Conversión a grados para el Tmp36. (Dcha)Conversión a grados para el Lm35

## 3.4. Multiplexores

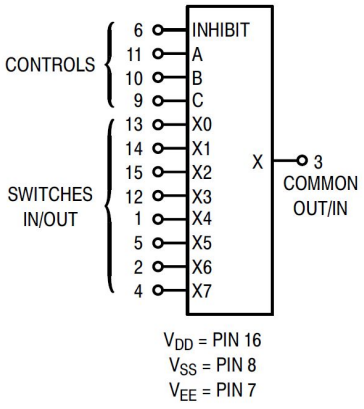
Como ya se comento anteriormente, el uso de varios de los recursos de la tarjeta Arduino requieren de comunicación especial con el microcontrolador. Los protocolos utilizados son el SPI e I2C, que van a reservar varias líneas analógicas y digitales para ellos.

Concretamente para nuestro proyecto, se hacen mucho mas importantes los pines I/O analógicos ya que debemos ofrecer varios puntos de medida de la temperatura. Aquí entra en cuestión el protocolo I2C, utilizado para conectar el reloj en tiempo real (RTC).

Este protocolo I2C precisa de dos conexiones a pines analógicos de la tarjeta Arduino para funcionar correctamente. Teniendo en cuenta que la Arduino ethernet Board rev3 posee 6 pines I/O analógicos, nos quedan libres 4 de ellos. Dependiendo de la aplicación a desarrollar, puede ser suficiente tener 4 puntos de medida con 4 sensores, pero quizás sean pocos en otro caso.

Para solventar este posible contratiempo, se propone la multiplexación de una de las entradas analógicas de la tarjeta. Esto es, utilizando un multiplexor de propósito general controlado desde programa, podamos conectarle mas sensores que serán leídos de forma escalonada. De esta forma, se elevan considerablemente los puntos de medida disponibles y evitamos así tener que adquirir una tarjeta con mayor número de entradas.

El multiplexor que vamos a utilizar es el ***MC14051BCP de ON Semiconductor***, con encapsulado PDIP *(Plastic Dual in line Package)* de 16 pines y 8 canales de entrada. Con el podemos recibir las señales de 8 sensores y procesarlas a través de una sola entrada analógica. Podemos ver la tabla de verdad y la distribución de pines del multiplexor: [21]




| Control Inputs | | | | ON Switches | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Select | | | | | | | | |
| Inhibit | C* | B | A | MC14051B | MC14052B | | MC14053B | | |
| 0 | 0 | 0 | 0 | X0 | Y0 | X0 | Z0 | Y0 | X0 |
| 0 | 0 | 0 | 1 | X1 | Y1 | X1 | Z0 | Y0 | X1 |
| 0 | 0 | 1 | 0 | X2 | Y2 | X2 | Z0 | Y1 | X0 |
| 0 | 0 | 1 | 1 | X3 | Y3 | X3 | Z0 | Y1 | X1 |
| 0 | 1 | 0 | 0 | X4 | | | Z1 | Y0 | X0 |
| 0 | 1 | 0 | 1 | X5 | | | Z1 | Y0 | X1 |
| 0 | 1 | 1 | 0 | X6 | | | Z1 | Y1 | X0 |
| 0 | 1 | 1 | 1 | X7 | | | Z1 | Y1 | X1 |
| 1 | x | x | x | None | None | | None | | |

*Not applicable for MC14052
x = Don't Care

Figura 22. (Izda)Descripción de pines del MC14051BCP. (Dcha)Tabla de verdad del MC14051BCP

Para realizar el control de las medidas a través del multiplexor, debemos conectar las 3 entradas de selección (A, B, C) a 3 pines digitales de la tarjeta Arduino. También conectaremos los sensores que necesitemos a las entradas X[0-7] del multiplexor y por último la salida X a una entrada analógica de la tarjeta. Mediante la programación de la tarjeta, que se explicara en el próximo capítulo, vamos a poder seleccionar de forma sucesiva todas las entradas del multiplexor, recogiendo así las medidas en instantes prácticamente idénticos.

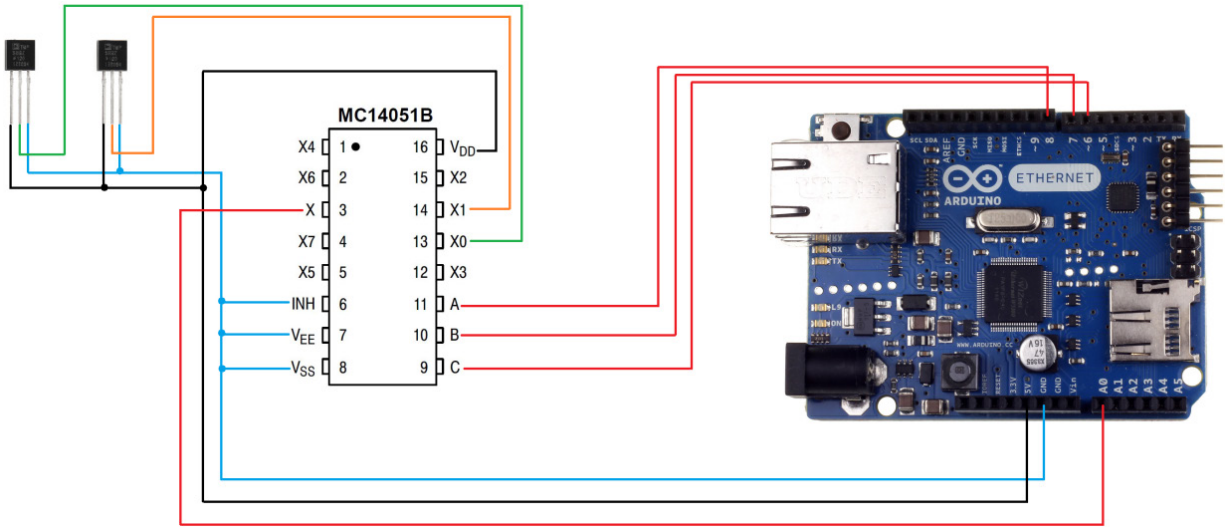



Figura 23. Conexión de 2 sensores de temperatura con la Arduino Ethernet Board mediante multiplexor

### 3.4.1. Ejemplo básico de programación

Para utilizar los multiplexores en este proyecto, debemos controlar las entradas de selección de estos. De esta forma podremos acceder cuando lo necesitemos a la entrada con el sensor deseado. La forma de establecer las entradas de selección es el mayor reto que supone el uso de los multiplexores.

Como primera toma de contacto con los multiplexores, se propone realizar un código muy básico para comprobar el correcto funcionamiento de estos. Más adelante se explicara un método más sofisticado para acceder a cada entrada.

En la figura 23, hemos conectado los pines digitales 6, 7 y 8 a las entradas de selección del multiplexor y las hemos declarado como salidas. Para recoger las medidas, hemos conectado el pin analógico A0 a la salida del multiplexor.

Estableciendo manualmente los bits de selección de forma secuencial, podemos comprobar fácilmente si se muestran las medidas de nuestros sensores, recordando que cada uno está conectado a una entrada distinta de nuestro multiplexor.

Para comenzar a modular el código, podemos realizar una función idéntica a la explicada en el apartado de sensores, que se encargue de leer y convertir la medida analógica recogida en el pin A0.

Como se ha comentado arriba, este código es puramente de depuración, así que más adelante se propondrá un código mucho más compacto que determine la entrada a leer y la convierta a 3 bits de selección automáticamente. [22]

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

  // entradas selección mux
 pinMode(6,OUTPUT);  // C MSB
 pinMode(7,OUTPUT);  // B
 pinMode(8,OUTPUT);  // A LSB

 // salida mux
 pinMode(A0,INPUT);
}

void loop()
{
 // selección 000 - canal x0
 digitalWrite(8,LOW);
 digitalWrite(7,LOW);
 digitalWrite(6,LOW);

 //llamar a función medida  Tmp36

 // selección 001 - canal x1
 digitalWrite(8,HIGH);
 digitalWrite(7,LOW);
 digitalWrite(6,LOW);

 //llamar a función medida  Tmp36

 // selección 010 - canal x2
 digitalWrite(8,LOW);
 digitalWrite(7,HIGH);
 digitalWrite(6,LOW);

 //continuar hasta selección 111
}
```

Figura 24. Ejemplo de programación de sensores multiplexados

## 3.5. Reloj en tiempo real (RTC)

Con el objetivo de documentar perfectamente las medidas tomadas por nuestro registrador de datos, se hace imprescindible el uso de un reloj en tiempo real. Este nos permite registrar la fecha y la hora durante años de forma independiente al funcionamiento de Arduino, es decir, que después de ser configurado por primera vez, no será necesario volver a hacerlo.

Además, el reloj debe respaldarse con una batería de botón de 3.3V, la cual permite que cuando la tarjeta Arduino pierda la alimentación, el reloj pueda seguir de forma independiente su cuenta. De este modo cuando se vuelva a requerir la fecha y hora, el reloj estará completamente actualizado.

El reloj en tiempo real (RTC) utilizado, está basado en el ***chip DS1307*** *(figura 25.Dcha)*, presentado en encapsulado PDIP *(Plastic Dual in line Package)* con 8 pines de conexión. Es un chip de bajo consumo, ofrece reloj y calendario codificado en BCD *(binary coded decimal)* y además posee un pequeño espacio de memoria Sram de 56 bytes. [23]

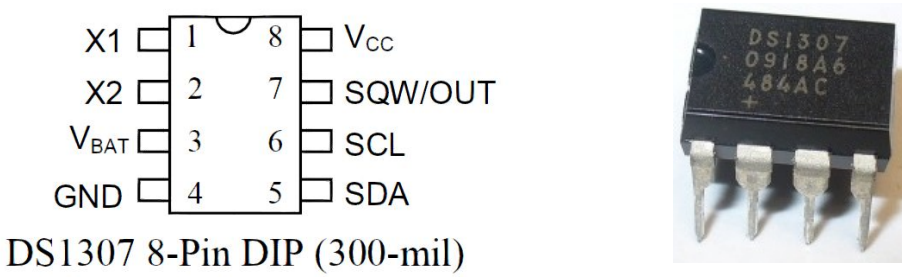



Figura 25. (Izda)Descripción de pines del chip DS1307. (Dcha)Chip DS1307

Las características técnicas más importantes son:

- Cuenta de segundos, minutos, horas, día, mes y año (también bisiesto)
- 56 bytes de memoria Sram
- Interfaz de comunicación I2C (SDA y SDL)
- Señal cuadrada de salida opcional (SQW)
- Bajo consumo en modo respaldo con batería
- Rango temperaturas de operación -40ºC a +85ºC

Para el correcto funcionamiento del RTC, se debe realizar un pequeño montaje con varios componentes discretos que se citan a continuación: [8]

- Chip DS1307
- Cristal oscilador de cuarzo de 32MHz
- 2 resistencias de 2200 ohm 1/4W
- Condensador cerámico de 0.1uF
- Conector macho simple con paso de 2.54mm
- Batería de litio de 12mm y 3.3V
- Zócalo para batería de 12mm
- PCB (placa soldadura) o placa de prototipado para el montaje

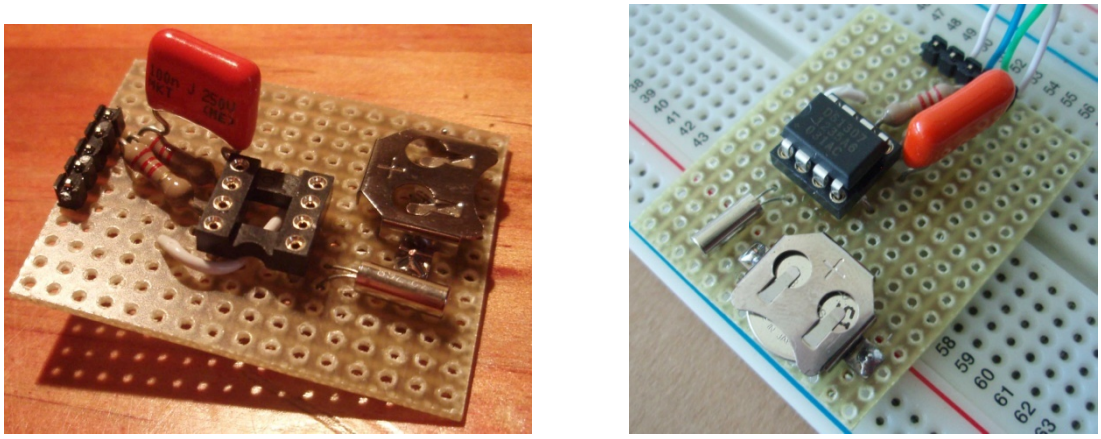


Figura 26. Montaje casero del reloj en tiempo real

A continuación se muestra un esquema de las conexiones necesarias para conectar nuestro RTC con la tarjeta Arduino: [8]

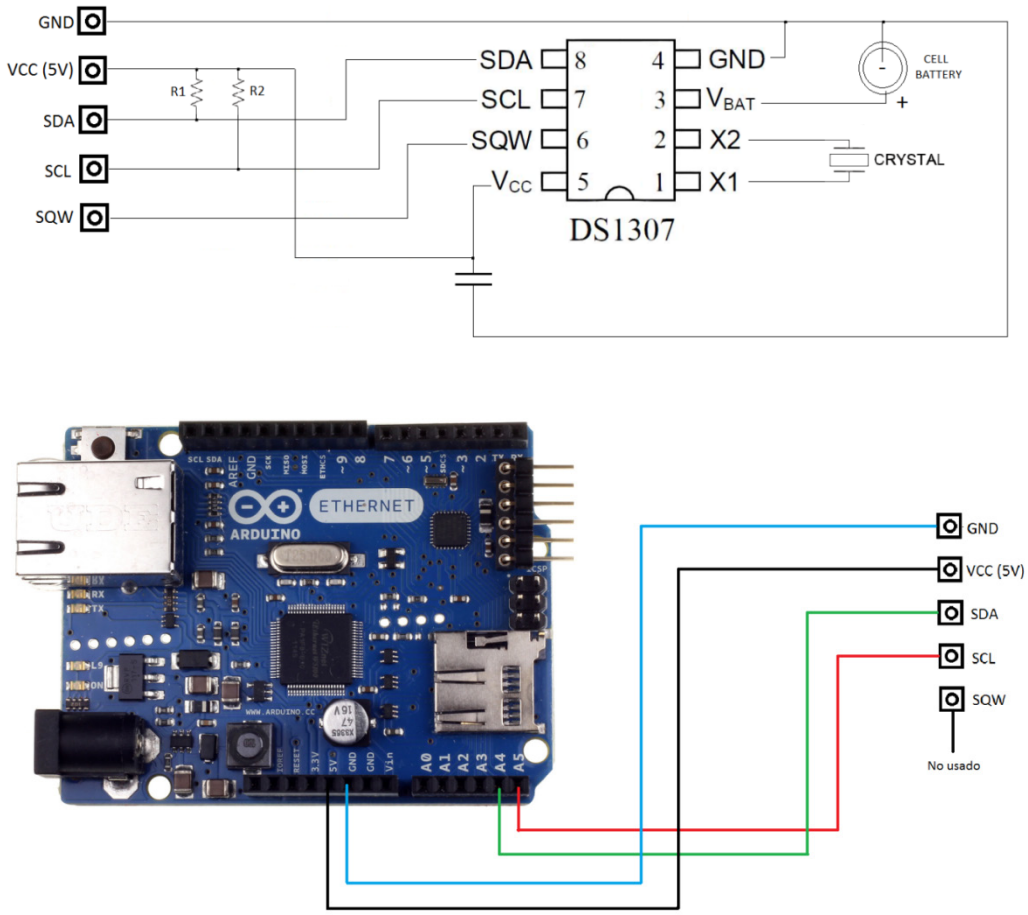



Figura 27. (Arriba)Componentes para la configuración del DS1307. (Abajo)Conexión del DS1307 con Arduino Ethernet Board

### 3.5.1. Ejemplo básico de programación

Para la utilización del reloj en tiempo real basado en el chip DS1307, debemos descargar e incluir una librería nueva que no se encuentra por defecto en el software de Arduino. La inclusión de librerías en el entorno Arduino se abordara en el capitulo siguiente, de modo que aquí solo se va a mostrar un código básico del uso del reloj.

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

RTC_DS1307 RTC;

void setup ()
{
 Serial.begin(9600);
 Wire.begin();
 RTC.begin();
 if (! RTC.isrunning()){
  Serial.println("RTC is NOT running!");
 }
 else{
  RTC.adjust(DateTime(__DATE__, __TIME__));
 }
}

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

 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();

 delay(1000);
}
```

En primer lugar debemos incluir la librería estándar *<Wire.h>* que es la que controla la comunicación I2C entre el reloj y la tarjeta Arduino. La librería *"RTClib.h"* contiene todas las instrucciones para controlar el reloj y poder acceder a la fecha y hora actuales.

La instrucción *RTC.adjust* se utiliza para establecer la hora del reloj igual que la hora del sistema operativo en el momento de la compilación del código. De esta forma solo es necesario que la ejecutemos una vez, luego podremos dejarla comentada.

Declaramos la variable *now* a través de la cual podemos acceder al año, mes, día, hora, etc. La librería, también nos permite utilizar el elemento temporal *unixtime()* que más adelante se explicara con detalle.

Con este programa y el reloj conectado a la tarjeta como se explica en la figura 27, observaremos la fecha y hora por el puerto serie cada segundo. [8]

Figura 28. Ejemplo de programación del reloj en tiempo real para que devuelva la hora actual

## 3.6. Tarjeta micro SD

En un sistema registrador de datos, se hace casi imprescindible disponer de algún dispositivo de almacenamiento masivo que permita recoger y guardar todos los datos de forma segura y estable. Por suerte, la tarjeta Arduino que vamos a utilizar dispone de una ranura para tarjetas de memoria micro SD.

Este hecho y todos los descritos anteriormente, convierten a la tarjeta Arduino en un sistema totalmente adecuado para realizar el proyecto actual. Por todo ello, se va a utilizar una tarjeta ***Kingston Micro SD HC de 8GB*** que nos permitirá guardar y leer gran cantidad de datos cuando sea necesario. [24]

Las características técnicas principales son: [24]

- Clase 4 (HC) con velocidad de transferencia mínima de 4MB/s
- Capacidad de 8GB
- Temperatura de funcionamiento -25ºC a +85ºC
- Dimensiones 11x15x1mm
- Adaptador a tamaño SD
- Dimensiones del adaptador 24x32x2.1mm

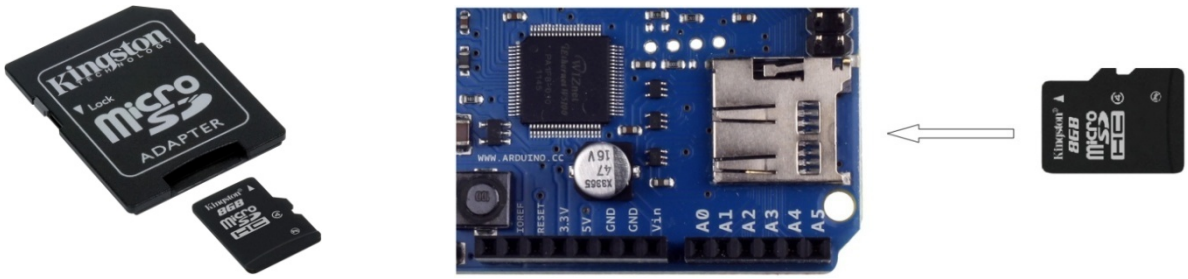



Figura 29. Utilización de la tarjeta micro SD mediante la ranura disponible en la Arduino Ethernet Board

### 3.6.1. Ejemplo básico de programación

Para poder utilizar el sistema de almacenamiento mediante la tarjeta micro SD, debemos utilizar la librería estándar *<SD.h>* la cual nos permite crear ficheros, editarlos y guardarlos durante la ejecución del programa. Aquí se presenta un código sencillo para realizar la escritura y lectura de un archivo.

```
#include <SD.h>

void setup()
{
 Serial.begin(9600);
 Serial.println("Inicializando tarjeta SD...");
 if(!SD.begin(4))
 {
  Serial.println("Error inicializando SD");
  return;
 }

 File fp;

 fp = SD.open("file.txt",FILE_WRITE);
 if(fp)
 {
  fp.println("Escritura en SD de prueba");
  fp.println("Escribiendo 1,2,3");
 fp.close();
 }
 else
 {
  Serial.print("Error creando archivo");
 }

 fp = SD.open("file.txt",FILE_READ);
 if(fp)
 {
  while(fp.available())
  {
   Serial.write(fp.read());
  }
 fp.close();
 }
 else
 {
 Serial.print("Error leyendo archivo");
 }
}

void loop()
{}
```

En este ejemplo se ha escrito todo el código en la función *setup(),* ya que solo queremos que se haga la lectura y escritura en la tarjeta una vez.

Primero necesitamos activar la comunicación con la tarjeta SD mediante *SD.begin().* En caso de errores, los mensajes de alerta se mandan por el terminal serie.

En primer lugar se crea un archivo para escribir y después se abre el mismo para leer. Para comprobar la lectura, se enviaran los datos por el terminal serie de forma que podamos comprobar que la escritura ha sido correcta.

Hay que destacar que para que la escritura en la tarjeta se haga correctamente, el manejador del archivo debe ser cerrado, sino no se garantiza que los datos estén guardados de forma correcta.

La librería *<SD.h>* posee gran variedad de funciones que nos permiten realizar tareas muy útiles sobre este tipo de tarjetas. Debido a esto, la utilización de estas funciones requiere de una gran cantidad de memoria de programa *(Flash),* así que debemos ser cuidadosos cuando las utilicemos para no llenar nuestro microcontrolador Avr. [25]

Figura 30. Ejemplo de programación de la tarjeta micro SD para realizar lecturas y escrituras

# Capítulo 4

# Desarrollo del Software

En este capítulo se va a explicar en profundidad todo lo relacionado con la programación del proyecto. Se puede decir que el código es la parte fundamental, ya que es aquí donde establecemos por completo la funcionalidad y las características de nuestro registrador de datos.

Para realizar el desarrollo del software, se ha utilizado el entorno de programación Arduino IDE junto con la Arduino Ethernet Board para así poder compilar, volcar y ejecutar las partes del código deseadas. El Arduino IDE no dispone de simulador ni de depurador, por lo que debemos tener físicamente el hardware para ser cargado con el código y ejecutado.

En el capitulo anterior, se han comenzado a introducir unas pequeñas referencias a la codificación del hardware utilizado. Estos códigos sencillos, tienen la finalidad de mostrar al programador una forma sencilla de comprobar la funcionalidad del hardware adquirido.

Cabe destacar que en la realización de un proyecto extenso como el actual, conviene conocer todos los pequeños detalles de programación del hardware que va a ser usado. Teniendo controlada la programación básica del hardware, tendremos todas las piezas listas para comenzar a unir el puzle y realizar el proyecto global con garantías.

## 4.1. Instalación del software Arduino IDE

La instalación del entorno de programación de Arduino se ha realizado sobre una maquina con Ubuntu Linux 11.04. Para ello se deben tener en cuenta una serie de consideraciones, como la instalación previa de Java en nuestra maquina debido a que el IDE de Arduino lo necesita para ejecutarse correctamente.

Para comenzar, debemos visitar la página oficial de soporte de instalación de Arduino [26]. Aquí encontraremos las instrucciones generales de instalación sobre cualquier distribución Linux. También podemos encontrar las instrucciones especificas para cada de ellas.

Primero debemos instalar uno de los siguientes paquetes de Java:

- OpenJDK-7-jre o la versión 6
- Sun-java6-jre
- Oracle JRE-7

También es muy recomendado desde Arduino instalar la versión del entorno IDE 1.0.1 sobre Linux debido ya que viene con el compilador gcc integrado y nos ahorra tener que hacer mas configuraciones en nuestro sistema. Podemos descargar cualquier versión de Arduino desde la pagina [27].

En nuestro sistema se ha decidido instalar la versión de Oracle Java 7. Para ello hay que tener en cuenta las siglas JDK y JRE puesto que son paquetes distintos.

- JDK (Java Development Kit), es el kit completo de Java para desarrolladores. Incluye todo el entorno java, compilador, JRE y JVM
- JRE (Java Runtime Environment), es el entorno de ejecución de Java y está destinado a usuarios. Incluye JVM.
- JVM (Java Virtual Machine), es un programa que ejecuta código Java previamente compilado.

Para saber que versión JRE de Java tenemos, si es que ya teníamos alguna, podemos comprobarlo desde el terminal:

- $java -version
- $javac -version (para comprobar la versión del compilador completo JDK)

Para instalar Oracle Java JRE de forma manual, vamos a la página de Oracle [28] y entramos a descargar la última versión en este caso la SE 7u17. Entramos en el apartado JRE (download), aceptamos la licencia de uso y buscamos la versión que en nuestro caso es Linux x86 (32 bits). Podemos comprobar el tipo de sistema operativo que tenemos desde el terminal con $lscpu.

El paquete tiene el nombre "j*re-7u17-linux-i586.tar.gz*" y debemos guardarlo en nuestra carpeta personal. Después realizamos las siguientes acciones: [29]

- Descomprimir el archivo: $tar -xvf jre-7u17-linux-i586.tar.gz
- Creamos un directorio en la ruta: $sudo mkdir -p /usr/lib/jvm/jre1.7.0 *(con -p sobrescribe si ya hay una carpeta con ese nombre).*
- Movemos la carpeta descomprimida dentro del directorio creado: $sudo mv jre1.7.0_17/* /usr/lib/jvm/jre1.7.0 *(con el * muevo el contenido de la carpeta jre1.7.0_17 dentro de la carpeta destino, sino movería la carpeta).*
- Instalamos el paquete: $sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jre1.7.0/bin/java 0
- Escogemos la versión del JRE si tenemos varios instalados: sudo update-alternatives --config java *(pulsar Intro para escoger la opción *).*

Después de instalar Java y una vez descargado el paquete de Arduino IDE 1.0.1 desde la web, solo debemos descomprimirlo en el directorio deseado, por ejemplo en nuestro Home. El paquete de Arduino tiene el nombre *"arduino-1.0.1-linux.tgz".*

- Descomprimir: *$tar -xvf arduino-1.0.1-linux.tgz*
- Crear una carpeta */bin* en nuestro */Home*: *$mkdir bin* para albergar los enlaces a los ejecutables de nuestros programas.
- Crear un enlace simbólico al ejecutable de Arduino y guardarlo en la carpeta */bin*: *$ln -s /home/javier/arduino-1.0.1/arduino .*
- Incluir el enlace en el path del sistema: *$echo $PATH*, editar con *$vi .bashrc* y actualizar con *$. .bashrc*
- En el path introducir la carpeta */bin* que hemos creado en nuestro */home*: *export PATH=$PATH:$HOME/bin*

## 4.2. Inclusión de librerías externas

Por defecto, el entorno de desarrollo de Arduino nos permite acceder a una serie de librerías para el control de los recursos del hardware de nuestra tarjeta. Librerías como *<SD.h>, <Wire.h>, <SPI.h> o <Ethernet.h>* ya se encuentran disponibles en el IDE Arduino, de modo que podemos hacer uso de todas las funciones de control de protocolos, almacenamiento, ethernet, etc.

Para utilizar determinadas expansiones junto con las tarjetas Arduino, primero debemos adquirir el hardware o construirlo nosotros mismos y después debemos descargar las librerías de funciones para programar correctamente. Las instrucciones siguientes son validas para incluir cualquier librería en nuestro proyecto.

En nuestro caso debemos descargar e incluir en el entorno Arduino la librería de control del reloj en tiempo real DS1307. Accediendo a la página de descarga [30], encontramos los archivos necesarios como el fichero fuente **.cpp* y el header **.h* que deben ser descargados como un archivo Zip.

Una vez descargado el archivo Zip, lo descomprimimos y obtendremos una carpeta llamada *RTClib-master* con los archivos fuente en su interior. Esta carpeta la debemos copiar dentro de directorio *Arduino/Libraries*. Para que el entorno de Arduino la reconozca solo tendremos que reiniciarlo.

Una vez reiniciado el entorno, podremos incluir la librería en nuestro código con la directiva *#include "RTClib-master.h".* Ahora ya tenemos acceso a las funciones de control del reloj en tiempo real. [31]

## 4.3. Diagrama general del registrador de datos [32]

En este apartado se comienza a explicar de forma general la estructura, vista de forma general, del código para nuestro registrador de datos. Debido a la amplitud del código, aquí se presentara un diagrama con las fases del procesado de los datos recibidos.

En nuestro registrador de datos, vamos a disponer de una serie de bloques de información referente a cada puerto de medida. Estos bloques son leídos y procesados con el fin de conocer cuáles son los requerimientos del sistema y así poder recoger medidas, procesarlas y guardarlas de forma conveniente en nuestra tarjeta de almacenamiento.

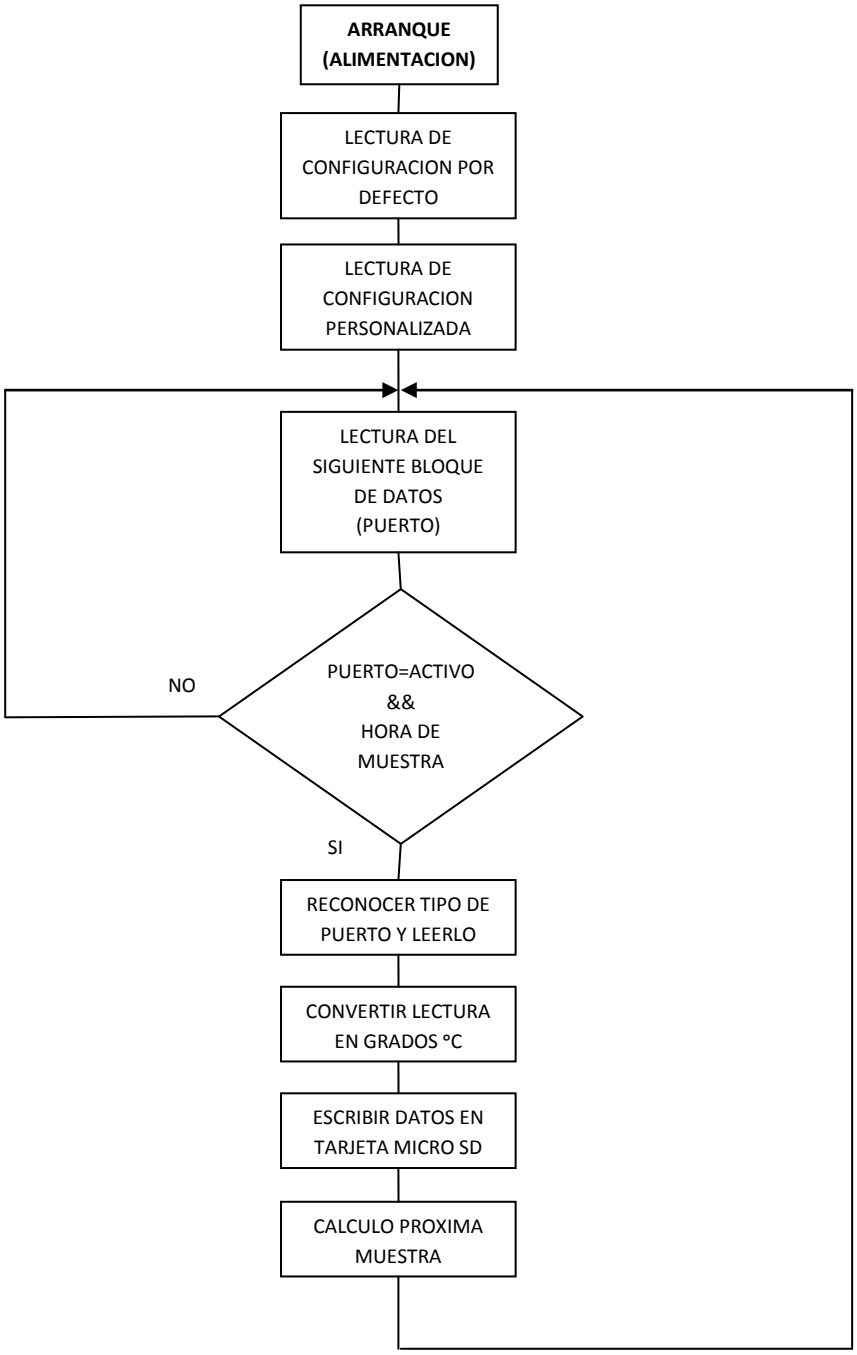



Figura 31. Diagrama de bloques que representa el funcionamiento del sistema

### 4.3.1. Estructura de datos principal

Los bloques de datos mencionados anteriormente, contienen toda la información necesaria para describir la funcionalidad de un puerto en concreto. Cada puerto contendrá una serie de campos que lo identifican y lo habilitan para la medida, como por ejemplo, el tipo de sensor, la hora inicial de medida, el intervalo de medida, etc.

Debido a que a codificación se realiza en lenguaje C/C++, cada bloque de datos será representado mediante una estructura de datos. Para trabajar con estos datos de forma más cómoda, se ha definido un nuevo tipo de dato que representa a la estructura general *(figura 32)*.

La definición se hace mediante la instrucción *typedef* y se le ha dado un nombre característico *(datatemp),* para ser recordado fácilmente durante la escritura del código. Teniendo en cuenta que los recursos de memoria de nuestro microcontrolador están muy limitados, la definición de los campos se ha hecho estableciendo los tipos de datos más pequeños que nos permitan ofrecer completa funcionalidad.

La definición de la estructura de datos de cada puerto es la siguiente:

```
//definición de los tipos básicos de Arduino
#include "Arduino.h"

//estructura de datos de cada puerto
typedef struct{
  char sensor[6];          //nombre sensor
  byte mux;                //tipo de entrada
  byte puerto;             //puerto de lectura
  byte sample_hour;   //hora de lectura
  byte sample_min;    //minuto de lectura
  byte sample_sec;    //segundo de lectura
  long interval;         //siguiente medida (4 bytes)
  byte enable;           //activar/desactivar puerto
}datatemp;
```

Figura 32. Estructura de datos para cada uno de los sensores activos

A continuación se hace una descripción de cada campo de la estructura, especificando los posibles valores que pueden tomar para que el sistema los acepte correctamente.

- *Variable sensor:* debe especificar uno de los dos tipos de sensores de los que disponemos en este proyecto, TMP36 o LM35. Si en el futuro se incluye un nuevo tipo de sensor en el sistema, el nombre de este debería tener una longitud de 5 caracteres como máximo.

- *Variable mux:* es utilizada para especificar si un sensor está conectado directamente a un puerto analógico o a través de un multiplexor. Si *mux* toma el valor 0 entonces el puerto es directamente analógico, si *mux* toma el valor 1 entonces está conectado a través del multiplexor 1 y si toma el valor 2 correspondería al multiplexor 2.

- *Variable puerto:* sirve para especificar el número de puerto a leer y está íntimamente relacionado con la variable *mux*. Si *mux* toma el valor 0, entonces *puerto* solo podrá tomar los valores 0 y 1 ya que son las únicas dos entradas directamente analógicas que están disponibles. Si *mux* toma el valor 1 o 2, *puerto* podrá tomar valores entre 0 y 7 ya que son el numero de puertos del que dispone cada multiplexor.

- *Variable sample_hour:* esta variable especifica la hora en que debe producirse la primera medida.

- *Variable sample_min:* esta variable especifica el minuto en que debe producirse la primera medida.

- *Variable sample_sec:* esta variable especifica el segundo en que debe producirse la primera medida.

- *Variable interval:* especifica el intervalo entre la primera medida y la siguiente medida. Debe ser introducida en segundos.

- *Variable enable:* esta es una de las variables más importantes ya que permite activar o desactivar el puerto en cuestión. Si toma el valor 0, el puerto no será tenido en cuenta y no tomara tiempo de ejecución de código, pero si toma el valor 1 el puerto se toma como activo.

En caso de que alguna de las variables de la figura 32 tome algún valor no especificado arriba, el control del programa lanzara un código de error único para cada suceso, así se podrá saber que parte de la configuración es errónea.

Haciendo uso del nuevo tipo de dato *datatemp* y teniendo en cuenta que hay disponibles 18 puertos de lectura, definiremos fácilmente en el código principal un array de 18 elementos del tipo *datatemp* para configurar los puertos disponibles.

Debido a un error en el preprocesador del Arduino IDE, siempre deberán declararse las estructura de datos en un archivo **.h* adjunto a nuestro código. Para ello, solo tendremos que crear nuestro archivo *struct_data.h* en la misma carpeta del código principal e incluirlo en este mediante la directiva *#include "struct_data.h".* [33]

Si no realizamos este paso, el compilador de Arduino nos informara de que las variables no están declaradas en el ámbito del código principal. Debido a este hecho, podemos observar que también se hace imprescindible la inclusión de la librería de tipos *#include "Arduino .h"* en nuestro archivo header, ya que si no, no podríamos declarar los tipos de datos utilizados como *byte, char* o *long*.

### 4.3.2. Control del flujo de programa

Para comenzar a introducirnos en la estructura del código fuente, es necesario en primer lugar mostrar un pequeño seudocódigo donde se observen de manera rápida las partes más importantes del programa.

El núcleo central del código se ha modulado en funciones separadas para dotarlo de un lectura más fácil. En él, se implementa un bucle para leer e identificar los puertos que están activos. En cada vuelta, cada uno de los puertos son comprobados mediante dos elementos de decisión, el campo de habilitación *"enable"* y la hora inicial de medida.

```
 //Declarar e iniciar el array de 18 elementos datatemp
datatemp port[18];

void setup()
{
 //inicialización de protocolos de comunicación
 //iniciar tarjeta SD
 //iniciar reloj en tiempo real
 //iniciar comunicación serie
}

void loop()
{
 //definir puntero al array de puertos
datatemp *pt;

//recorrer el array de puertos
for(i=0;i<18;i++)
{
 //apuntar a las estructura por orden
 pt=&port[i];
 //comprobar la habilitación de cada puerto
 if(pt->enable == 1){
   //comprobar la hora de medida
  if(pt->hora de medida == ahora){
    leer_puerto();
    convertir_lectura();
    escribir_tarjeta_sd();
    calcular_proxima_muestra();
    }
   }
 }
}
```

Figura 33. seudocódigo que representa el bucle principal del registrador

### 4.3.3. Librerías utilizadas y definición de pines

Antes de comenzar a detallar todas las funciones que componen nuestro código fuente, es necesario realizar la inclusión de las librerías que contienen las funciones que necesitamos utilizar. Esto se realiza como en cualquier programa C mediante la directiva de preprocesador *#include,* al principio del código.

Disponemos de todas las librerías estándar del lenguaje C, como *<stdio.h>* o *<stdlib.h>.* Aparte de estas, Arduino ofrece sus propias librerías con las que se nos permite programar todos los recursos del hardware.

```
//declaración de librerías
#include <stdio.h>          //funciones i/o estándar
#include <string.h>         //funciones para cadenas de caracteres
#include <SD.h>             //funciones para controlar tarjeta micro SD
#include <Wire.h>           //funciones para control del bus I2C
#include <RTClib.h>         //funciones para control del reloj en tiempo real
#include "struct_data.h"    //definición de estructuras de datos
```

Figura 34. Librerías utilizadas en el código

Para realizar un código robusto y fácil de modificar, se han definido al comienzo del código una serie de constantes mediante la directiva *#define*, que permiten establecer fácilmente los pines utilizados en nuestra tarjeta para conectar el hardware externo.

Una de las acciones más comunes, consiste en declarar la dirección de los pines utilizados como entrada o salida. Esto se realiza dentro de la función *setup()* y es imprescindible no olvidarse de ella ya que sino el funcionamiento será indeterminado.

```
//definición de entradas y salidas


//MUX1
#define mux1_sel0 2  //LSB
#define mux1_sel1 3
#define mux1_sel2 5  //MSB
#define mux1_out 2
//MUX2
#define mux2_sel0 6  //LSB
#define mux2_sel1 7
#define mux2_sel2 8  //MSB
#define mux2_out 3
//PIN_LED
#define pin_led 9
```

```
//definición de los pines selmux1
  pinMode(mux1_sel0,OUTPUT);
  pinMode(mux1_sel1,OUTPUT);
  pinMode(mux1_sel2,OUTPUT);

  //definición de los pines selmux2
  pinMode(mux2_sel0,OUTPUT);
  pinMode(mux2_sel1,OUTPUT);
  pinMode(mux2_sel2,OUTPUT);

  //definition del pin led
  pinMode(pin_led,OUTPUT);

   //definición de los pines analógicos
  pinMode(A0,INPUT);
  pinMode(A1,INPUT);
  pinMode(mux1_out,INPUT);
  pinMode(mux2_out,INPUT);
```

Figura 35. (Izda)Definición de variables simbólicas. (Dcha)Sentido de los pines utilizados

### 4.3.4. Identificación y lectura de puertos

El reconocimiento inicial de los puertos de nuestro registrador de datos es la primera de las cuatro grandes funciones del código fuente. En esta función se leen las variables *mux* y *puerto* de las estructuras de datos para poder identificar el sensor que debe ser leído.

| *Variable Mux* | *Tipo de puerto* | *Puertos disponibles* |
|---|---|---|
| 0 | Analógico directamente | A0 y A1 |
| 1 | A través de MUX1 | 0 a 7 (MUX1) |
| 2 | A través de MUX2 | 0 a 7 (MUX2) |

Tabla 2. Relación entre la variable Mux y los puertos a los que hace referencia esta

Antes de introducirnos en la codificación de la función, es imprescindible mostrar cual es la inicialización de nuestras estructuras de datos. La inicialización por defecto que define el comportamiento del sistema, se realiza en el momento de la declaración de nuestro array de estructuras.

Para ser accesible desde las funciones *setup()* y *loop()* debemos declararlo de forma global al principio del código. Debido al orden establecido de las variables agrupadas en el tipo *datatemp*, la inicialización se tiene que hacer de modo similar.

A continuación se muestra un ejemplo de inicialización de los 18 puertos disponibles, donde el orden de los campos es el siguiente:

***Configuración*** *= {nombre sensor , mux , puerto , hora , minuto , segundo , intervalo , habilitación}*

```
datatemp ports[18] = {
                {"TMP36",0,0,12,30,0,20,1},
                {"TMP36",0,1,12,30,0,20,1},
                {"TMP36",1,0,12,30,0,20,1},
                {"TMP36",1,1,12,30,0,20,0},
                {"TMP36",1,2,12,30,0,20,0},
                {"TMP36",1,3,12,30,0,20,0},
                {"TMP36",1,4,12,30,0,20,0},
                .....
                .....
};
```

Figura 36. (Arriba)Orden de las variables agrupadas en la estructura principal. (Abajo)Ejemplo de inicialización de algunos de los puertos

Una vez inicializado el comportamiento de los sensores de nuestro registrador de datos, podemos pasar a explicar cómo se identifica y se lee cada uno de los sensores activos. *El prototipo de la función utilizada para esta tarea, devolverá a la función principal el valor analógico leído y digitalizado a través de un conversor ADC de 10 bits de resolución. La lectura es un numero entero.*

```
//prototipo de la función
int leer_puerto (datatemp *puntero)
{
 //entrada directamente analógica
 if(puntero->mux == 0)
 {
  //solo se disponen de los puertos 0 y 1
  if(puntero->puerto == 0 || puntero->puerto == 1)
  {
   return(lectura_precisa(puntero->mux,puntero->puerto));
  }
 }
 //entrada mediante mux1 o mux2
 else if(puntero->mux == 1 || puntero->mux == 2)
 {
  //solo hay puertos entre 0 y 7
  if(puntero->puerto <= 7 && puntero->puerto >= 0)
  {
   //se convierte el numero de puerto en binario de 3 bits
   //para mandar las líneas de selección del multiplexor

   if(puntero->mux == 1)
   {
    //se establecen los bits de selección mux1
    digitalWrite(mux1_sel0, selmuxbit[0]);          //LSB
    digitalWrite(mux1_sel1, selmuxbit[1]);
    digitalWrite(mux1_sel2, selmuxbit[2]);          //MSB
    return(lectura_precisa(puntero->mux,puntero->puerto));
   }
   if(puntero->mux == 2)
   {
    //se establecen los bits de selección mux2
    digitalWrite(mux2_sel0, selmuxbit[0]);          //LSB
    digitalWrite(mux2_sel1, selmuxbit[1]);
    digitalWrite(mux2_sel2, selmuxbit[2]) ;         //MSB
    return(lectura_precisa(puntero->mux,puntero->puerto));
   }
  }
 }
```

Figura 37. Programación para la identificación de puertos activos

En el capitulo anterior, se mostro un pequeño ejemplo de la utilización de los multiplexores en el proyecto. En esta función del código, se hace imprescindible una estrategia más elaborada para acceder a los puertos de nuestros multiplexores.

Como se observa en la figura 38, el numero de puerto a leer se representa mediante un numero entero, pero para establecer los bits de selección de nuestro multiplexor debemos convertir este entero en un binario de 3 bits *(selmuxbit[3]),* uno para cada línea.

```
//declaración de variables
byte puerto_mux;
byte selmuxbit[3];

//guardar el numero entero del puerto
puerto_mux = puntero->puerto;

//3 vueltas para 3 bits
for(i=0;i<3;i++)
{
 selmuxbit[i] = puerto_mux % 2;
 puerto_mux = puerto_mux / 2;
}
```

Si por ejemplo necesitamos convertir el numero de puerto 6, el bucle procede del siguiente modo:

selmuxbit[0] = 6 % 2 = 0
puerto_mux = 6 / 2 = 3

selmuxbit[1] = 3 % 2 = 1
puerto_mux = 3 / 2 = 1 (solo parte entera)

selmuxbit[2] = 1 % 2 = 1
puerto_mux = 1 / 2 = 0

*selmuxbit[2,1,0] = [1,1,0]*

Figura 38. Algoritmo para obtener las líneas de selección de los multiplexores

La tarea de leer el puerto se deja en manos de otra función llamada *lectura_precisa(),* la cual hace varias lecturas consecutivas y calcula la media de estas para hacer un promedio más estable.

```
//prototipo de la función
int lectura_precisa(byte mux,byte puerto)
{
 if(mux == 1){
  puerto = mux1_out;}
 else if(mux == 2){
  puerto = mux2_out;}

 for(i=0;i<=3;i++)
 {
  var = analogRead(puerto);
  temp = temp + var;
 delay(40);
 }
 return( temp >> 2);
}
```

En caso de que el sensor a leer se encuentre conectado a través de un multiplexor, se debe seleccionar la entrada analógica correspondiente al inicio.

Si el sensor estuviese directamente conectado a una entrada analógica, la medida sería directa.

Acumulando cuatro medidas analógicas con un retraso de 40 ms cada una, obtenemos una media más estable.

Figura 39. Se realizan múltiples lecturas de un sensor para hacer después una media más exacta

### 4.3.5. Conversión de lecturas

Después de recoger la lectura analógica de nuestro sensor, tenemos que realizar la conversión de este valor a grados centígrados. En el capitulo anterior, se explicaron las ecuaciones de conversión de cada uno de los sensores utilizados aquí. Utilizando estas ecuaciones, podemos definir una función de conversión para cada uno de los sensores y así llamarlas cuando corresponda.

Para aumentar el nivel de precisión de las medidas, la tarjeta Arduino permite variar la tensión de referencia de los puertos analógicos mediante una instrucción de programación *(1.1V).* Por contra, se reducirá el rango de la tensión medida, pero esto no influye en exceso en el proyecto actual. [34]

$$Tension\ referencia = 5V \rightarrow \frac{5000}{1024} = 4.882\frac{mV}{esc}$$

$$Tension\ referencia = 1.1V \rightarrow \frac{1100}{1024} = 1.076\frac{mV}{esc} \rightarrow Mayor\ Precision$$

$$Sensor\ Tmp36 \rightarrow Vout(max) = 2V > 1.1V \rightarrow Menor\ Rango$$
$$Sensor\ Lm35 \rightarrow Vout(max) = 1.5V > 1.1V \rightarrow Menor\ Rango$$

La función *convertir_lectura()* deberá comprobar el nombre del sensor para poder ejecutar una ecuación de conversión o otra, recibirá la medida analógica recogida en la función *leer_puerto()* y devolverá los grados centígrados correspondientes.

```
//establece referencia analógica a 1.1v
analogReference(INTERNAL);

//prototipo de la función
float convert_lectura(datatemp *puntero,int medida)
{
  //comparación de cadenas de caracteres
  if(strcmp(puntero->sensor,"TMP36") == 0)
  {
    //función especifica para tmp36
    return(convert_temp36(medida));
  }
  else if(strcmp(puntero->sensor,"LM35") == 0)
  {
    //función especifica para lm35
    return(convert_lm35(medida));
  }
}
```

Figura 40. Código que identifica el tipo de sensor conectado al puerto actual activo

Las funciones de conversión son invocadas desde *convert_lectura()* y contienen en su interior las ecuaciones de cada tipo de sensor utilizado. Si en un futuro se quisiera añadir otro tipo de sensor en el sistema, bastaría con definir su propia función de conversión y realizar otra comparación en la función *convert_lectura()* con el nombre del nuevo sensor.

Las funciones de conversión definidas para nuestros sensores, calculan los grados centígrados con variables tipo *float*, por lo que se debe hacer *casting* a la medida analógica *(medida)* pasada como argumento de entrada.

```
//prototipo de la función para Tmp36
float convert_tmp36(int medida)
{
 float mvolt,grados;

 mvolt = (float)medida * 1100 / 1024;
 grados = (mvolt  - 500) / 10;
return(grados);
}
```

```
//prototipo de la función para Lm35
float convert_lm35(int medida)
{
 float mvolt,grados;

 mvolt = (float)medida * 1100 / 1024;
 grados = mvolt / 10;
return(grados);
}
```

Figura 41. Funciones de conversión de mili voltios a grados centígrados

### 4.3.6. Escritura de datos en tarjeta SD

La función encargada de la escritura de datos en la tarjeta micro SD es muy importante dentro del código, debido a que se encarga de nombrar adecuadamente los archivos, escribir las temperaturas y registrar la fecha y hora exacta de cada medida.

En este sistema registrador de datos, cada sensor activo tendrá asociado su propio archivo de datos, en el que se registrara su actividad de forma indefinida. Para poder identificar de forma sencilla cada archivo de datos, se aplicara un formato único a los nombres de archivo.

Según la librería utilizada <*SD.h*>, la longitud máxima del nombre de archivo debe ser de 8 caracteres mas 3 para la extensión (8+3), si se sobrepasan los 8 caracteres el archivo no se podrá crear correctamente. El formato establecido es el siguiente:

*sensor-muxpuerto.extension -> Ejemplo = TMP36-15.txt*

El prototipo de la función *write_SD()* tiene que recibir varios parámetros para desempeñar su función. En primer lugar se le pasa el puntero a la estructura correspondiente, después la medida en grados y por ultimo una variable temporal para el manejo del reloj en tiempo real. En el siguiente apartado se explicara en detalle el uso de las variables temporales.

La función *write_SD()* escribe toda la información de la medida utilizando un formato especifico para todos los registros. Este formato requiere de la hora de medida y de la medida en sí misma. El formato es el siguiente:

| *Año/Mes/Día Hora:Minuto:Segundo Grados C* |
| --- |
| *AAAA/MM/DD HH:MM:SS XX.XX C* |

A continuación se muestra el prototipo de la función de escritura, en la que utilizamos la función *print()* para guardar datos en la tarjeta micro SD:

```
void write_sd(datatemp *pt,DateTime now,float grados)
{
 char ext[5]=".txt";
 char buf[5];

 //construir el nombre de archivo
 sprintf(filename,"%s-%d%d%s",pt->sensor,pt->mux,pt->puerto,ext);

 fp = SD.open(filename,FILE_WRITE);
 if(fp)
 {
  fp.print(now.year());              //escribe la  fecha
  fp.print('/');
  fp.print(now.month());
  fp.print('/');
  fp.print(now.day());
  fp.print(' ');

  fp.print(now.hour());              //escribe la hora
  fp.print(':');
  fp.print(now.minute());
  fp.print(':');
  fp.print(now.second());
  fp.print(' ');

  dtostrf(grados,4,2,buf);
  fp.print(buf);                     //escribe la medida
  fp.print(' ');
  fp.println('C');

  //necesario cerrar fichero para guardar datos correctamente
  fp.close();
 }
}
```

Figura 42. Función que escribe los datos medidos en la tarjeta micro SD

### 4.3.7. Calculo de muestra inicial y próxima muestra

El cálculo de las siguientes medidas de nuestros sensores, se establece haciendo uso del reloj en tiempo real añadido en el sistema. Teniendo en cuenta los datos almacenados en las estructuras de cada sensor, hacemos uso de la hora inicial y del intervalo de medida para establecer la siguiente muestra.

Utilizando la librería *<RTClib.h>,* definimos las variables temporales que nos ayudaran a controlar los tiempos. En el capitulo anterior, se presento un código inicial para acceder al reloj en tiempo real y mostrar la fecha y hora. Aquí ahondaremos mas en el juego de instrucciones disponibles y en las estrategias de cálculo de tiempos.

```
//inicialización del RTC
#include <Wire.h>
#include "RTClib.h"

//tipo de variable RTC
RTC_DS1307 RTC;

void setup()
{
 Wire.begin();
 RTC.begin();

 //comprueba el arranque del RTC
 if(!RTC.isrunning()){
  Serial.print("error en el RTC");}
}
void loop()
{
 //variable temporal
 DateTime now = RTC.now();

  //acceso a los tiempos deseados
 Serial.print(now.hour());
}
```

Aquí se presenta el conjunto de instrucciones básico para poder definir y usar una variable temporal en nuestro código.

Como se observa, la variable *now* es de tipo *DateTime*. Este tipo permite acceder a todos los campos de la fecha y hora actuales e incluso futuros, como se verá más adelante:

- now.year()
- now.month()
- now.day()
- now.hour()
- now.minute()
- now.second()
- now.unixtime()

La referencia temporal *unixtime(),* también es accesible mediante el reloj en tiempo real. Esta consiste en un contador incorporado en sistemas Unix, que permanece activo desde la medianoche del 1 de enero de 1970, contando segundos indefinidamente. Más adelante se hará uso de esta referencia temporal. [8]

Figura 43. Código básico para inicializar el reloj en tiempo real

Para garantizar que la medida inicial se produce en el segundo establecido o en un múltiplo de este, se debe realizar un cálculo previo en la función *setup()* que establezca el segundo concreto para cada uno de los sensores, en el arranque del sistema, cuando llegue la alimentación.

Este cálculo previo, se realiza basándose en que la primera medida debe coincidir en un múltiplo entero del intervalo, para así poder evitar que cualquier retraso en la inicialización del sistema se acumule en los sucesivos intervalos de medida.

Las variables utilizadas en las ecuaciones, tienen el siguiente significado:

- Hora_Actual: hora actual en modo unixtime().
- Tinicial: hora inicial de medida (estructura de datos) en modo unixtime().
- Tmuestra: hora de la siguiente muestra en modo unixtime().
- Δt: intervalo entre medidas (estructura de datos).

En el caso de que el *Tinicial* de medida sea inferior a la hora actual, indica que la medida debe realizarse inmediatamente, por lo que se puede establecer el *Tmuestra*:

$$Si \rightarrow Hora\_Actual > Tinicial$$
$$Entonces \rightarrow Tmuestra = Hora\_Actual + \Delta t$$

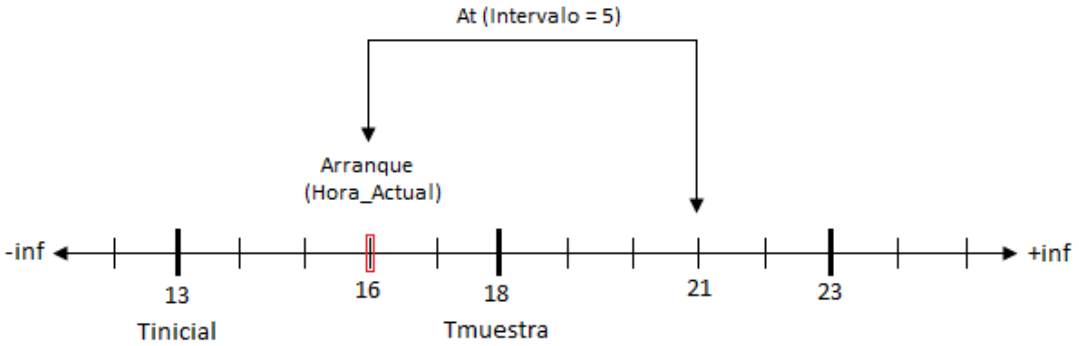
De aquí se desprende la primera de las ecuaciones con *Tmuestra* como incógnita. Si además tenemos en cuenta que se debe cumplir la siguiente ecuación:

$$Tmuestra = Tinicial + n * \Delta t$$

Ya disponemos de dos ecuaciones con dos incógnitas, *Tmuestra* y *n* que debe ser un numero entero (si es necesario se eliminan los decimales), que forzara que la primera medida coincida con un múltiplo del *Tinicial*.

$$Despejando \rightarrow n = \frac{Hora\_Actual + \Delta t - Tinicial}{\Delta t}$$

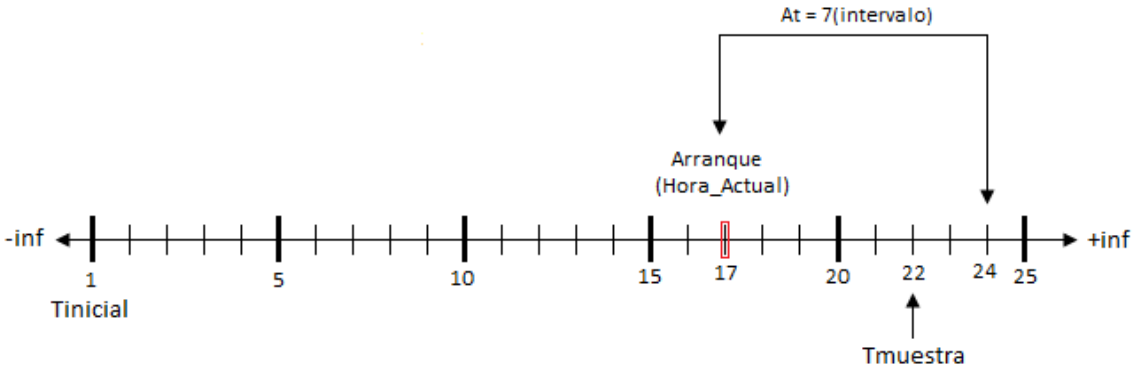
Con el valor de *n* calculado, podemos volver a la segunda ecuación y establecer la hora correcta para la medida. A continuación se presenta unos pequeños ejemplos del cálculo anterior:




$$n = \frac{16 + 5 - 13}{5} = \frac{8}{5} = 1.6 \rightarrow truncar\ decimales = 1$$

$$Tmuestra = 13 + 1 * 5 = 18$$

Figura 44. Ejemplo 1 de cálculo de la muestra inicial




$$n = \frac{17 + 7 - 1}{7} = \frac{23}{7} = 3.28 \rightarrow truncar\ decimales = 3$$

$$Tmuestra = 1 + 3 * 7 = 22$$

Figura 45. Ejemplo 2 de cálculo de la muestra inicial

Una vez analizado el comportamiento del sistema en lo referente al cálculo de las muestras iníciales, se presenta el código equivalente que describe las ecuaciones anteriores. Asimismo, se presenta también la función *prox_muestra()*, que se encarga de ir actualizando las variables temporales referentes a las medidas.

```
//calculo muestra inicial

//array para las muestras iniciales
long init_sample[18];

void setup()
{
 datatemp *pt;
 DateTime now = RTC.now();
 for(i=0;i<18;i++)
 {
  //hora inicial en formato unixtime()
  DateTime tini(now.year(),now.month(),now.day(),pt->sample_hour,pt->.sample_min,pt->sample_sec);

  // n = (Tactual + intervalo - Tinit) / intervalo
  n = (now.unixtime() + pt->interval - tini.unixtime()) / pt->interval;

  // Tsample = Tinit + n * intervalo
  init_sample[i] = tini.unixtime() + n * pt->interval;
 }
}
```

Figura 46. Programación para el cálculo de la muestra inicial

El hecho de utilizar la referencia temporal *unixtime(),* nos facilita la comparación entre variables de tiempo, ya que se convierten en comparaciones de números enteros. Así, las tareas de cálculo de muestras iníciales y muestras sucesivas, no requieren operaciones sexagesimales que son menos amigables.

En el bucle principal de nuestro código, se pone de manifiesto la ventaja de utilizar valores temporales enteros *unixtime(),* en vez de usar la hora, minuto y segundo por separado.

```
//uso de la variable init_sample[18]
void loop()
{
   if(pactual->enable == 1)
   {
    if(init_sample[indice] <= now.unixtime())
    {
        //reconocer el puerto, su tipo y leerlo
        //convertir la lectura a grados
        //escribir lecturas en tarjeta sd

        //calcular la próxima lectura
        prox_muestra(pactual,now,indice);
    }
   }
 }
```

Figura 47. Comparación temporal entre números enteros mediante el contador Unixtime

Un aspecto importante de la función para el cálculo de la próxima muestra, es el hecho de que ignora los retrasos en las medidas. En la función *lectura_precisa(),* se realizan varias medidas seguidas para ganar en estabilidad, lo cual acumula un retraso de 60 milisegundos aproximadamente por cada medida. Debido a que la función calcula la próxima muestra utilizando la hora actual *now.unixtime(),* los posibles retrasos no influyen ni se acumulan.

```
//función de próxima muestra
void prox_muestra(datatemp *pactual,DateTime now,int i)
{
  //actualizar la hora de medida
  init_sample[i] = now.unixtime() + pactual->interval;
}
```

Figura 48. Función que actualiza la próxima hora de medida

### 4.3.8. Lectura de configuración desde archivo

Hasta este momento, hemos trabajado con la inicialización por defecto de las estructuras de datos del sistema. Esto supone que el sistema tenía una configuración dada que no podía ser alterada desde el exterior.

Esto representa una limitación muy grande a la hora de dotar de versatilidad a nuestro sistema registrador de datos, ya que deberíamos adaptarnos a la configuración establecida.

Para solucionar esto, se va a almacenar en nuestra tarjeta micro SD, un archivo de configuración editable por el usuario. Este será leído al inicio de nuestro programa y será almacenado en nuestras estructuras de datos, sobre escribiendo los datos por defecto.

Para el correcto funcionamiento del sistema, el archivo deberá llamarse *"config.txt"* y tendrá que estar situado directamente en la raíz de la micro SD. En la función *setup()* se preguntara si el archivo existe, y siendo así, será leído siguiendo el mismo formato de datos que la inicialización por defecto *(figura 37.Arriba).*

```
//pregunta si hay archivo de configuración
void setup()
{
  if(SD.exists("config.txt"))
  {
    config_sd();
  }
}
```

Figura 49. Si existe el archivo *'config.txt'* en la tarjeta SD, se deben leer sus datos

El archivo de configuración debe tener 18 líneas en orden, una para cada sensor, y el siguiente aspecto para ser procesado correctamente:

```
{TMP36,0,0,13,11,1,21,1}
{TMP36,0,1,15,5,0,32,1}
{TMP36,1,0,13,11,1,5000,1}
{TMP36,1,1,13,35,40,91,0}
{TMP36,1,2,13,11,1,120,0}
{LM35,1,3,13,11,1,120,0}
{LM35,1,4,13,11,1,120,0}
...
...
```

Figura 50. Aspecto correcto del fichero de configuración

El prototipo de función para la lectura del archivo *config_sd(),* primero lee línea a línea el archivo y después "*parsea" o procesa*, las cadenas de caracteres recogidas, almacenándolas en las estructuras de datos correspondientes. *Es importante que el archivo este escrito por orden ascendente de puertos, ya que este será el orden de acceso a cada estructura.*

```
//lectura del archivo de configuración
void config_sd()
{
 File fp;
 datatemp *pt;
 byte i,j;
 char c;
 char buffer[35];
 char delim[4]="{,}";
 char *res;

 //abrir fichero de configuración
 fp = SD.open("config.txt",FILE_READ);
 if(fp)
 {
  for(i=0 ; i<18 ; i++)
  {
   //recorro todas las estructuras
   pt = &ports[i];

   //recojo cada línea del fichero
   for(j=0 ; j<35 ; j++)
   {
    c = fp.read();

    if(c == '\n' || c == EOF)
    {
     buffer[j] = '\0';
     break;
    }
    else
    {
     buffer[j] = c;
    }
   }
```

Figura 51. Lectura archivo configuración. Recogida línea a línea

```
//continuación
    //parsear los datos en buffer y guardarlos en estructura
    res = strtok(buffer,delim);
    if(res == NULL) {strcpy(pt->sensor,res);}
    //mux
    res = strtok(NULL,delim);
    if(res == NULL) {pt->mux = atoi(res);}
    //puerto
    res = strtok(NULL,delim);
    if(res == NULL) {pt->puerto = atoi(res);}
    //sample_hour
    res = strtok(NULL,delim);
    if(res == NULL) {pt->sample_hour = atoi(res);}
    //sample_min
    res = strtok(NULL,delim);
    if(res == NULL) {pt->sample_min = atoi(res);}
    //sample_sec
    res = strtok(NULL,delim);
    if(res == NULL) {pt->sample_sec = atoi(res);}
    //intervalo
    res = strtok(NULL,delim);
    if(res == NULL) {pt->interval = atoi(res);}
    //enable
    res = strtok(NULL,delim);
    if(res == NULL) {pt->enable = atoi(res);}
   }
   fp.close();
  }
 }
```

Figura 52. Lectura archivo configuración. Procesado de valores

El procesado de cada línea del archivo, se ha realizado utilizando la función *strtok(),* la cual busca una serie de delimitadores con los que parte las cadenas de caracteres. Como la mayoría son valores numéricos, necesitamos realizar la conversión de ascii a entero con la función *atoi().*

Si el archivo tuviese algún error de formato, la función no podría realizar bien el procesado de las líneas y el programa nos lanzaría un código de error por el terminal serie.

### 4.3.9. Identificación y señalización de errores

Otro de los apartados más importantes de un proyecto, es el de realizar un control y manejo de los posibles errores encontrados en el sistema. En nuestro registrador de datos, se pueden producir gran variedad de errores relacionados con el hardware utilizado.

Debido a que el sistema no tiene una pantalla de visualización, se va a emplear el terminal serie para comunicar el tipo de error sucedido. Esto requiere de la utilización de un ordenador para conectar nuestra tarjeta Arduino y ver los errores. Para ello debemos tener el Arduino IDE en nuestro ordenador, que nos permitirá abrir el puerto serie.

Para realizar un rápido diagnostico del error ocurrido, se presenta a continuación una tabla con un código numérico de errores. Con él, solo tenemos que abrir el puerto serie, mirar el código de error y visitar esta tabla:

| *Nº Error* | *Evento ocurrido* |
|---|---|
| 1 | El reloj en tiempo real no está funcionando |
| 2 | La tarjeta micro SD no se ha iniciado correctamente |
| 3 | Puerto analógico especificado incorrecto. Solo disponible 0 y 1 |
| 4 | Puerto de multiplexor especificado incorrecto. Solo disponible de 0 a 7 |
| 5 | Numero de multiplexor incorrecto. Solo disponible 0, 1 y 2 |
| 6 | Nombre de sensor incorrecto. Solo disponible TMP36 y LM35 |
| 7 | Error abriendo un archivo de escritura de datos |
| 8 | Error abriendo o leyendo el archivo de configuración<br>Nombre valido de archivo: config.txt |
| 9 | Formato del archivo de configuración incorrecto.<br>Formato valido: {sensor,mux,puerto,hora,min,sec,intervalo,enable} |

Tabla 3. Correspondencia entre los códigos de error

```
void error(byte err)
{
  while(1)
  {
    Serial.print("Codigo_error: ");
    Serial.println(err);

    digitalWrite(pin_led,HIGH);
    delay(100);
    digitalWrite(pin_led,LOW);
    delay(100);
  }
}
```

Se ha definido una función específica que es llamada para realizar el tratamiento de los errores. Esta, muestra el error por el terminal serie y para la ejecución de nuestro registrador de datos de forma indefinida hasta que se repare el suceso.

Cada llamada a esta función, le pasa un numero como argumento que indica el error que se ha producido.

Aquí se puede observar la señalización mediante led de un error, gracias a la intermitencia rápida de este.

Figura 53. Función que muestra el código de error por el puerto serie y lo señala mediante un led parpadeante

El manejo de errores es realmente útil para garantizar el correcto funcionamiento del sistema, sin embargo, si no utilizamos alguna señalización adicional, no seremos capaces de saber si realmente ha ocurrido algún error.

Para solucionar esto, se ha añadido al sistema un diodo led de señalización. Este, tendrá varios comportamientos en función de la operación del registrador. Los tipos de señalización son los siguientes:

- *Intermitencia lenta*: indica el funcionamiento normal del sistema.
- *Intermitencia rápida*: indica que se ha producido un error. *(Ver Tabla 3)*
- *Led apagado o encendido* : indica error en la alimentación del RTC.

La codificación de la operación del led es la siguiente:

```
//operación del led
long led;

void setup()
{
 DateTime now = RTC.now();

 // encendido del led cada 3 segundos
 led = now.unixtime() + 3;
}
void loop()
{
 //recorrer todas las estructuras
 for(indice=0 ; indice<18 ; indice++)
 {
  //leer_puerto();
 //convertir_medida();
 //escribir_sd();
 //prox_muestra();
 }

 //intermitencia normal del led
 if(led <= now.unixtime())
 {
  if(digitalRead(pin_led) == HIGH)
  {
    digitalWrite(pin_led,LOW);
    led = now.unixtime() + 3;
  }
  else
  {
    digitalWrite(pin_led,HIGH);
    led = now.unixtime() + 1;
  }
 }
}
```

Figura 54. Se establece un parpadeo lento en bucle principal para señalizar el correcto funcionamiento

## 4.4. Web server básico

Inicialmente, el proyecto del registrador de datos sobre Arduino, debería haber sido configurable desde una interfaz web. Esto dotaría al sistema de un entorno de configuración y visualización de datos desde red local, muy interesante para el usuario.

Después de realizar un estudio inicial de las características del sistema, se llego a la conclusión de que las necesidades de memoria, tanto Sram como Flash, eran bastante superiores a las ofrecidas por nuestra tarjeta Arduino Ethernet Board.

Sin embargo, aquí se va a mostrar como configurar un web server totalmente accesible desde un ordenador conectado en red con la tarjeta Arduino. La figura 56 solamente es un punto de partida para poder entender cómo se gestionan las conexiones ethernet de la tarjeta. [35]

```
//inicialización del ethernet
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {0x90,0xA2,0xDA,0x0D,0x3A,0x85};
byte ip[] = {192,168,1,103};

EthernetServer server(80);

void setup()
{
  Ethernet.begin(mac,ip);
  server.begin();
}
```

Figura 55. Ordenes básicas para iniciar una conexión Ethernet

Para utilizar la conexión ethernet debemos realizar una serie de pasos iníciales indicados en el código de encima:

- Declarar la dirección MAC de nuestra tarjeta Arduino ethernet. La dirección Mac la podemos encontrar en el reverso de nuestra tarjeta.
- Establecer una dirección Ip para nuestra tarjeta. Esta debe encontrarse dentro del rango de asignación de Ip que tenga nuestro router.
- Declaramos un servidor ethernet que debe escuchar por un puerto. En nuestro caso utilizamos el puerto 80.
- En la función *setup(),* debemos iniciar la conexión ethernet pasándole los parámetros Mac e Ip y también el servidor que comenzara a escuchar por clientes.

Una vez iniciada la conexión, entramos en la función *loop()* en la cual se estará escuchando el puerto 80 hasta que llegue un cliente y haga una petición. Desde un ordenador en red local con la tarjeta Arduino, utilizamos el explorador de internet para acceder a la dirección Ip especificada arriba. Esto enviara una petición al servidor de Arduino que tendrá un aspecto parecido al siguiente: [36]

```
GET / HTTP/1.1\r\n
Host: 10.0.0.20\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:17.0)
Gecko/20100101 Firefox/17.0\r\n
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-ZA,en-GB;q=0.8,en-US;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
\r\n
```

Figura 56. Ejemplo de cabecera de petición web

Cuando el servidor de Arduino lea la petición y encuentre los caracteres *\r\n* seguidos de una línea vacía con otro *\r\n*, entenderá que ha finalizado y comenzara su respuesta. La respuesta comienza con la transmisión de la cabecera http estándar, que se muestra en la figura 57 y continuara con el envió de la pagina web propiamente dicha. [36]

```
void loop()
{
  //espera hasta que haya un cliente
  EthernetClient client = server.available();

  if(client)
  {
   //una solicitud http finaliza con una línea en blanco
   boolean currentLineIsBlank = true;

   while(client.connected())
   {
    if(client.available())
    {
     /*el cliente hace una petición al servidor web.*/
     char c = client.read();

     if(c == '\n' && currentLineIsBlank)
     {
      //encabezado http estándar
      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/html");
      client.println();
      .....
      .....
```

Figura 57. Respuesta inicial del servidor web Arduino

Por ejemplo, el servidor Arduino podría enviar una página web html con la siguiente información:

```
          //envío la web
          client.println("<DOCTYPE html>");
          client.println("<html>");
          client.println("<head><title>Arduino Página Web</title></head>");
          client.println("<body>");
          client.println("<h1>Monitor de temperaturas</h1>");
          client.printl("<p>Lectura sensor1:  ");
          client.print(medida1);
          client.println("</p>");
          client.println("<p>Lectura sensor2: ");
          client.print(medida2);
          client.println("</p>");
          client.println("</body>");
          client.println("</html>");
          break;
        }

        //cada línea recibida del cliente termina con \r\n
        if(c == '\n')
        {
          //ultimo carácter de una línea
          currentLineIsBlank = true;
        }
        //
        else if(c != '\r')
        {
          //caracteres recibidos desde el cliente
          currentLineIsBlank = false;
        }
       }
      }
      //damos tiempo al navegador para recibir datos
      delay(1);
      client.stop();
    }
  }
```

Figura 58. Envió de la web Html desde el servidor Arduino

Como se ha podido observar, establecer un web server sencillo con la tarjeta Arduino, no requiere de grandes conocimientos sobre protocolos ethernet, http o lenguaje html, haciendo a Arduino una plataforma muy flexible para estas tareas.

## 4.5. Webduino web server

Dependiendo del proyecto web que se quiera desarrollar sobre Arduino, el web server mostrado anteriormente puede resultar demasiado rudimentario a la hora de programar varias páginas html e incluso ficheros de estilo Css.

Gracias al carácter *"open source"* del mundo Arduino, podemos aprovecharnos de librerías y códigos fuente que implementan funciones avanzadas. En este caso, podemos utilizar un proyecto muy ambicioso de creación de web servers llamado *Webduino*.

Con Webduino se nos dota de una herramienta muy potente con la que configurar un web server totalmente personalizado. Tendremos la capacidad de crear todas las paginas html que sean necesarias, así como ficheros de estilo Css y Java scripts de forma relativamente sencilla.

Una de los requerimientos más importantes para el uso del proyecto Webduino, es la disponibilidad de recursos de memoria de la tarjeta Arduino utilizada. En caso de que no sean suficientes, Arduino ofrece tarjetas con mayores recursos que son perfectas para configurar un Webduino server. [37]

```
#include "SPI.h"
#include "Ethernet.h"
#include "WebServer.h"

byte mac[] = {0x90,0xA2,0xDA,0x0D,0x3A,0x85};
byte ip[] = {192,168,1,103};

WebServer servidor("", 80);

void helloCmd(WebServer &server, WebServer::ConnectionType type, char *, bool)
{
 server.httpSuccess();

 if (type != WebServer::HEAD){
  P(helloMsg) = "<h1>Hello, World!</h1>";
  server.printP(helloMsg);}
}

void setup()
{
 Ethernet.begin(mac, ip);
 webserver.setDefaultCommand(&helloCmd);
 webserver.addCommand("index.html", &helloCmd);
 webserver.begin();
}

void loop()
{
 webserver.processConnection();
}
```

Figura 59. Ejemplo de pagina web basada en Webduino server

En la figura 59 se muestra un ejemplo sencillo que podemos hacer mediante Webduino, el *"Helloworld"*. A continuacion se va a mostrar un codigo mas complejo, que puede representar un proyecto mas ambicioso de web server, en el se configura una pagina html y un fichero de estilo Css cargado desde una tarjeta micro SD.

```
#include "SPI.h"
#include "Ethernet.h"
#include "WebServer.h"
#include <SD.h>

byte mac[] = {0x90,0xA2,0xDA,0x0D,0x3A,0x85};
byte ip[] = {192,168,1,103};
WebServer webserver("", 80);

void csscmd(WebServer &server, WebServer::ConnectionType type, char *url_tail, bool tail_complete)
{
 server.httpSuccess("text/css");
   if (type == WebServer::GET){
    File fp = SD.open("style.css",FILE_READ);
    if(fp){
     while(fp.available()){
      server.write(fp.read());}
     fp.close();}
 }
}
void pgcmd(WebServer &server, WebServer::ConnectionType type, char *url_tail, bool tail_complete)
{
 server.httpSuccess();
 if (type == WebServer::GET){
    P(head) = "<html><head><title>Arduino Page</title>";
    P(css) = "<link href="style.css" rel="stylesheet" type="text/css"></head>";
    P(body) = "<body><h1>Sensores de temperatura</h1></body></html>";
    server.printP(head);
    server.printP(css);
    server.printP(body);
  }
}
void setup()
{
   if(!SD.begin(4)){
    Serial.println("SD_init_error");}

 Ethernet.begin(mac, ip);
 webserver.setDefaultCommand(&pgcmd);
 webserver.addCommand("index.html", &pgcmd);
 webserver.addCommand("style.css", &csscmd);
 webserver.begin();
}
void loop(){
 webserver.processConnection();}
```

Figura 60. Ejemplo de pagina web con ficheros Html y Css

## 4.6. Comprobación de memoria Sram disponible

Uno de los grandes problemas que pueden surgir a la hora de realizar un proyecto con la plataforma Arduino, es la capacidad de memoria disponible en el microcontrolador utilizado. Nuestra Arduino Ethernet Board dispone de 32 KB de memoria Flash (Programa), de 2 KB de memoria Sram (ejecución) y 1 KB de EEPROM.

El control de la memoria de programa es sencillo, ya que depende del tamaño de nuestro código fuente. Podremos visualizarlo cuando compilemos el código en el entorno de programación Arduino.

Sin embargo, el control de la memoria Sram utilizada, es mucho más complicado ya que por defecto no se dispone de ninguna herramienta para ello. Para poder hacer un control de la Sram, disponemos de un código que incluido en nuestro programa nos mostrara por el terminal serie la cantidad de memoria libre durante la ejecución del código. [38]

```
extern int __bss_end;
extern void *__brkval;

int memoryFree()
{
   int freeValue;
   if((int)__brkval == 0){
   freeValue = ((int)&freeValue) - ((int)&__bss_end);
   }
   else{
   freeValue = ((int)&freeValue) - ((int)__brkval);
   }
  return freeValue;
}

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

void loop()
{
   Serial.println(memoryFree());
}
```

Figura 61. Código para comprobar la memoria Sram disponible

Para controlar el uso excesivo de la memoria disponible en nuestro proyecto, podemos llevar a cabo varias actuaciones a la hora de almacenar nuestras variables. Estas, pueden ser totalmente necesarias en el caso de que nos quedemos sin recursos,

ya que si ocupamos toda la memoria disponible Flash o Sram de la tarjeta Arduino, la ejecución del código tendrá un funcionamiento impredecible.

A continuación se citan algunas de las actuaciones más importantes:

- Cuando se necesite definir constantes en el código, utilizar el tipo *Const*, el cual no utiliza memoria Sram para guardar los valores. Por ejemplo, deberíamos declarar: *const int ledpin = 9;.* [38]

- Otra alternativa para declarar constantes, es utilizar la directiva del preprocesador *#define*, con la que se crea una variable simbólica con un valor asociado, pero que tampoco utiliza memoria Sram. Por ejemplo, deberíamos declarar: *#define ledpin 9*. [38]

- Si se dispone de memoria Flash libre, podemos utilizarla para almacenar nuestros valores o strings y que de este modo no ocupen Sram. Para hacer esto tenemos que utilizar la librería *avr/pgmspace.h*. [39]

```
#include <avr/pgmspace.h>

PROGMEM prog_char srt1[] = "cadena de caracteres1";
PROGMEM prog_char str2[] = "cadena de caracteres2";
PROGMEM prog_char str3[] = "cadena de caracteres3";
char buffer[20];

PGM_P strings PROGMEM = {str1,str2,str3};

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

void loop()
{
  strcpy_P(buffer, (char*)pgm_read_word(&(strings[i])));
  Serial.println(buffer);
}
```

Figura 62. Almacenamiento de cadenas de caracteres en memoria FLASH

- Si tenemos que definir cadenas de caracteres para mostrarlas por el puerto serie o para escribirlas en un archivo, es importante intentar abreviar los mensajes, ya que byte a byte la memoria Sram se ve reducida rápidamente.

```
Serial.print("Error inicializando la tarjeta SD");   //33 bytes
Serial.print("SD init err");                          //11 bytes
```

Figura 63. Ejemplo de reducción de la longitud de las cadenas de caracteres utilizadas

- Si no disponemos de demasiado espacio en la memoria Flash, ya que nuestro programa es grande, nos queda la opción de almacenar datos y strings en la memoria EEPROM *(no volátil)*del microcontrolador. [40]

```
#include <EEPROM.h>

const int direccion = 10;

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

void loop()
{
 int valor = 25;
 int var;

 EEPROM.write(direccion, valor);
 var = EEPROM.read(direccion);
 Serial.print(var);
}
```

Figura 64. Ejemplo de almacenamiento de datos en la memoria EEPROM

- Otra técnica a tener en cuenta, es la planificación de nuestro código, es decir, podemos programar eligiendo entre el tamaño o la velocidad del código, siempre y cuando nuestras necesidades y recursos así lo permitan.

  Si utilizamos códigos más lentos en su ejecución, ocuparemos en muchos casos menos memoria de programa. Esto no siempre se cumple así como se ve en el segundo ejemplo del puntero.

```
//opción de menor tamaño
//opción mas lenta
for(i=0;i<5;i++)
{
    serial.print(port[i].mux);
}
```

```
//opción mas grande
//opción mas rápida
Serial.print(port[0].mux);
Serial.print(port[1].mux);
Serial.print(port[2].mux);
Serial.print(port[3].mux);
Serial.print(port[4].mux);
```

```
//opción menos rápida
//opción mas grande
datatemp *pt;
pt=&port[0];
pt->puerto;
pt->mux;
pt->interval;
```

```
//opción mas rápida
//opción de menor tamaño
port[0].puerto;
port[0].mux;
port[0].interval;
```

Figura 65. Estrategias de programación basadas en el tamaño del código y en la rapidez de su ejecución

# Capítulo 5

# Análisis de datos y diseño del Pcb

En este capítulo se va exponer un modo para poder analizar de forma eficaz los archivos de datos que quedan almacenados en nuestra tarjeta de memoria. Debido a la gran popularidad entre los usuarios del sistema operativo Windows, se utiliza una herramienta para representar gráficos de código libre llamada *GnuPlot*, la cual inicialmente se implemento en sistemas Unix y más adelante en Windows.

Con esta herramienta, podremos abrir los archivos de datos y representarlos en forma de graficas temporales, para realizar un análisis temporal donde observar las principales variaciones de temperatura acontecidas.

## 5.1 Utilización de GnuPlot

Para utilizar el software de Gnuplot sobre Windows, primero debemos descargarlo desde su página oficial [41], en este link podemos encontrar todas las versiones del programa. Descargamos el instalador para Windows o el archivo Zip que nos permite utilizar el programa sin tenerlo instalado.

Como recomendación se propone descargar el archivo Zip *"gp463-win32.zip"*, que es la última versión del programa, y descomprimirlo en la raíz del sistema *C:/*. Para ejecutar el programa debemos ir a *C:/gnuplot/bin/wgnuplot.*exe que nos abrirá un terminal similar al utilizado en sistemas Unix:

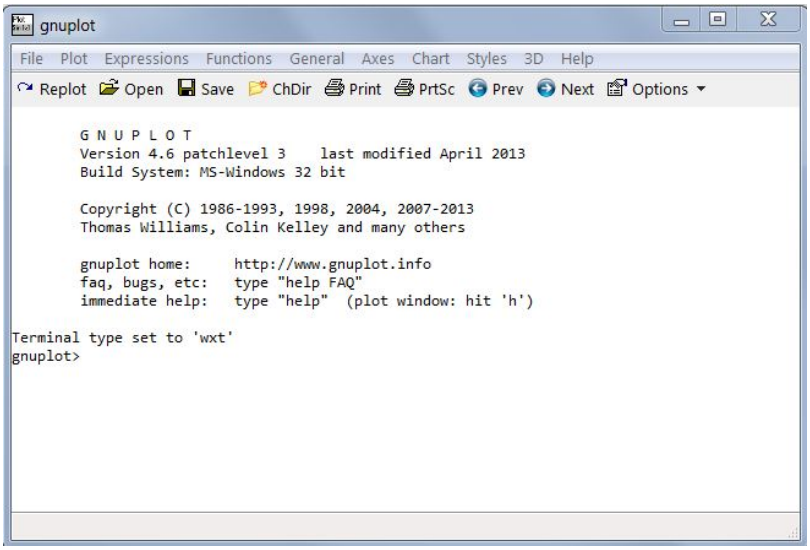



Figura 66. Terminal de comandos de Gnuplot sobre el sistema Windows

La utilización de los comandos de gnuplot, no tiene ninguna diferencia entre su versión Unix o Windows, por lo que no existe ningún inconveniente en utilizar un sistema operativo o otro.

Para analizar gráficamente los archivos de datos de Arduino, se propone que sean almacenados en una carpeta nueva dentro del directorio de gnuplot, en concreto en el directorio *C:\gnuplot\datalogger_data.*

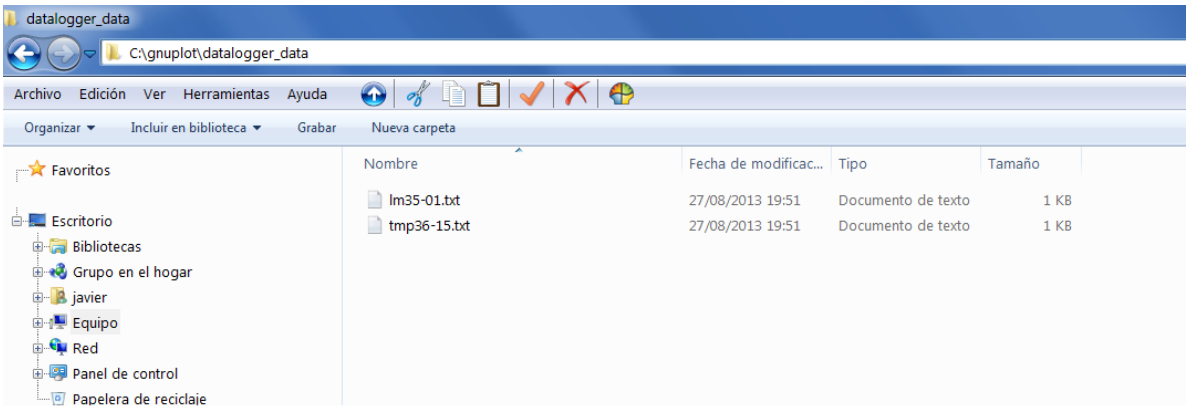



Figura 67. Directorio nuevo situado dentro del directorio de Gnuplot

Como ejemplo de análisis, se han creado dos archivos que poseen el mismo formato de datos que los archivos recogidos del registrador. Para analizar uno de ellos, debemos escribir las siguientes ordenes en el terminal de gnuplot:

```
reset
clear
cd '../datalogger_data'
set grid
set xdata time
set timefmt "%Y/%m/%d %H:%M:%S"
set xrange [*:*]
set yrange [0:*]
set format x "%H:%M:%S"
set xtics 10 scale 0.5 rotate by -45
set title 'tmp36-15'
plot 'tmp36-15.txt' using 1:3 with linespoints
cd '../bin'
```

Por defecto, el *Path* está colocado en la ruta del ejecutable, por lo que debemos moverlo a la carpeta de datos creada anteriormente.

Gnuplot permite establecer los datos en el eje x como datos temporales, pero debemos indicarle como debe leerlos. Con la instrucción *set timefmt* le definimos el formato de los datos temporales que posee el fichero.

Figura 68. Instrucciones de Gnuplot para analizar los archivos de datos del registrador

Los rangos de la grafica *xrange e yrange* se dejan ajustarse automáticamente mediante el programa introduciendo un *'*'* en el campo deseado. De este modo no se requieren más órdenes para que la grafica quede dentro de los limites.

Para que las graficas muestren claramente los puntos de temperatura frente al tiempo de muestreo, se debe configurar el eje x mediante *set format x y set xtics*. Con la instrucción *set format x* declaramos el formato de la hora (HH:MM:SS) en el eje x, y con *set xtics 10* declaramos el incremento.

Por último, utilizando la orden *plot* pintamos los datos del archivo utilizando la columna 1 de la fecha y la columna 3 de las temperaturas *(1:3).* [42]

La grafica obtenida del fichero *'tmp36-15.txt'* mediante las ordenes introducidas de la figura 68 tiene el siguiente aspecto:

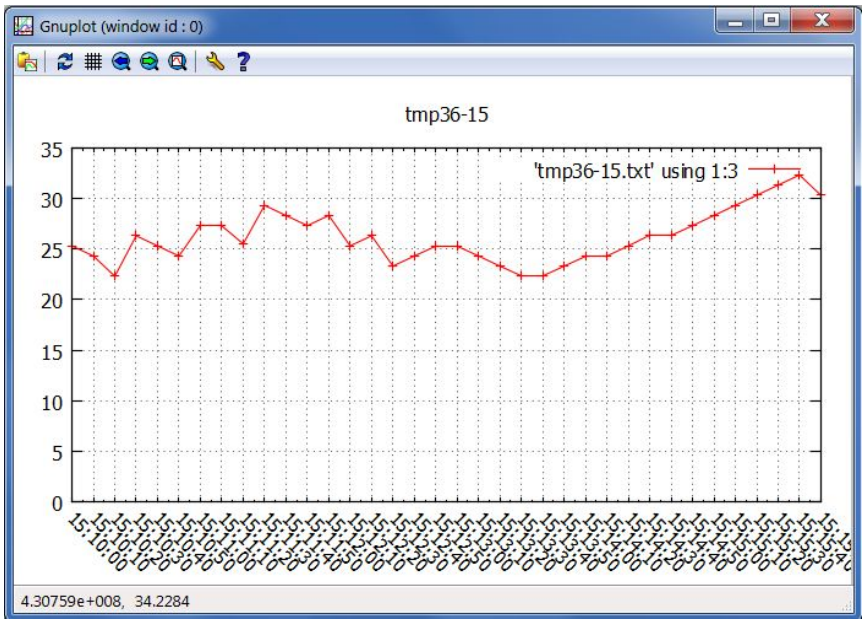



Figura 69. Aspecto gráfico de un archivo de temperaturas del registrador

Si la ventana donde aparece nuestra grafica tiene poca resolución, podemos ampliarla haciendo la ventana más grande con el ratón y después escribiendo *replot* en el terminal para que sea dibujada en el nuevo tamaño de ventana.

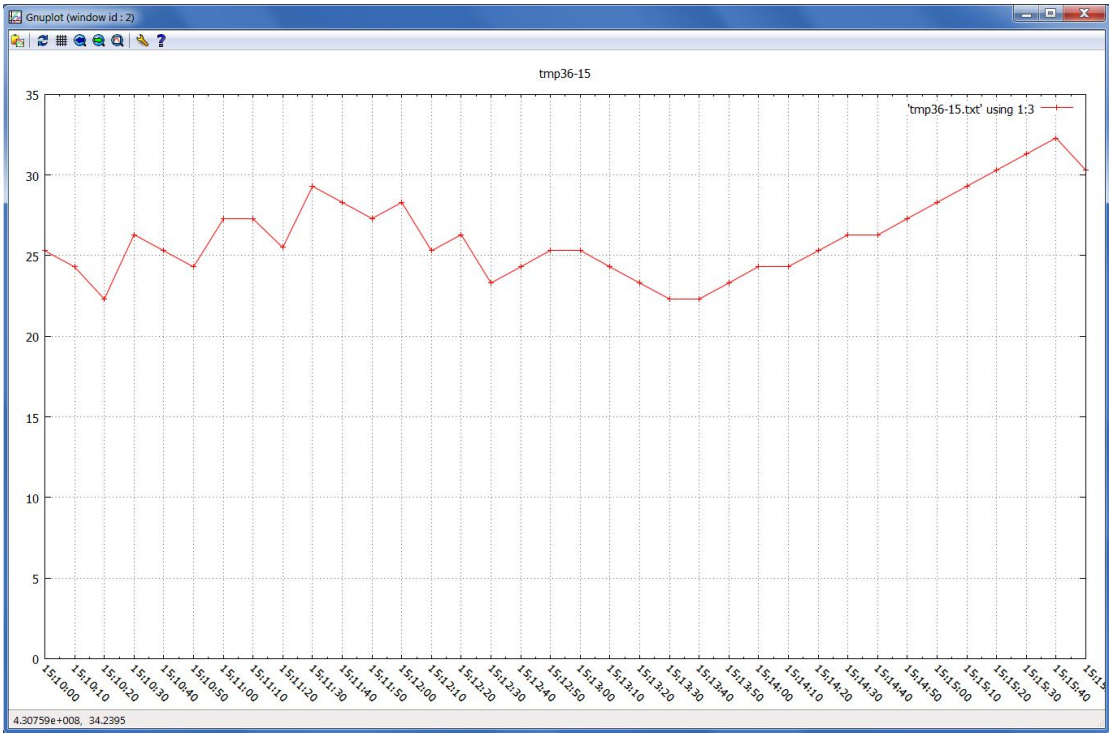



Figura 70. Grafica figura 69 con mayor resolución

Cuando necesitemos analizar varios archivos de datos del registrador Arduino, el método presentado antes puede resultar muy repetitivo. Gracias a gnuplot, podemos editar varios archivos o *'scripts'* que realicen las operaciones de forma secuencial.

Si quisiéramos ver las graficas resultantes de dos archivos de datos distintos, primero debemos editar sus scripts y después estos serán ejecutados desde otro script general que los llama. Todos los scripts escritos, han sido colocados para mayor comodidad en la misma carpeta donde se encuentre el ejecutable de gnuplot *(c:/gnuplot/bin).* [42]

- Scripts para los archivos *tmp36-15.txt y lm35-01.txt:*

```
//script para tmp36-15.txt
reset
cd '../datalogger_data'
set term wxt 1
set grid
set xdata time
set timefmt "%Y/%m/%d %H:%M:%S"
set xrange [*:*]
set yrange [0:*]
set format x "%H:%M:%S"
set xtics 10 scale 0.5 rotate by -45
set title 'tmp36-15'
plot 'tmp36-15.txt' using 1:3 with linespoints
cd '../bin'
```

```
//script para lm35-01.txt
reset
cd '../datalogger_data'
set term wxt 2
set grid
set xdata time
set timefmt "%Y/%m/%d %H:%M:%S"
set xrange [*:*]
set yrange [0:*]
set format x "%H:%M:%S"
set xtics 5 scale 0.5 rotate by -45
set title 'lm35-01'
plot ' lm35-01.txt' using 1:3 with linespoints
cd '../bin'
```

Figura 71. Ordenes personalizadas para cada archivo de datos

- Script general para llamar a los script de cada archivo de datos:

```
load "analisis01.gnu"
load "analisis15.gnu"
.....
.....
.....
```

Figura 72. Contenido del script general. Carga los scripts particulares

- Para ejecutar el script general, nos dirigimos al terminal de gnuplot y escribimos la orden *load* junto con el nombre dado a nuestro script:

```
load "arduino_script.gnu"
```

Figura 73. Orden para cargar el script general. Ha sido nombrado 'arduino_script.gnu'

## 5.2 Diseño del Shield sobre un Pcb

En la fase de pruebas del registrador de datos, se ha utilizado una placa de prototipos Board para hacer los montajes necesarios para comprobar el funcionamiento del sistema. Esto nos ha permitido que fácilmente se puedan agregar o sustituir partes del circuito hasta conseguir la correcta operación de este.

Una vez depurado el funcionamiento del sistema y con vistas a ser utilizado por un usuario, se hace necesario realizar un pequeño diseño de circuito impreso sobre Pcb, que pueda albergar de forma compacta todos los componentes utilizados en el sistema registrador de datos.

Para realizar esta tarea, vamos a utilizar un software llamado *CadSoft EAGLE Pcb*, con el que podemos editar nuestro archivo esquemático, mas adelante nuestro archivo de Layout y finalmente nos permitirá realizar un auto enrutado de las pistas.

Para descargar nuestra copia del software EAGLE, nos dirigimos a la página principal [43]. Desde este lugar podemos descargar la versión para Windows o Unix, nosotros vamos a descargar la versión para Windows, aunque no hay ninguna diferencia en función del sistema operativo utilizado.

El programa EAGLE es inicialmente un software de pago, pero nos podemos aprovechar en este caso de usarlo de forma gratuita. Podemos establecer en el momento de la instalación, utilizar el software de forma gratuita con la opción *'Run as a Freeware'*, la cual permite usarlo libremente con algunas limitaciones: [44]

- El área utilizable del Pcb estará limitado al tamaño 100x80 mm.
- Solo podrán utilizarse dos capas de pistas, Top y Bottom.
- En el editor de esquemáticos solo podremos crear un archivo.
- El soporte solo se realizara a través de E-mail o desde el foro oficial.

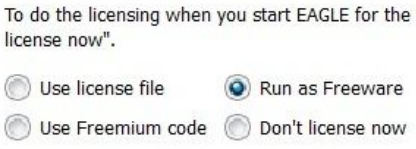



Figura 74. Detalle para establecer el software Eagle en modo gratuito

Si decidimos no especificar la licencia en este punto, podemos establecer la opción *Freeware* mas adelante desde el panel de control de EAGLE. Vamos al menú *Help* y después a *EAGLE License:*

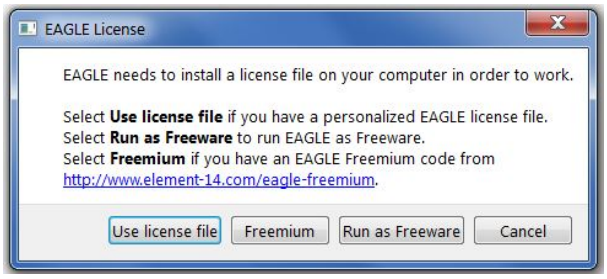



Figura 75. Establecer el modo gratuito una vez instalado el software

### 5.2.1 Instalación de librerías de componentes

Una vez instalado el programa EAGLE, se nos mostrara la ventana principal que representa el panel de control del programa. En esta ventana podemos ver las librerías de las que dispone el programa por defecto, y podemos crear nuestro proyecto nuevo y su correspondiente esquemático.

En el apartado *Projects*, se encuentra la carpeta *Eagle*, sobre esta haciendo click derecho podemos crear un proyecto nuevo. Dentro del proyecto nuevo, haciendo click derecho creamos un nuevo esquemático.

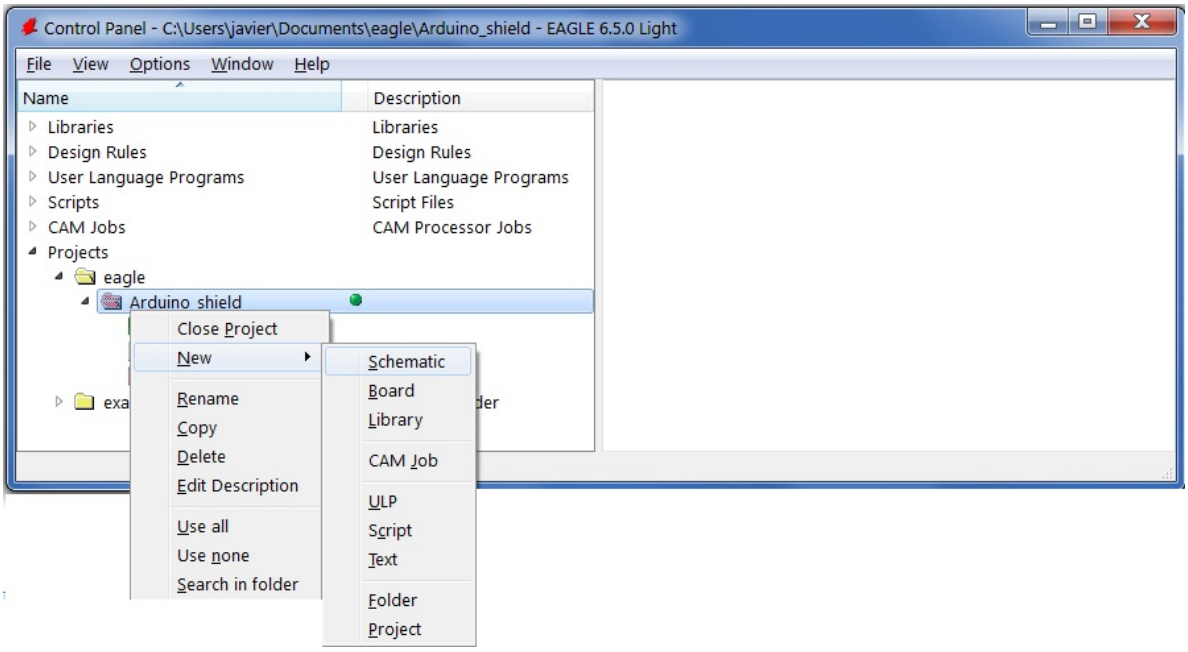



Figura 76. Detalle de la creación de un nuevo proyecto esquemático

Para poder disponer de los componentes utilizados en nuestro proyecto, como el reloj en tiempo real, la tarjeta Arduino con sus entradas y salidas, etc, es imprescindible que utilicemos alguna librería especifica que nos brinde todo estos componentes en forma de símbolo esquemático y también de huella para el layout.

Debido a la relación con el mundo Arduino de varias páginas web, como son *Adafruit y Sparkfun*, tenemos el posibilidad de utilizar sus propias librerías de EAGLE con todos los componentes más relacionados con el mundo Arduino.

Descargamos desde internet las librerías mencionadas desde: [45] [46]

- https://github.com/adafruit/Adafruit-Eagle-Library
- https://github.com/sparkfun/SparkFun-Eagle-Libraries

Para incluir estas librerías en el programa EAGLE y que sean accesibles debemos descomprimir los archivos *Zip* descargados. Obtendremos así una carpeta para cada una de ellas con los archivos **.lbr* que contienen la definición de los componentes de Arduino. [47]

Teniendo en cuenta el directorio de instalación elegido por cada usuario, nos dirigimos a este y copiamos las carpetas obtenidas dentro de la carpeta *lbr*.

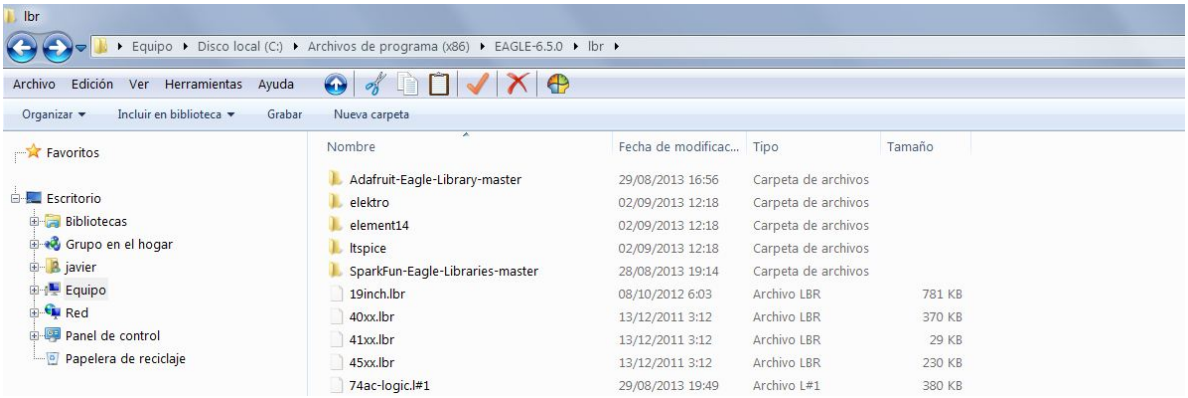



Figura 77. Directorio 'lbr' donde se deben copiar las librerías deseadas

Una vez hecho esto, debemos reiniciar el programa EAGLE y prestar atención al panel de control de este. Se pueden observar, si expandimos el apartado llamado *'libraries'*, las carpetas llamadas *Adafruit y Sparkfun*. Inicialmente se encuentran desactivadas, por lo que debemos hacer click derecho sobre la carpeta y seleccionar *Use All*. Después de hacer esto se colocara al lado de cada librería un circulo verde que indica que ya esta activa. [47]

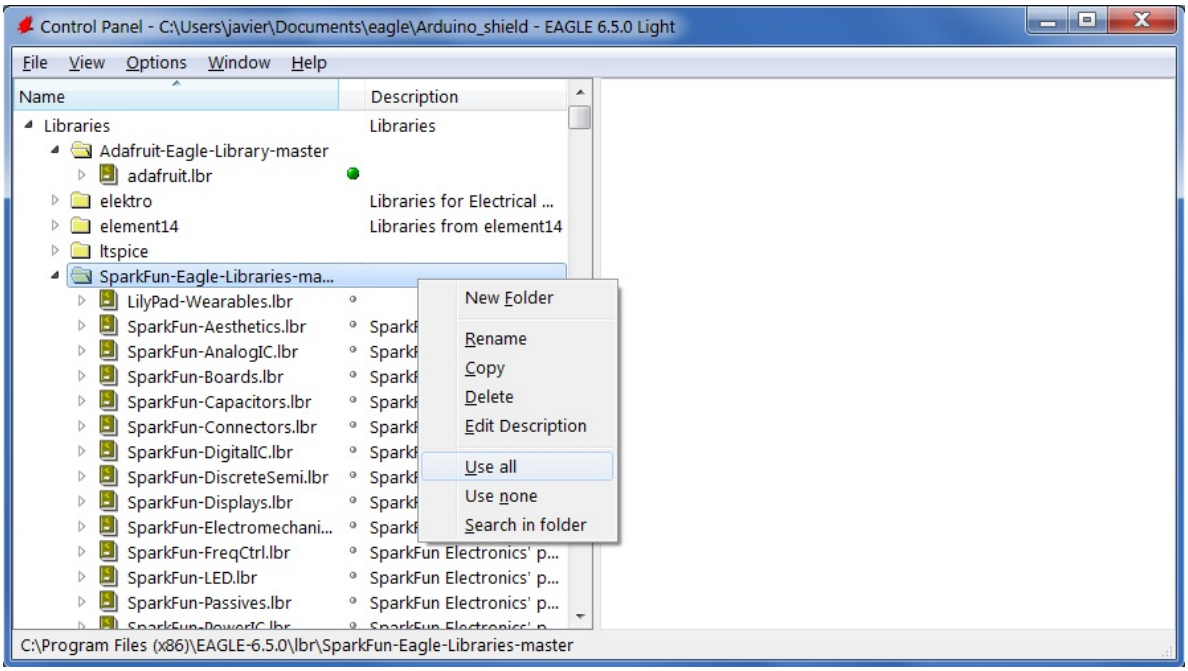



Figura 78. Detalle de la activación de las librerías desde el panel de control de Eagle

### 5.2.2 Realización del circuito esquemático

Una vez añadidas las librerías de componentes necesarias, podemos pasar a construir nuestro archivo esquemático donde colocaremos y conectaremos todos los componentes utilizados para el registrador de datos. [48]

Inicialmente conviene visualizar una rejilla en la hoja del esquemático, para así poder colocar los componentes con una separación estándar de una decima de pulgada. Vamos al menú *View/Grid* y establecemos un *Size* de 0.1 pulgadas (inches).

Después de esto comenzamos a incluir los componentes con el botón *Add* : 

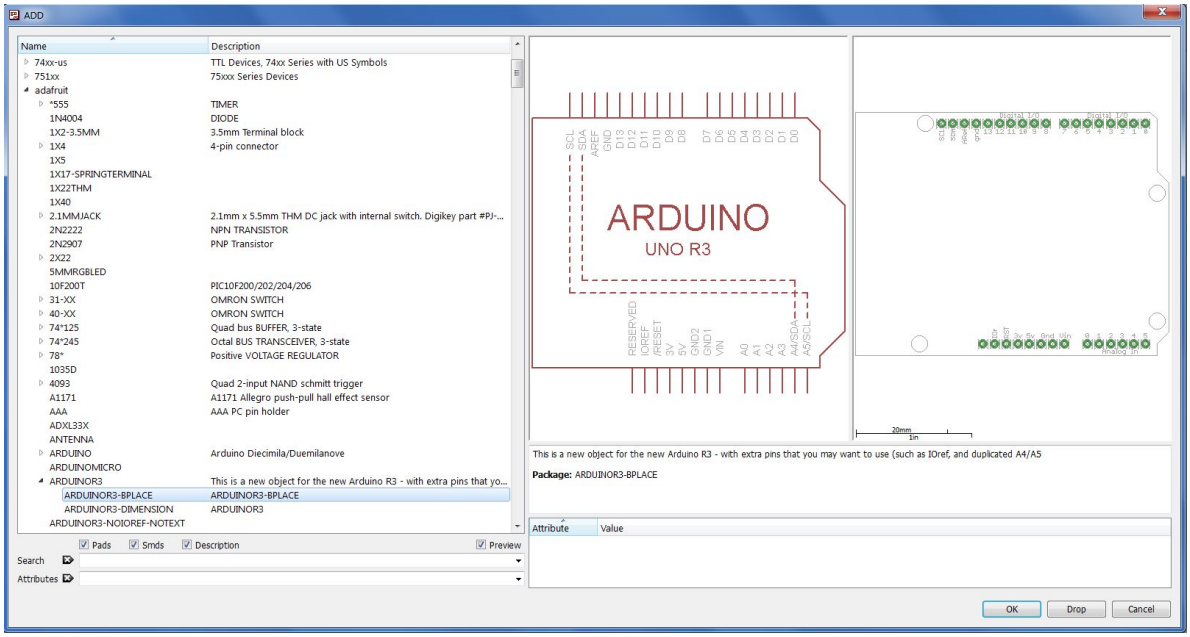



Figura 79. Ventana desde la que introducimos los componentes necesarios

En la imagen podemos observar que se ha buscado la placa Arduino Uno R3 dentro de la librería incluida de Adafruit. Este componente tiene las mismas dimensiones que la tarjeta utilizada por nosotros, por lo que el *Shield* que vamos a diseñar deberá tener la misma dimensión y distribución de conexiones.

Cabe destacar que aparte de las librerías que hemos añadido anteriormente, el propio software EAGLE dispone de multitud de librerías con componentes de propósito general que también tendremos que utilizar.

En la imagen superior, se observa que durante la búsqueda de componentes podemos visualizar rápidamente el símbolo esquemático y al lado la huella o *'footprint'* que veremos más adelante al realizar el layout y el enrutado de pistas. El hecho de disponer de un *footprint* diseñado específicamente para las tarjetas Arduino, facilita enormemente la realización de esquemáticos y layouts.

Para realizar el conexionado de los componentes incluidos, se recomienda realizar la herramienta *Net* en vez de *Wire* , ya que realiza una conexión más segura. Una vez realizada la conexión de todos los componentes, hemos obtenido un esquemático que tiene el siguiente aspecto:

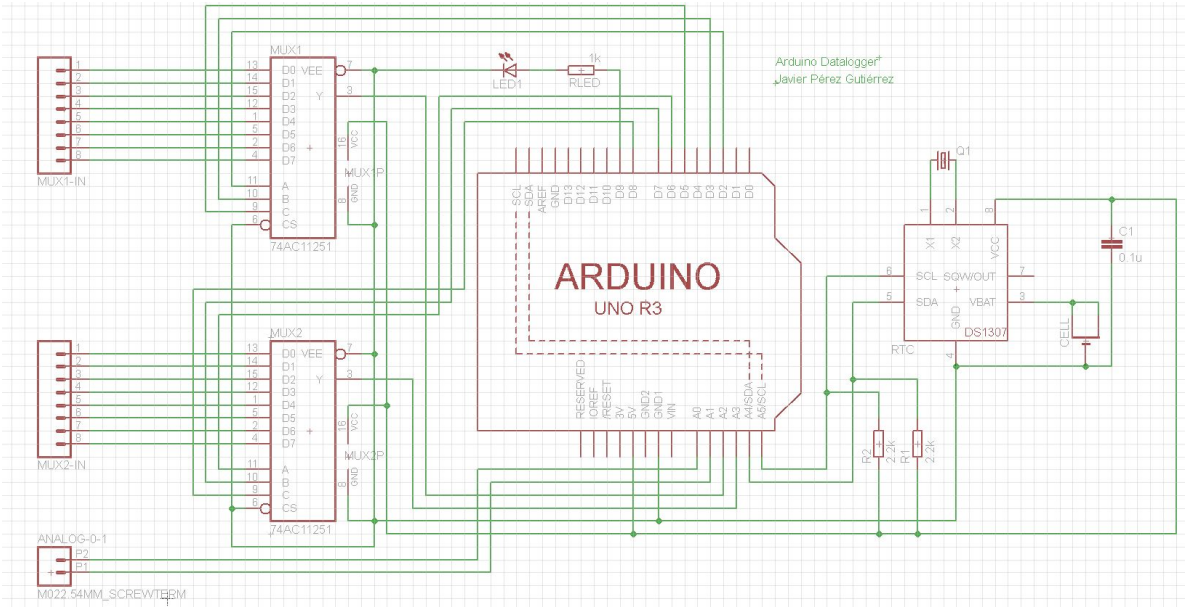



Figura 80. Circuito esquemático completo

En el esquemático podemos observar todos los componentes necesarios para construir nuestro propio *Shield* de Arduino. A continuación se hace un listado de todos ellos y se indica desde que librería de EAGLE han sido incluidos:

| *Componente* | *Librería* | *Dispositivo* |
|---|---|---|
| Tarjeta Arduino | Adafruit | ARDUINOR3-DIMENSION (ARDUINOR3) |
| Multiplexores 8 a 1 | 74ac-logic | 74AC11251 *(modificado para obtener distribución de pines MC14051BCP)* |
| Conectores macho 8 pines | SparkFun-Connectors | M083.5MM-8 (M08) |
| Conectores macho 2 pines | SparkFun-Connectors | M022.54MM_SCREWTERM (M02) |
| Chip DS1307 (RTC) | Adafruit | DS1307 |
| Resistencias 2.2K | rcl | R-EU_0207/10 (R-EU_) |
| Condensador 0.1uF | rcl | C-EU050-025X075 (C-EU) |
| Cristal oscilador | Adafruit | CRYSTALTC26H (CRYSTAL) |
| Zócalo batería 3V | Adafruit | CR1220THM (CR1220) |
| Diodo Led | Adafruit | LED5MM (LED) |
| Resistencia led 1K | rcl | R-EU_0207/10 (R-EU_) |

Tabla 4. Relación de los nombres de los componentes

### 5.2.3 Realización del Layout y enrutado

Cuando ya tenemos finalizado el archivo esquemático con todos los componentes conectados, estamos en condiciones de comenzar a editar el archivo de layout. En el layout, vamos a encontrarnos con las huellas o *'footprints'* correspondientes a los componentes utilizados. [49]

Para editar el layout debemos apretar el botón *'Generate/Switch to board'*: 

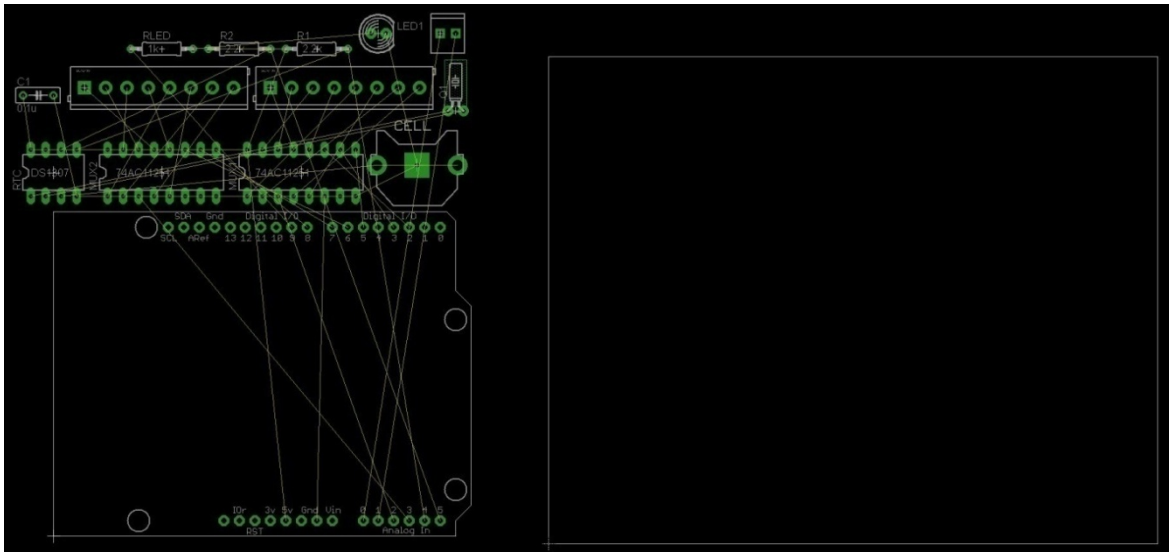


Figura 81. Pantalla inicial del editor de layout

Inicialmente, podemos ver los *footprints* agrupados con las conexiones representadas mediante los cables amarillos. Al lado vemos un rectángulo que representa el área disponible para colocar nuestros componentes. Este área está limitado al tamaño que se observa ya que estamos utilizando la versión gratuita del software EAGLE.

Como podemos ver, aunque el tamaño máximo del Pcb este limitado, nos sirve perfectamente para diseñar un *Shield* de Arduino, ya que este tiene unas dimensiones inferiores.

Igual que en el archivo esquemático, es muy útil activar la rejilla o cuadricula sobre la pantalla, porque nos permitirá colocar de forma exacta los *footprints* sobre la tarjeta. Para ello vamos de nuevo al menú *View/Grid*, pero ahora vamos a establecer una cuadricula más pequeña 0.05 pulgadas:

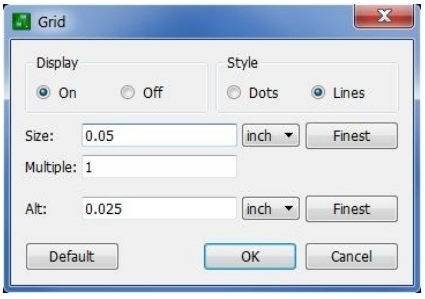



Figura 82. Método para establecer la rejilla en el editor de layout. (También valido en el editor de esquemáticos)

Haciendo uso del botón *move* , vamos introduciendo los componentes dentro del recuadro blanco y los vamos colocando de modo que las líneas de conexión no queden demasiado cruzadas, ya que esto facilitara la operación de enrutado de pistas.

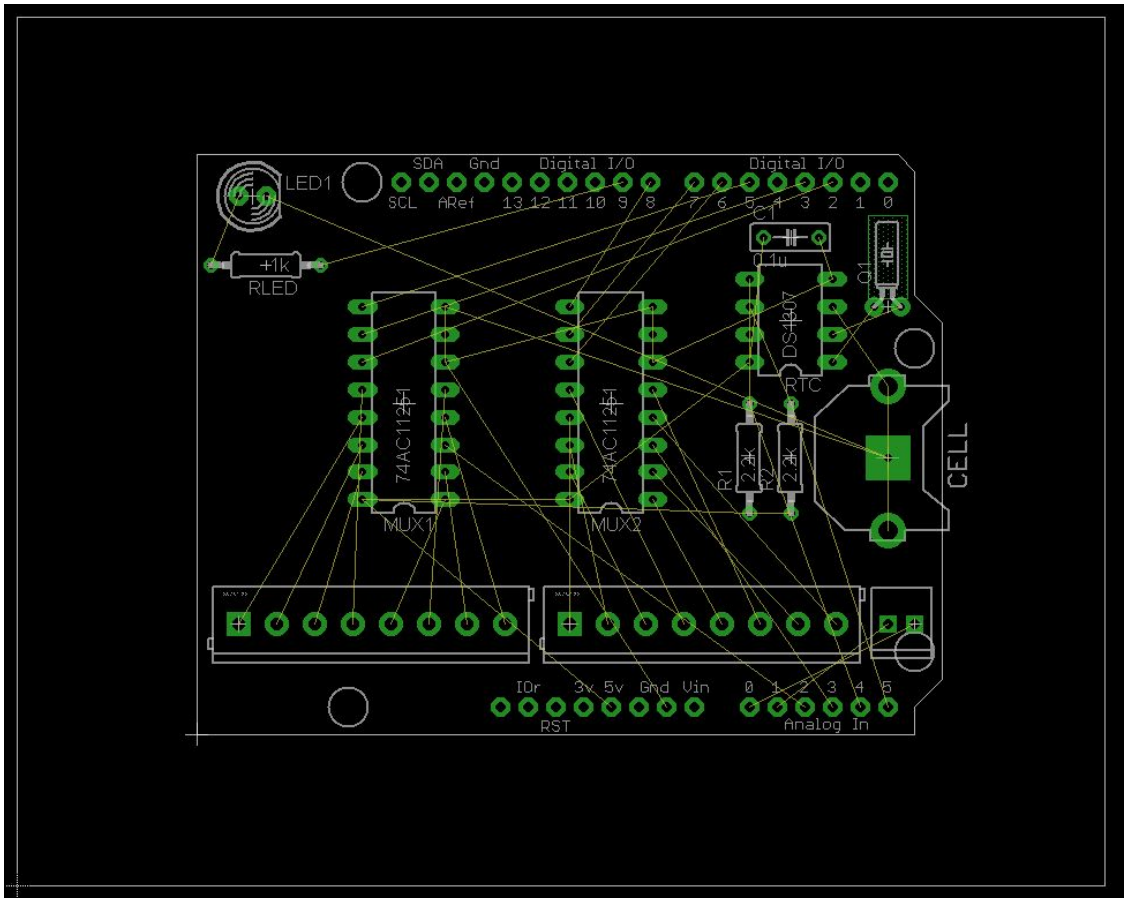



Figura 83. Todos los componentes colocados dentro de los límites de la tarjeta Arduino

Como podemos ver en la figura 83, es imposible distribuir los elementos de forma que las conexiones no queden cruzadas. Esto no supone un problema importante, ya que disponemos de un enrutador que utilizara dos capas para dibujar las pistas. Las capas son la superior o *Top* y la inferior o *Bottom*.

Llegados a este punto, tenemos distribuidos los componentes de nuestro Pcb por lo que podemos hacer el enrutado de estos. En la operación de enrutado automático , el software buscara el modo de resolver las conexiones utilizando las dos capas disponibles.

La operación de enrutado, se puede realizar de modo manual con EAGLE, mediante el botón *Route* . En la figura 84 se muestran las opciones disponibles:


Figura 84. Opciones en el modo manual de enrutado, capa, línea, grosor, etc

En este caso, se ha utilizado la opción de enrutado automático para resolver las conexiones de nuestra placa. Presionando el botón *Autorouter* , accedemos a una ventana con una serie de configuraciones. En principio, se han dejado en los valores por defecto.

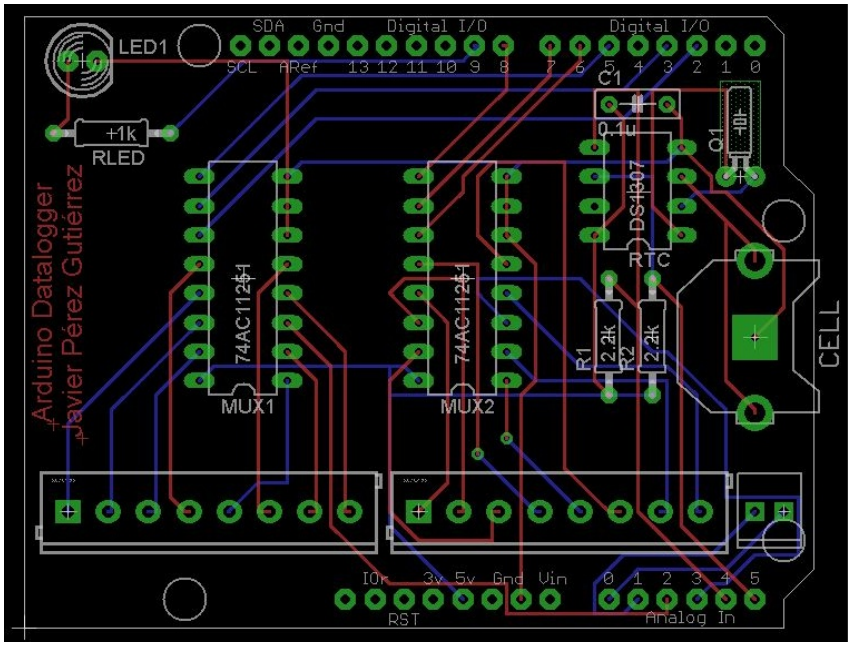



Figura 85. Placa final enrutada de forma automática

Para mejorar la estabilidad del nivel de referencia de 0V de nuestro diseño, se ha establecido un plano de masa en la capa *Bottom* mediante la herramienta *Ratsnest* . Antes de todo debemos dibujar un polígono , por el perímetro de nuestra placa y situado en la capa *Bottom*. Después usaremos la herramienta *Ratsnest*.

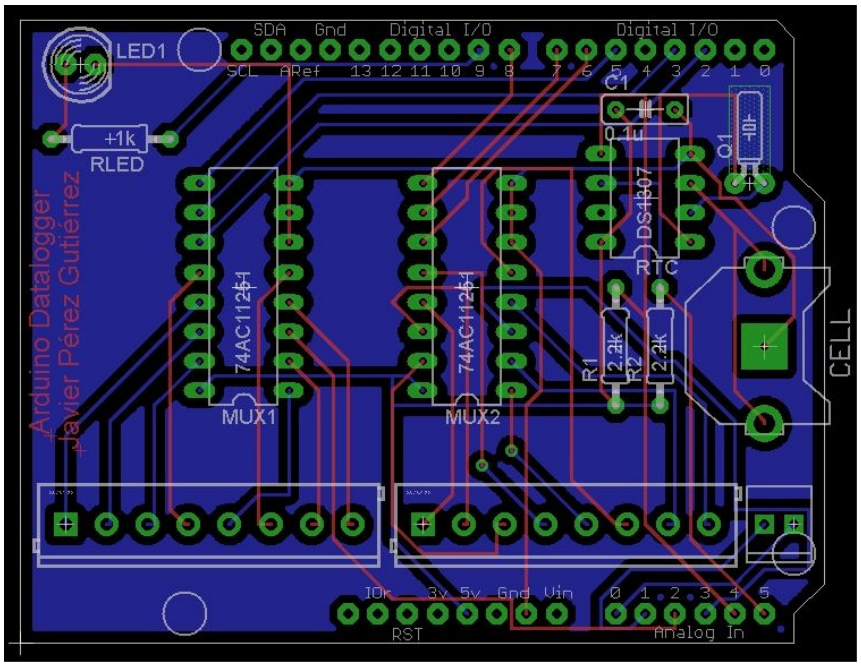



Figura 86. Placa final enrutada y con un plano de masa

# Capítulo 6

# Conclusiones y presupuesto

En este capítulo se expondrá el presupuesto total fijado para llevar a cabo el proyecto del registrador de datos. En base al coste de los materiales que se han utilizado, se realizara una comparación con algunas de las soluciones ya existentes en el mercado.

Este punto es realmente importante, ya que se va a mostrar la rentabilidad económica y la flexibilidad del sistema que se ha diseñado en el proyecto, en comparación con soluciones similares que pudieran ser atractivas a primera vista.

## 6.1 Presupuesto del proyecto

Los componentes utilizados en este proyecto, se han presupuestado en base a los precios establecidos en la web de componentes electrónicos RS España. En ella, se ha podido encontrar todo lo necesario como las tarjetas Arduino, los sensores de temperatura o los multiplexores.

Inicialmente, se ha hecho un presupuesto más extenso con varias opciones de compra buscadas a través de varias webs, que después ha sido ajustado a la realidad del proyecto. De esta forma, hemos elaborado el presupuesto final desde RS España: [50]

| *Cantidad* | *Concepto* | *Ref.* | *Precio unidad* |
|---|---|---|---|
| 1 | **Arduino Ethernet Board sin PoE**<br>*Descripción:* Posee las mismas características de la UNO+Ethernet shield. No programada por Usb sino por cable Ftdi-Usb-serial.<br>*Link:* http://arduino.cc/en/Main/ArduinoBoardEthernet | (RS) 748-5313 | 42€ |
| 1 | **Reloj DS1307**<br>*Descripción*: Chip programable que permite conocer la fecha y hora actual. Encapsulado PDIP-8. Posibilidad de backup mediante batería de 3V.<br>*Link:* http://es.rs-online.com/web/p/relojes-de-tiempo-real/5402726/ | (RS) 540-2726 | 3.7€ |
| 1 | **Cristal cuarzo 32.25 MHz**<br>*Descripción*: cristal resonador de cuarzo de 32MHz de frecuencia.<br>*Link:* http://es.rs-online.com/web/p/unidades-de-cristal/2261910/ | (RS) 226-1910 | 1.79€ |
| 1 | **Batería de litio 12mm 3.3V**<br>*Descripción:* batería de botón de 12 mm de diámetro con tensión de 3.3V<br>*Link:* http://es.rs-online.com/web/p/pilas-de-boton/7450929/ | (RS) 745-0929 | 1.4€ |

| | | | |
|---|---|---|---|
| 1 | **Zócalo para batería 12mm**<br>*Descripción:* zócalo para albergar baterías de botón de 12 mm<br>*Link:* http://es.rs-online.com/web/p/soportes-accesorios-de-montaje-para-baterias/2197926/ | (RS) 219-7926 | 0.8€ |
| 2 | **Resistencia película metálica 0.25W**<br>*Descripción:* resistencia de 0.25W de potencia<br>*Link:* http://es.rs-online.com/web/p/resistencias-fijas-de-orificio-pasante/0148720/ | (RS) 148-720 | 0.06€ |
| 1 | **Cable de interfaz, USB 2.0 A-B 1 m**<br>*Descripción:* cable de programación Usb solo para la placa UNO<br>*Link:* http://es.rs-online.com/web/p/latiguillos-usb/7587490/ | (RS) 758-7490 | 2.5€ |
| 1 | **USB FTDI cable TTL 232R 3.3V**<br>*Descripción:* cable programación serial solo para la placa Arduino Ethernet Board.<br>*Link:* http://es.rs-online.com/web/p/kits-de-desarrollo-de-interfaz/6877780/ | (RS) 687-7780 | 17,6€ |
| 1 | **Full Breadboard PCB Module**<br>*Descripción:* Placa Board externa a Arduino para conectar todos los sensores y circuitería necesaria.<br>*Link:* http://es.rs-online.com/web/p/kits-de-desarrollo-de-procesador-y-microcontrolador/7589367/ | (RS) 758-9367 | 7€ |
| 1 | **Fuente Mascot 7.5vcc 5.5w**<br>*Descripción:* Fuente de alimentación enchufable. Conector 2.1x5.5 mm compatible con placa Arduino UNO.<br>*Link:* http://es.rs-online.com/web/p/fuente-de-alimentacion-enchufable/0869483 | (RS) 869-483 | 24€ |
| 1 | **Tarjeta Kingston microSDHC 8gb**<br>*Descripción:* Tarjeta de memoria para utilizar como almacenamiento de datos. Se utilizaría con el Ethernet Shield.<br>*Link:* http://es.rs-online.com/web/p/tarjetas-sd/6957334 | (RS) 695-7334 | 8€ |
| 3 | **Sensor temperatura TMP36GT9Z**<br>*Descripción:* Alimentación entre 2.7-5.5v, calibrado directamente en grados Celsius, operación entre -40 y 125ºc. Conexión sencilla<br>*Link:* http://es.rs-online.com/web/p/sensores-de-humedad-y-temperatura/0427351/ | (RS) 427-351 | 1.5€ |
| 3 | **Sensor Temperatura LM35DZ**<br>*Descripción:* Alimentado entre 4-30v, Sensor calibrado directamente en grados Celsius, operación entre -55 y 150ºc. Similar al TMP36, conexión sencilla.<br>*Link:* http://es.rs-online.com/web/p/sensores-de-humedad-y-temperatura/5335907/ | (RS) 533-5907 | 1.5€ |
| 2 | **Multiplexores MC14051BCP**<br>*Descripción:* multiplexor de 8 canales a 1. Encapsulado PDIP de 16 pines. Alimentación a 5V compatible con la Arduino Board.<br>*Link:* http://es.rs-online.com/web/p/circuitos-integrados-de-conmutador-multiplexor/5190300/ | (RS) 519-0300 | 0.6€ |

Tabla 5. Presupuesto detallado del proyecto

El coste total del proyecto añadiendo el correspondiente IVA resulta:

| | |
|---|---|
| Subtotal | 119.11€ |
| IVA 21% | 25€ |
| **TOTAL** | **144.1€** |

Tabla 6. Coste del proyecto

## 6.2 Registradores de datos comerciales

En el mercado, podemos encontrar gran variedad de registradores de datos similares al que hemos diseñado aquí. Por ello, vamos a hacer una comparación de las características y precios encontrados en la web RS España, la cual es una excelente referencia debido a su gran variedad de productos.

La comparación se hará en base a las características ofrecidas en los modelos comerciales, en contra de lo ofrecido por nuestro proyecto Arduino. En materia de registradores de datos, es importante el tiempo de muestreo, el número de entradas para sensores, la capacidad de medir magnitudes, etc. [50]

| ***Nombre (Ref.)*** | ***Características*** | ***Precio (Sin IVA)*** |
|---|---|---|
| Testo mini datalogger 174T (712-5465) | *Sensor:* Termistor NTC interno<br>*Ciclo medida:* desde 1 minuto hasta 24 horas<br>*Rango temp:* desde -30C hasta +70C<br>*Precisión:* ±0.5C<br>*Numero Canales:* 1 (interno)<br>*Memoria:* hasta 16000 medidas<br>*Software monitor:* Si | 52.5€ |
| N2011 comark diligence EV (399-0221) | *Sensor:* Termistor NTC interno<br>*Ciclo medida:* desde 1 segundo hasta 99 horas<br>*Rango temp:* desde -40C hasta +70C<br>*Precisión:* ±0.1C<br>*Numero Canales:* 1 (interno)<br>*Memoria:* hasta 16000 medidas<br>*Software monitor:* Si | 108.2€ |
| TinytagView2 (283-921) | *Sensor:* Termistor NTC interno<br>*Ciclo medida:* desde 1 segundo hasta 10 días<br>*Rango temp:* desde -25C hasta +50C<br>*Precisión:* ±0.02C<br>*Numero Canales:* 2 (interno)<br>*Memoria:* hasta 30000 medidas<br>*Software monitor:* Si | 216.3€ |

| | | |
|---|---|---|
| Usb TC-08<br>(492-5105) | *Sensor:* Termopar<br>*Ciclo medida:* 10 medidas/segundo máximo<br>*Rango temp:* desde -270C hasta +1820C<br>*Precisión:* ±0.025C<br>*Numero Canales:* 8 (externos)<br>*Memoria:* almacenaje en ordenador<br>*Software monitor:* Si | 301€ |

Tabla 7. Características de algunos registradores de datos comerciales

## 6.3 Conclusiones

Con el análisis realizado en el apartado anterior, podemos realizar una comparación de características y precios entre nuestro sistema registrador de datos y los vistos anteriormente. A continuación se presentan las principales características de nuestro sistema Arduino:

| ***Nombre (Ref.)*** | ***Características*** | ***Precio (Sin IVA)*** |
|---|---|---|
| Registrador de datos basado en Arduino | *Sensor:* Discretos, Tmp36 y Lm35<br>*Ciclo medida:* configurable desde 1 segundo sin límite.<br>*Rango temp:* de -40C hasta +69C para Tmp36, de -55C hasta 110C para Lm35<br>*Precisión:* ±1C para Tmp36 y ±0.5C para Lm35<br>*Numero Canales:* 18 (externos)<br>*Memoria:* tarjeta micro SD 8GB<br>*Software monitor:* No | 116€ |

Tabla 8. Resumen de características de nuestro registrador de datos

Como se puede observar, nuestro sistema Arduino posee más características a favor que en contra con los examinados arriba. Si nos fijamos en los tipos de sensores usados, veremos que los basados en termistor o termopar son más precisos que los nuestros, pero hay que recordar que con la plataforma Arduino podemos utilizar cualquier sensor *(realizando su montaje de configuración necesario)*, obteniendo una precisión de 1mV, mediante la conversión ADC de 10 bits disponible en nuestra tarjeta Arduino.

Los tiempos de medición son completamente ajustables, pudiendo establecerse como mínimo en 1 segundo y como máximo en un valor que tendría su límite en el tipo de variable usada para ello. En nuestro caso usamos una variable de 4 bytes con la que podemos contar hasta 2.147.483.647 segundos, lo cual permite establecer tiempos de registro de años.

El número de canales disponibles para el usuario, también es una de las grandes características que no se encuentran fácilmente en los registradores comerciales.

Nosotros ofrecemos 18 canales de entrada, donde podemos usar casi cualquier sensor previamente configurado, en oposición a los 8 canales disponibles en el registrador más caro de la tabla anterior.

Quizás una de las deficiencias de nuestro sistema registrador, sea la falta de un entorno de monitorización de los datos obtenidos. Estas tareas podrían haberse implementado mediante un web server, pero nuestra tarjeta Arduino no ofrecía los suficientes recursos de hardware para ello.

Los recursos de memoria de almacenamiento, se ven claramente suficientes para guardar miles de registros, gracias a una tarjeta de 8GB de capacidad. Esta por supuesto podría ser cambiada por otra de mayor capacidad si fuese necesario.

Si realizamos una comparación con el producto comercial más parecido en coste, obtenemos la siguiente tabla:

| ***Característica*** | ***N2011 comark diligence EV*** | ***Registrador Arduino*** |
|---|---|---|
| Sensor | Termistor NTC | Discreto Tmp36, Lm35 |
| Ciclo de medida | 1 segundo hasta 99 horas | 1 segundo hasta años |
| Rango de medida | -40C a +70C | Tmp36: -40C a +69C<br>Lm35: -55C a +110C |
| Precisión | ±0.1C | Tmp36: ±1C<br>Lm35: ±0.5C |
| Número de canales | 1 (interno) | 18 (externos) |
| Memoria | 16.000 registros | 8 GB |
| Software monitor | Si | No |
| Precio (Sin IVA) | 108.2€ | 116€ |

Tabla 8. Comparación de características entre nuestro registrador y uno comercial de coste similar

Como se observa, el proyecto de registrador de datos Arduino, ofrece una gran versatilidad debido a que dispone de una gran cantidad de puertos de medida, en este caso se trabaja con temperatura, pero se puede recoger multitud de magnitudes físicas.

La capacidad de adaptación a las necesidades del usuario también es otro de sus puntos fuertes, porque permite programar la aplicación como se desee, variando los tiempos de registro, los sensores a utilizar, la capacidad de almacenamiento, etc.

La utilización de la herramienta *GnuPlot* para realizar graficas, representa una alternativa sin grandes complicaciones, que puede mitigar el hecho de no disponer de una interfaz de monitorización en tiempo real.

Si precisamos de un sistema de registro amplio, con unas necesidades de precisión no demasiado exigentes y a un precio muy competitivo en comparación con las soluciones comerciales, el proyecto sobre Arduino cumple perfectamente estas necesidades.

# Anexo I

# Codigo fuente completo

## I. Fichero de cabecera *struct_header.h*

```
//definicion de los tipos basicos de Arduino
#include "Arduino.h"

//estructura de datos de cada puerto
typedef struct{
  char sensor[6];    //nombre de sensores
  byte mux;          //selecciona entrada analogica o mux (1 byte)
  byte puerto;       //puerto de lectura
  byte sample_hour;  //hora de lectura
  byte sample_min;   //minuto de lectura
  byte sample_sec;   //segundo de lectura
  long interval;    //siguiente medida (4 bytes)
  byte enable;       //activar/desactivar puerto
}datatemp;
```

## II. Fichero fuente *proyecto.ino*

```
#include <stdio.h>
#include <string.h>
#include <SD.h> //5mb
#include <Wire.h>//nada
#include <RTClib.h>//2mb
#include "struct_header.h"

///////VARIABLES GLOBALES////////

//definicion del tipo rtc
RTC_DS1307 RTC;

//variable para calculo de la muestra inicial
long init_sample[18];

//variable para led de señalizacion
long led;

//array con todos los puertos disponibles
//inicializo con los valores por defecto
//DEBE SER GLOBAL
```

```
//configuracion = {tipo_sensor , mux , puerto , hora , min , seg , interv
alo , enable}
datatemp ports[18] = { {"TMP36",0,0,12,30,0,20,1},
                       {"TMP36",0,1,12,30,0,20,1},
                       {"TMP36",1,0,12,30,0,20,0},
                       {"TMP36",1,1,12,30,0,20,0},
                       {"TMP36",1,2,12,30,0,20,0},
                       {"TMP36",1,3,12,30,0,20,0},
                       {"TMP36",1,4,12,30,0,20,0},
                       {"TMP36",1,5,12,30,0,20,0},
                       {"TMP36",1,6,12,30,0,20,0},
                       {"LM35",1,7,12,30,0,20,0},
                       {"LM35",2,0,12,30,0,20,0},
                       {"LM35",2,1,12,30,0,20,0},
                       {"LM35",2,2,12,30,0,20,0},
                       {"LM35",2,3,12,30,0,20,0},
                       {"LM35",2,4,12,30,0,20,0},
                       {"LM35",2,5,12,30,0,20,0},
                       {"LM35",2,6,12,30,0,20,0},
                       {"LM35",2,7,12,30,0,20,0} };

//definicion entradas de seleccion y salidas
//MUX1
#define mux1_sel0 2  //LSB
#define mux1_sel1 3
#define mux1_sel2 5  //MSB
#define mux1_out 2
//MUX2
#define mux2_sel0 6  //LSB
#define mux2_sel1 7
#define mux2_sel2 8  //MSB
#define mux2_out 3
//PIN_LED
#define pin_led 9

////////FUNCIONES DE CONVERSION////////
float convert_tmp36(int medida)
{
  return ((((float)medida * 1100) / 1024) - 500) / 10;
}

float convert_lm35(int medida)
{
  return(110 * (float)medida / 1024);
}

////////FUNCION SETUP////////
void setup()
{
  long n;
  int i;

  Serial.begin(9600);
  Wire.begin();
  RTC.begin();

  //definicion de los pines selmux1
  pinMode(mux1_sel0,OUTPUT);
  pinMode(mux1_sel1,OUTPUT);
  pinMode(mux1_sel2,OUTPUT);
  //definicion de los pines selmux2
```

```
  pinMode(mux2_sel0,OUTPUT);
  pinMode(mux2_sel1,OUTPUT);
  pinMode(mux2_sel2,OUTPUT);
  //definicion del pin led
  pinMode(pin_led,OUTPUT);

  //definicion de los pines analogicos
  pinMode(A0,INPUT);
  pinMode(A1,INPUT);
  pinMode(A2,INPUT);
  pinMode(A3,INPUT);

  //referencia de los pines analogicos establecida a 1.1V
  analogReference(INTERNAL);

  //comprobar RTC en el arranque
  if (!RTC.isrunning())
  {
    error(1);
  }
  /*Solo debe ejecutarse una vez el ajuste de hora
  else
  {
    RTC.adjust(DateTime(__DATE__, __TIME__));
  }*/

  //iniciar la tarjeta SD en el arranque
  if (!SD.begin(4))
  {
    error(2);
  }

   //escribir la conf. desde tarjeta SD
  if(SD.exists("config.txt"))
  {
    config_sd();
  }

  //calculo de la muestra inicial de cada puerto.
  //El momemto debe ser multiplo entero del intervalo para ignorar
retrasos
  DateTime now = RTC.now();
  for(i=0;i<18;i++)
  {
    //hora inicial en formato unixtime()
    DateTime
tini(now.year(),now.month(),now.day(),ports[i].sample_hour,ports[i].sampl
e_min,ports[i].sample_sec);
    // n = (Tactual + intervalo - Tinit) / intervalo
    n = (now.unixtime() + ports[i].interval - tini.unixtime()) /
ports[i].interval;
    // Tsample = Tinit + n * intervalo
    init_sample[i] = tini.unixtime() + n * ports[i].interval;
  }

  //inicializacion del encendido del led cada 3 segundos
  led = now.unixtime() + 3;

}
```

```
////////FUNCION LOOP////////
void loop()
{
  //puntero al array de estructuras
  datatemp *pactual;

  //variables intermedias
  byte indice;
  int analog;
  float grados;

  //objeto de acceso al rtc
  DateTime now = RTC.now();

  //comprobar en tiempo real errores del RTC
  if (!RTC.isrunning())
  {
    error(1);
  }

  //recorrer todas las estructuras
  for(indice=0 ; indice<18 ; indice++)
  {
    //apunta consecutivamente a las estructuras
    pactual = &ports[indice];

    //si enable=1 el puerto esta operativo
    if(pactual->enable == 1)
    {
      if(init_sample[indice] <= now.unixtime())
      {
            //reconocer el puerto, su tipo y leerlo
            analog = leer_puerto(pactual);

            //convertir la lectura a grados
            grados = convert_lectura(pactual,analog);

            //escribir lecturas en tarjeta sd
            write_sd(pactual,now,grados);

            //calcular la proxima lectura
            prox_muestra(pactual,now,indice);
      }
    }
  }

  //intermitencia normal del led
  if(led <= now.unixtime())
  {
    if(digitalRead(pin_led))
    {
       digitalWrite(pin_led,LOW);
       led = now.unixtime() + 3;
    }
    else
    {
       digitalWrite(pin_led,HIGH);
       led = now.unixtime() + 1;
    }
  }
}
```

```
////////FUNCION DE LECTURA PRECISA////////
int lectura_precisa(byte mux,byte port)
{
  unsigned int tempread = 0;
  unsigned int intmp;
  byte i;

  if(mux == 1)
  {
    port = mux1_out;
  }
  else if(mux == 2)
  {
    port = mux2_out;
  }

  //Descartar la primera medida por fluctuaciones
  intmp=analogRead(port);

  for (i=0;i<=3;i++)
  {
    intmp=analogRead(port),
    // 4 lecturas para obtener mayor estabilidad
    tempread = tempread + intmp;
    delay(15);
  }
  tempread = tempread>>2;
  return(tempread);
}

////////FUNCION LEER PUERTO////////
int leer_puerto(datatemp *pactual)
{
  byte i;
  byte puerto_mux;
  byte selmuxbit[3];

  //ver si es puerto analogico o puerto mux
  if(pactual->mux == 0)
  {
    //puerto analogico, solo el 0 y 1
    if(pactual->puerto == 0 || pactual->puerto == 1)
    {
      //salida del conversor ADC de 10 bits
      return(lectura_precisa(pactual->mux,pactual->puerto));
    }
    else
    {
      error(3);
    }
  }
  else if(pactual->mux == 1 || pactual->mux == 2)
  {
    if(pactual->puerto <= 7 && pactual->puerto >= 0)
    {
      //puerto desde multiplexor 1 o 2. Leer bits seleccion del mux
      puerto_mux = pactual->puerto;

      // 3 iteraciones para 3 bits
      for(i=0;i<3;i++)
      {
```

```
        //convierto decimal a binario
        selmuxbit[i] = puerto_mux % 2;
        puerto_mux = puerto_mux / 2;
      }

      //mando los bits seleccion al multiplexor 1
      if(pactual->mux == 1)
      {
        digitalWrite(mux1_sel0,selmuxbit[0]);
        digitalWrite(mux1_sel1,selmuxbit[1]);
        digitalWrite(mux1_sel2,selmuxbit[2]);
        return(lectura_precisa(pactual->mux,pactual->puerto));
      }
      //mando los bits seleccion al multiplexor 2
      if(pactual->mux == 2)
      {
        digitalWrite(mux2_sel0,selmuxbit[0]);
        digitalWrite(mux2_sel1,selmuxbit[1]);
        digitalWrite(mux2_sel2,selmuxbit[2]);
        return(lectura_precisa(pactual->mux,pactual->puerto));
      }
    }
    else
    {
      error(4);
    }
  }
  else
  {
    error(5);
  }
}

////////FUNCION CONVERTIR MEDIDA////////
float convert_lectura(datatemp *pactual,int medida)
{
  if(strcmp(pactual->sensor,"TMP36") == 0)
  {
    //llamar a funcion para tmp36
    return(convert_tmp36(medida));
  }
  else if(strcmp(pactual->sensor,"LM35") == 0)
  {
    //llamar a funcion para lm35
    return(convert_lm35(medida));
  }
  else
  {
    error(6);
  }
}

////////FUNCION PROXIMA MEDIDA////////
void prox_muestra(datatemp *pactual,DateTime now,int i)
{
  //actualizar la hora de medida
  init_sample[i] = now.unixtime() + pactual->interval;
}

////////ESCRIBIR TARJETA SD////////
void write_sd(datatemp *pactual,DateTime now,float grados)
```

```
{
  File fp;
  char filename[13];
  char ext[5] = ".txt";
  char buf[5];

  //nombre de archivo para cada puerto activo
  //longitud maxima 8.3 caracteres
  sprintf(filename,"%s-%d%d%s",pactual->sensor,pactual->mux,pactual-
>puerto,ext);

  fp = SD.open(filename,FILE_WRITE);
  if(fp)
  {
    if(fp.print(now.year()) < 1)
    {
      error(7);
    }
    fp.print('/');
    centenas(now.month(),fp);
    fp.print(now.month());
    fp.print('/');
    centenas(now.day(),fp);
    fp.print(now.day());
    fp.print(' ');

    centenas(now.hour(),fp);
    fp.print(now.hour());
    fp.print(':');
    centenas(now.minute(),fp);
    fp.print(now.minute());
    fp.print(':');
    centenas(now.second(),fp);
    fp.print(now.second());
    fp.print(' ');

    dtostrf(grados,4,2,buf);
    fp.print(buf);
    fp.print(' ');
    fp.println('C');

    //es necesario cerrar fichero para guardar datos
    fp.close();
  }
  else
  {
    error(7);
  }

}

////////FUNCION PINTAR CENTENAS////////
void centenas(uint8_t var,File fp)
{
  if(var < 10)
  {
    fp.print('0');
  }
}

////////LEER CONFIGURACION SD////////
```

```
void config_sd()
{
  File fp;
  datatemp *pt;
  byte i,j;
  char c;

  char buffer[35];
  char delim[4]="{,}";
  char *res;

  //abrir fichero de configuracion
  fp = SD.open("config.txt",FILE_READ);
  if(fp)
  {
    for(i=0 ; i<18 ; i++)
    {
      //recorro todas las estructuras
      pt = &ports[i];

      //recogo cada linea del fichero
      for(j=0 ; j<35 ; j++)
      {
        c = fp.read();
        //compruebo errores en la lectura
        if(c == -1)
        {
          error(8);
        }

        if(c == '\n' || c == EOF)
        {
          buffer[j] = '\0';
          break;
        }
        else
        {
          buffer[j] = c;
        }
      }

      //parsear los datos en buffer y guardarlos en estructura
      res = strtok(buffer,delim);
      if(res == NULL){error(9);} else{strcpy(pt->sensor,res);}
      //mux
      res = strtok(NULL,delim);
      if(res == NULL){error(9);} else{pt->mux = atoi(res);}
      //puerto
      res = strtok(NULL,delim);
      if(res == NULL){error(9);} else{pt->puerto = atoi(res);}
      //sample_hour
      res = strtok(NULL,delim);
      if(res == NULL){error(9);} else{pt->sample_hour = atoi(res);}
      //sample_min
      res = strtok(NULL,delim);
      if(res == NULL){error(9);} else{pt->sample_min = atoi(res);}
      //sample_sec
      res = strtok(NULL,delim);
      if(res == NULL){error(9);} else{pt->sample_sec = atoi(res);}
      //intervalo
      res = strtok(NULL,delim);
```

```
      if(res == NULL){error(9);} else{pt->interval = atoi(res);}
      //enable
      res = strtok(NULL,delim);
      if(res == NULL){error(9);} else{pt->enable = atoi(res);}
    }
    fp.close();
  }
  else
  {
    error(8);
  }
}

////////FUNCION CODIGO ERROR////////
void error(byte err)
{
  while(1)
  {
    Serial.print("Codigo_error: ");
    Serial.println(err);

    //parpadeo intenso indica error
    digitalWrite(pin_led,HIGH);
    delay(100);
    digitalWrite(pin_led,LOW);
    delay(100);
  }
}
```

# Bibliografía

## *Capitulo 1*

[1] Arduino Website. (2013, Julio 22). Página principal. Recuperado el 1 de Agosto de 2013 desde URL: http://www.arduino.cc/

## *Capitulo 2*

[2] Single-Board Computer. (2013, Agosto 1). Recuperado el 5 de Agosto de 2013 desde URL: http://en.wikipedia.org/wiki/Single-board_computer

[3] Single-Board Microcontroller. (2013, Mayo 20). Recuperado el 5 de Agosto de 2013 desde URL: http://en.wikipedia.org/wiki/Single-board_microcontroller

[4] Atmel AVR. (2013, Julio 28). Recuperado el 7 de Agosto de 2013 desde URL: http://en.wikipedia.org/wiki/Atmel_AVR

[5] Atmel Corporation. (2013). 8051 Architecture Microcontroller. Overview. Recuperado el 8 de Agosto de 2013 desde URL: http://www.atmel.com/products/microcontrollers/8051architecture/default.aspx

[6] Arduino Website. (2012, Octubre 19). Getting Started. Introduction. Recuperado el 2 de Agosto de 2013 desde URL: http://www.arduino.cc/en/Guide/Introduction

[7] Arduino Website. (2013, Julio 11). Products. Arduino UNO R3. Arduino MEGA 2560. Ethernet Shield. Recuperado el 2 de Agosto de 2013 desde URL: http://arduino.cc/en/Main/Products

[8] Cooper, Tyler. (2013, Enero 29). DS1307 Real Time Clock Breakout Board Kit. Recuperado el 6 de Agosto de 2013 desde URL: http://learn.adafruit.com/ds1307-real-time-clock-breakout-board-kit/overview

[9] Modtronix Engineering. (2007, Mayo 16). Product USB SBC44UC. Recuperado el 7 de Agosto de 2013 desde URL: http://www.modtronix.com/product_info.php?products_id=286

[10] Modtronix Engineering. (2005, Enero 16). Product Ethernet Board SBC65EC. Recuperado el 7 de Agosto desde URL: http://www.modtronix.com/product_info.php?cPath=1_36&products_id=149

[11] Raspberry Pi Website. (2012). Frecuently Asked Questions. Recuperado el 9 de Agosto de 2013 desde URL: http://www.raspberrypi.org/faqs

[12] Broadcom. (2011, Septiembre 9). High Definition 1080p embedded multimedia applications processor. Recuperado el 9 de Agosto de 2013 desde URL: http://www.broadcom.com/products/BCM2835.

[13] Arquitectura ARM. (2013, Agosto 6). Recuperado el 9 de Agosto de 2013 desde URL: http://es.wikipedia.org/wiki/Arquitectura_ARM

[14] Farnell Element14 Comunity. (2013, Mayo 3). Raspberry Pi Model A versus Model B chart. Recuperado el 9 de Agosto de 2013 desde URL: http://www.element14.com/community/docs/DOC-51668?ICID=raspi-group

## *Capitulo 3*

[15] Arduino Website. (2013, Abril 29). Products. Arduino Ethernet. Recuperado el 3 de Agosto de 2013 desde URL: http://arduino.cc/en/Main/ArduinoBoardEthernet

[16] Adafruit Industries. Product ID:70. USB FTDI TTL-232 cable - TTL-232R 3.3V. Recuperado el 8 de Agosto de 2013 desde URL: http://www.adafruit.com/products/70

[17] Analog Devices. (2010, Noviembre). Documentation. TMP35/TMP36/TMP37: Low Voltage Temperature Sensors Data Sheet. Recuperado el 10 de Agosto de 2013 desde URL: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf

[18] Ladyada Website. (2013, Julio 30). TMP36 Temperature Sensor. Recuperado el 1 de Agosto de 2013 desde URL: http://learn.adafruit.com/tmp36-temperature-sensor

[19] Texas Instruments. (2013, Julio 5). LM35 Precision Centigrade Temperature Sensors (Rev. C). Recuperado el 2 de Agosto de 2013 desde URL: http://www.ti.com/lit/ds/symlink/lm35.pdf

[20] Arduino Website. (2012, Diciembre 4). Reference. Structure. Recuperado el 20 de Julio de 2013 desde URL: http://arduino.cc/en/Reference/HomePage

[21] ON Semiconductor. (2013, Febrero). MC14051B: 8-Channel Analog Multiplexer/Demultiplexer (Mux/Demux). Recuperado el 2 de Agosto de 2013 desde URL: http://www.onsemi.com/PowerSolutions/product.do?id=MC14051B

[22] Arduino Website. (2012, Diciembre 4). Reference. Functions Digital I/O. Recuperado el 20 de Julio de 2013 desde URL: http://arduino.cc/en/Reference/HomePage

[23] Maxim Integrated. (2011, Febrero 15). DS1307. 64 x 8, Serial, $I^2C$ Real-Time Clock. Recuperado el 3 de Agosto de 2013 desde URL: http://www.maximintegrated.com/datasheet/index.mvp/id/2688/t/al

[24] Kingston Tecnology. (2013). Tarjetas Flash. Dispositivos móviles MicroSD. MicroSDHC Clase 4. Recuperado el 4 de Agosto de 2013 desde URL: http://www.kingston.com/es/flash/microsd_cards#sdc4 http://www.kingston.com/datasheets/sdc4_es.pdf

[25] Arduino Website. (2012, Enero 16). Learning. Libraries SD. Example Read/Write. Recuperado el 4 de Agosto de 2013 desde URL: http://arduino.cc/en/Tutorial/ReadWrite

## *Capitulo 4*

[26] Arduino Playground. Installing Arduino on Linux. Recuperado el 10 de abril de 2013 desde URL: http://playground.arduino.cc/Learning/Linux

[27] Arduino Website. (2013, Marzo 18). Download the Arduino Software. Recuperado el 10 de abril de 2013 desde URL: http://arduino.cc/en/Main/Software

[28] Oracle. Java platform Standard Edition (SE). Download Java SE for developers. Java SE Runtime Environment 7u17. Recuperado el 10 de abril de 2013 desde URL: http://www.oracle.com/technetwork/java/javase/downloads/index.html

[29] Juanetebitel. (2012, Abril). Instalar Oracle Java 7 en Ubuntu 12.04.Forma manual: instalación del JRE para usuarios. Recuperado el 10 de Abril de 2013 desde URL: http://www.ubuntu-guia.com/2012/04/instalar-oracle-java-7-en-ubuntu-1204.html

[30] PaintYourDragon. (2013, Julio 5). Adafruit/RTClib. A fork of Jeelab's fantastic RTC library. Recuperado el 10 de Julio de 2013 desde URL: https://github.com/adafruit/RTClib

[31] Arduino Website. (2013, Mayo 12). Getting Started. Libraries. Installing Additional Arduino libraries. Recuperado el 11 de Julio de 2013 desde URL: http://arduino.cc/en/Guide/Libraries

[32] *Referente al sub-capitulo 4.3 completo*.
Banzi, Massimo. (2011, Septiembre). Getting Started with Arduino 2nd Edition. Edit O´Reilly Books.
Margolis, Michael. (2011, Diciembre). Arduino CookBook 2nd edition. Edit O´Reilly Books.

[33] Nisley, Ed. (2009, Octubre 6). The Smell of Molten Projects in the Morning. Arduino: Be careful with the preprocessor. Recuperado el 10 de Julio de 2013 desde URL: http://softsolder.com/2009/10/06/arduino-be-careful-with-the-preprocessor/

[34] Accurate LM35 readings. (2011, Septiembre 2). Recuperado el 11 de Julio de 2013 desde URL: http://www.ogalik.ee/accurate-lm35-reading/

[35] Arduino Website. (2010, Junio 6). Reference. Ethernet. Example Web Server. Recuperado el 15 de Julio de 2013 desde URL: http://arduino.cc/en/Tutorial/WebServer

[36] Starting Electronics Website. (2013, Enero 21). Arduino Ethernet Shield web server tutorial. Recuperado el 16 de Julio de 2013 desde URL: http://startingelectronics.com/tutorials/arduino/ethernet-shield-web-server-tutorial/basic-web-server/

[37] Webduino project hosting. (2012, Enero). Simple and extensible web server for Arduino and Ethernet Shield. Recuperado el 25 de Julio de 2013 desde URL: http://code.google.com/p/webduino/

[38] Margolis, Michael. (2011, Diciembre). Arduino CookBook 2nd edition. Edit O´Reilly Books. Chapter 17-Advanced coding and memory handling.

[39] Arduino Website. (2010, Junio 2). Reference. AVR LibC User manual. Recuperado el 15 de Agosto de 2013 desde URL: http://arduino.cc/en/Reference/PROGMEM http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html

[40] Arduino Website. (2010, Junio 2). Reference. Libraries Page EEprom. Recuperado el 15 de Agosto de 2013 desde URL: http://arduino.cc/en/Reference/EEPROM

## *Capitulo 5*

[41] GnuPlot Homepage. (2013, Abril). Download. Download Site on SourceForge. Versión 4.6.3. Recuperado el 20 de Agosto de 2013 desde URL: http://www.gnuplot.info/download.html

[42] GnuPlot Homepage. (2012). Current release 4.6. User Manual Pdf. Recuperado el 20 de Agosto de 2013 desde URL: http://www.gnuplot.info/index.html

[43] Cadsoftusa. Downloads. Windows versión 6.5.0. Recuperado el 22 de Agosto de 2013 desde URL: http://www.cadsoftusa.com/

[44] Cadsoftusa. Downloads. Freeware. Get the Pcb design software with the EAGLE Freeware version. Recuperado el 20 de Agosto de 2013 desde URL: http://www.cadsoftusa.com/download-eagle/freeware/?language=en

[45] Ladyada Website. (2011, Mayo 17). Eagle Library. Ladyada/Adafruit's own Eagle CAD library. Recuperado el 21 de Agosto de 2013 desde URL: http://www.ladyada.net/library/pcb/eaglelibrary.html

[46] Sparkfun electronics. (2013, Agosto 15). Articles. EAGLE library. Recuperado el 21 de Agosto de 2013 desde URL: https://www.sparkfun.com/pages/eagle_lib_lightbox

[47] gaussmarkov. (2006, Abril 16). Eagle: Adding libraries. Recuperado el 21 de Agosto de 2013 desde URL: http://gaussmarkov.net/eagle/addinglibs.html

[48] Coaguila Mayanaza, Dante. (2012, Junio 23). EAGLE 6.2 Tutorial paso a paso. Capitulo 1, esquemático. Recuperado el 22 de Agosto de 2013 desde URL: http://www.youtube.com/watch?v=1oRQShk9S2U

[49] Coaguila Mayanaza, Dante. (2012, Junio 24). EAGLE 6.2 Tutorial paso a paso. Capitulo 2, pcb. Recuperado el 23 de Agosto de 2013 desde URL: http://www.youtube.com/watch?v=Kjgqt3Vi-mo

## *Capitulo 6*

[50] RS España. (2013). Componentes electrónicos. Recuperado el 22 de Agosto de 2013 desde URL: http://es.rs-online.com/web/