



**UNIVERSIDAD DE JAÉN**  
Escuela Politécnica Superior de Linares

**Trabajo Fin de Grado**

**DISEÑO DE UN REGISTRADOR DE  
DATOS DE BAJO COSTE  
AUTÓNOMO PARA  
MONITORIZACIÓN DE SISTEMAS  
FOTOVOLTAICOS HÍBRIDOS  
BASADO EN PLATAFORMAS DE  
HARDWARE Y SOFTWARE LIBRE.**

**Alumno:** Vicente Crespo Rico

**Tutores:** Manuel Fuentes Conde/Marta Vivar García  
**Dpto.:** Ingeniería Electrónica y Automática

**Septiembre, 2016**



**UNIVERSIDAD DE JAÉN**  
Escuela Politécnica Superior de Linares

**Trabajo Fin de Grado**

**DISEÑO DE UN REGISTRADOR DE  
DATOS DE BAJO COSTE  
AUTÓNOMO PARA  
MONITORIZACIÓN DE SISTEMAS  
FOTOVOLTAICOS HÍBRIDOS  
BASADO EN PLATAFORMAS DE  
HARDWARE Y SOFTWARE LIBRE.**

**Alumno: Vicente Crespo Rico**

**Tutores: Manuel Fuentes Conde/Marta Vivar García**

**Dpto.: Ingeniería Electrónica y Automática**

A handwritten signature in black ink that reads "Vicente Crespo". The signature is slanted and appears to be a digital scan of a physical signature.

**Fdo. Alumno**

A handwritten signature in blue ink that includes the name "Marta" and a large, stylized, crisscrossed "X" mark, likely indicating a rejection or cancellation.

**Fdo. Tutores**

**Septiembre, 2016**

# ÍNDICE GENERAL

1	ÍNDICE DE FIGURAS .....	3
2	ÍNDICE DE TABLAS.....	5
3	RESUMEN.....	6
4	INTRODUCCIÓN.....	7
5	ANTECEDENTES .....	9
5.1	<b>Registrador de datos .....</b>	9
5.1.1	Plataformas comerciales de hardware libre .....	9
5.1.1.1	Arduino UNO.....	10
5.1.1.2	Raspberry Pi.....	12
5.1.1.3	Nanode .....	13
5.1.1.4	Libelium Wapsmote .....	14
5.1.2	Registros de datos comerciales .....	15
5.1.2.1	Termohigrómetro PCE-THB 40.....	15
5.1.2.2	Logger de datos de temperatura y humedad Log110-Start.....	16
5.1.2.3	Sistema de telecontrol GSM/GPRS Hermes LC-2 .....	17
5.1.2.4	Estación meteorológica PCE-FWS20 .....	19
5.1.3	Datalogger del trabajo anterior .....	21
5.2	<b>Funcionamiento del registrador.....</b>	22
5.3	<b>Soluciones usando microcontroladores Atmel .....</b>	24
5.3.1	Hardware .....	24
5.3.1.1	Tipos de microcontroladores .....	25
5.3.1.2	Placas de bajo consumo.....	25
5.3.2	Software.....	28
5.3.2.1	Reducción de la frecuencia de oscilación.....	29
5.3.2.2	Modos sleep microcontroladores ATMega AVR-8 bits .....	31
5.3.2.3	Wake up mediante timers e interrupciones programadas y hardware .....	32
5.3.2.4	Wake up mediante watchdog .....	35
5.3.2.5	Uso de librerías de bajo consumo.....	36
5.4	<b>Comunicación .....</b>	37
5.4.1	Comunicación con hilos .....	37
5.4.1.1	Comunicación Serie (UART).....	37
5.4.1.2	Comunicación Ethernet .....	38
5.4.2	Comunicación sin hilos .....	40
5.4.2.1	Bluetooth.....	40
5.4.2.2	WiFi .....	42
5.4.2.3	RF.....	42
5.4.2.4	GPRS.....	43
6	OBJETIVOS.....	45
7	DESCRIPCIÓN DE LAS SOLUCIONES ADOPTADAS .....	46
7.1	<b>Equipamiento de medida, almacenamiento y envío de datos.....</b>	46

7.1.1	Construcción del registrador de datos.....	47
7.1.2	Puesta en hora del reloj.....	50
<b>7.2 Análisis de soluciones: Hardware</b>	.....	<b>50</b>
7.2.1	Sustitución regulador de voltaje .....	51
7.2.2	Diseño de bareboard .....	54
7.2.3	Implantación módulo GPRS .....	57
7.2.4	Diseño de SFA (Sistemas Fotovoltaico Autónomo) .....	58
7.2.4.1	Cálculo de la instalación (Dimensionamiento).....	60
<b>7.3 Análisis de soluciones: Software</b>	.....	<b>63</b>
7.3.1	Soluciones no contempladas.....	64
7.3.2	Software del registrador de datos.....	64
7.3.4	Creación de aplicación de servidor para almacenamiento y procesado de datos y sincronización del reloj RTC .....	65
7.3.4.1	Aplicación de procesado de datos .....	66
7.3.4.2	Aplicación para sincronización del reloj RTC .....	68
7.3.5	Presentación de los datos enviados.....	68
8	JUEGO DE PRUEBAS .....	71
<b>8.1 Necesidad de toma de medidas</b>	.....	<b>71</b>
<b>8.2 Circuitos de medida</b>	.....	<b>72</b>
<b>8.2 Registrador de datos con librería Jeelib.</b>	.....	<b>73</b>
9	DISCUSIÓN DE RESULTADOS.....	78
10	CONCLUSIONES.....	81
11	LÍNEAS FUTURAS.....	83
12	BIBLIOGRAFÍA.....	84
13	PRESUPUESTOS .....	87
14	ANEXOS.....	90
<b>14.1 Manual de usuario</b>	.....	<b>90</b>
<b>14.2 Uso de bareboard</b>	.....	<b>92</b>
<b>14.3 Desarrollo de hardware</b>	.....	<b>97</b>
<b>14.4 Desarrollo de software</b>	.....	<b>99</b>
14.4.1	Diagrama de flujo del software desarrollado.....	99
14.4.2	Programa general .....	100
14.4.3	Toma de corrientes .....	114
14.4.4	Aplicación web PHP para procesado de datos .....	115
14.4.5	Aplicación web PHP para sincronización hora reloj RTC .....	117
14.4.6	Aplicación web PHP para comprobar si el fichero de datos existe .....	117

# 1 ÍNDICE DE FIGURAS

1. Resistencias pull-up y pull-down .....	11
2. Pinout placa Arduino UNO R3 .....	12
3. Raspberry Pi .....	13
4. Nanode .....	14
5. Libelium Wasp mote .....	15
6. Termohigrómetro PCE-THB 40 .....	16
7. Logger Log110-Start .....	17
8. Sistema de telecontrol GSM/GPRS Hermes LC-2 .....	18
9. Estación meteorológica PCE-FWS20 .....	20
10. Datallogger secundario .....	21
11. Prototipo final datalogger secundario .....	22
12. Diagrama de flujo datalogger secundario .....	24
13. Arduino Pro Mini .....	26
14. Arduino Lilypad .....	26
15. Arduino Moteino .....	27
16. Arduino Mosquino .....	28
17. Boards.txt .....	29
18. Boards.txt 2 .....	30
19. Registros TCCR 1A y 1B .....	33
20. Configuración watchdog .....	35
21. Comunicación serie .....	38
22. Shield Ethernet .....	39
23. Bluetooth HC-05 .....	41
24. WiFi ESP8266 .....	42
25. RF NRF2401s .....	43
26. GSM/GPRS SIM800L .....	44
27. Prototipo final .....	49
28. Regulador AMS1117 .....	51
29. Regulador AMS1117 en placa Arduino UNO R3 .....	52
30. Regulador AMS1117 retirado de placa Arduino UNO R3 .....	52
31. Fuente conmutada .....	53
32. Regulador DC-DC LM2596 .....	53
33. ATMega1284p montado con mínimos componentes en protoboard .....	55
34. Programador FTDI232 .....	56
35. ATMega1284p en placa de circuito impreso con mínimos componentes .....	56

36. SFA autónomo AC/DC .....	59
37. Diagrama de flujo software general .....	65
38. Diagrama de flujo software PHP recogida de datos .....	67
39. OpenEnergyMonitor .....	69
40. Dashboard con gráficas de sensores .....	70
41. Sensor de corriente INA219B .....	71
42. Circuito toma de corrientes .....	72
43. Gráfico de corriente proceso completo de medida .....	75
44. Gráfico de corriente TFG anterior .....	77
45. Medidas recogidas en el servidor .....	78
46. Página web TFG .....	79
47. Pines utilizados ATMega328p TFG anterior .....	93
48. Montaje ATMega168 en protoboard con mínimos componentes y reloj externo .....	93
49. Montaje ATMega168 en protoboard con mínimos componentes sin reloj externo .....	94
50. Programación de placa arduino como programador SPI .....	95
51. Compatibilidad con IDE ATMega328p usando reloj interno .....	95
52. Selección de placa para escribir bootloader .....	96
53. Selección programador para grabar bootloader .....	97
54. Proceso de grabación bootloader .....	97
55. Esquema prototipo final Proteus .....	98
56. Esquema prototipo final Fritzing .....	99
57. Diagrama de flujo del software general .....	100

## 2 ÍNDICE DE TABLAS

1. Tabla comparativa de microcontroladores Atmega .....	25
2. Escalado dinámico CLKPR .....	30
3. Modos sleep Atmega .....	31
4. Medias de las medidas de corriente .....	75
5. Medidas de corriente registrador primario TFG anterior .....	76

### 3 RESUMEN

Este trabajo de fin de grado tiene como objetivo el desarrollo e implementación de un registrador de datos autónomo de bajo coste tanto energético como económico, para sistemas fotovoltaicos híbridos utilizando hardware y software libre.

Dicho dispositivo permite la monitorización de los parámetros más importantes de un sistema fotovoltaico, aunque el que aquí se va a tratar, se centra más en los ubicados en sitios donde no es accesible la conexión a la red eléctrica, por lo que debe buscar otras formas de energía, como baterías o módulos fotovoltaicos.

Este trabajo, toma en consideración las deficiencias energéticas en los países subdesarrollados para los cuales se centra el diseño de dicho registrador. Por lo cual, el dispositivo debe tener como pauta a seguir, un consumo mínimo, tanto a nivel en el desarrollo hardware como optimización software.

Para llevar a cabo dicha tarea, se deberá realizar un estudio en profundidad, para poder elegir la mejor solución en cuanto a optimización en el consumo y reducción de costes en el diseño y montaje.

Por último, resaltar que este trabajo continúa con las líneas de futuro propuestas en el anterior trabajo “optimización de consumo de un registrador de datos de bajo coste destinado a monitorización para aplicaciones en países en desarrollo” [1], presentado el pasado año en la E.P.S Linares. Continúa cumpliendo las expectativas de precisión y robustez, y sigue con la demanda de ser una base para posibles trabajos de investigación.

## 4 INTRODUCCIÓN

El siguiente proyecto trata como línea principal el diseño de un registrador de datos de bajo coste autónomo, para monitorizar los diferentes parámetros presentes en una instalación solar fotovoltaica.

El punto crítico en este diseño, ya que se pretende usar en países subdesarrollados en los cuales no se dispone de red eléctrica en la mayoría de los casos, es tratar de optimizar el consumo energético al máximo posible y abaratar los costes de fabricación adyacentes, por tanto, buscamos la mayor autonomía con los dispositivos a usar en dicho diseño.

Otro aspecto a tener en cuenta es que el registrador estará ubicado en un emplazamiento que se encontrará muy alejado de cualquier núcleo urbano, lo cual nos obligaría a buscar una solución para, además de almacenar los datos regularmente en el dispositivo, poder disponer de ellos en tiempo real y desde cualquier parte para poder monitorizar el sistema. Aun con estas metas impuestas como prioritarias en nuestro trabajo, no podemos dejar de lado las características de fiabilidad, precisión y robustez de los cuales debemos dotar a nuestro registrador para obtener unas medidas de calidad.

En el mercado podemos encontrar un gran abanico de opciones para elegir qué tipo de registrador queremos usar en función de nuestras necesidades y, como no, el desembolso que el cliente quiera realizar en función de la calidad de las mediciones que le sean necesarias. Pero estos puntos tienen como factor negativo un alto incremento del presupuesto final. Después de una búsqueda exhaustiva teniendo en cuenta el bajo coste, la máxima autonomía, los diferentes factores ambientales necesarios para medir, almacenamiento de datos y transmisión de los datos, se han podido encontrar diferentes soluciones comerciales, de las cuales se van a destacar algunos ejemplos.

- Termohigrómetro PCE-THB 40, dispositivo portátil capaz de detectar temperatura y humedad ambiental, así como presión barométrica guardando datos en una SD y alimentado mediante pilas desde un precio de 191,18 €
- Log110-Start, dispositivo portátil con detección de temperatura y humedad relativa y alimentado mediante pilas desde un precio de 228,69 €

- Sistema de telecontrol GSM/GPRS HERMES LC-2 alimentado desde corriente alterna con baterías tipo LiPo para funcionar unas horas en caso de corte eléctrico, con capacidad para conexión de hasta 8 sensores digitales, conexión de sondas de temperatura y humedad relativa, 2 salidas para acción de relés, almacenamiento de datos, generación de alarmas vía GSM y telecontrol mediante llamadas GPRS y almacenamiento de datos desde un precio de 336,86 €
- Estación meteorológica PCE-FWS20 con pantalla táctil, mástil y 5 sensores de temperatura, humedad, dirección y velocidad del viento, pluviómetro y presión atmosférica. Todos estos datos son mandados al receptor con pantalla táctil, localizado en algún lugar de interior. El transmisor posee una alimentación de 2 pilas AAA recargables junto con una mini celda solar y 3 para el receptor además de la necesidad de alimentación mediante puerto USB, pudiendo llegar a una autonomía de hasta 1 año. La conexión USB del receptor se utiliza para el volcado de los datos a un pc para analizarlos mediante el software facilitado. Esta estación meteorológica, puede encontrarse desde un precio de 104,1 €

En los ejemplos expuestos, podemos ver que no cumplen todas las premisas. Los dos primeros quedan descartados por ser dispositivos portátiles. El tercero no dispone de sensores, solo las entradas para poder agregarlos y la necesidad de conexión a la red eléctrica. El último ejemplo, es uno de los más completos para dar solución a las necesidades de este trabajo. Sin embargo, el hecho de no disponer en primera instancia de algún método de almacenamiento, tener un reducido alcance en la transmisión de los datos y contar con la dependencia de que el receptor no sea para exteriores además de necesitar un PC para realizar el volcado de los datos, lo hacen inviable.

Todo esto lleva a la necesidad de realizar métodos que permitan al registrador tener una menor dependencia energética, ya que ninguna de las soluciones comerciales más completas dispone de opciones de bajo consumo. Además, se hace necesario disponer de sistemas de transmisión de largo alcance, ya que el dispositivo estará ubicado en zonas alejadas de núcleos urbanos, lo que imposibilita el uso de comunicaciones con cables o inalámbricas de poco alcance. A todo lo comentado, debe agregarse que no cumplen el objetivo de bajo coste lo que aleja a estos registradores comerciales de los objetivos de este trabajo.

## 5 ANTECEDENTES

Antes de pasar a exponer la solución adoptada que permite cumplir las expectativas de diseño, se propone un análisis entorno a la problemática expuesta en la introducción. Para ello se presenta un conjunto de apartados para aclarar el tema a tratar.

- Registrador de datos
- Funcionamiento del registrador
- Tipos de optimización
  - Hardware
  - Software
- Tipos de comunicación
  - Con hilos
    - Puerto Serie (UART)
    - Ethernet
  - Sin hilos
    - Bluetooth
    - WiFi
    - RF (Radiofrecuencia)
    - GPRS

### 5.1 Registrador de datos

Para una mejor comprensión de las tareas a realizar, es necesario el estudio del trabajo de partida [1], y para ello se realiza un resumen del mismo. El anterior registrador de datos está compuesto por un conjunto de placas que realizan las tareas necesarias para su cometido y las cuales se van a enumerar y clasificar.

#### 5.1.1 *Plataformas comerciales de hardware libre*

Las plataformas comerciales de hardware libre, permiten el desarrollo de la aplicación necesaria para solventar los problemas encontrados a este trabajo, por lo que se hace necesario de su mención y estudio. A continuación, se muestran las más conocidas en el mercado y por ser las que más se ajustan a la resolución de dichos problemas para el desarrollo del registrador de datos:

- Arduino
- Raspberry Pi
- Nanode
- Wasp mote

#### 5.1.1.1 *Arduino UNO*

Por ser la más extendida, se explica el uso de la placa Arduino UNO R3 [2]. Posee el microcontrolador ATMega328p-pu, el cual tiene un voltaje de funcionamiento de entre 1.8-5.5 V, aunque el regulador lineal integrado, marca un voltaje fijo de 5 V. Sin embargo, la alimentación recomendada oscila entre 7-12 V y el límite estaría en 20V. A continuación, se describen las memorias incorporadas:

- Memoria de 32 KB ISP flash para almacenar el código de programa, la cual lee mientras escribe, aunque 0.5 KB están destinados al bootloader (gestor de arranque), el cual se encarga de comprobar si hay disponibles datos para la comunicación USB-serie para poder realizar la carga del software al microcontrolador y en caso de no haberla, daría paso a la ejecución del programa principal.
- Una memoria SRAM de 2 KB, una EEPROM de 1 KB utilizada para leer y escribir, pero con un número bastante limitado de escrituras, poco utilizada, pero a veces necesaria para almacenar datos que queremos conservar aun sin alimentar el microcontrolador.

El microcontrolador, tiene 10 bits de resolución para tomar medidas, pero puede aumentarse la resolución de medidas regulando el voltaje de entrada mediante el pin AREF, el cual puede medir una referencia de voltaje fijo no oscilante, comprendida entre 1-5 V, y mediante software se pueden ajustar valores de referencia predeterminados, como, por ejemplo, “analogReference(DEFAULT);”, el cual activa la referencia por defecto de 5 V, “analogReference(INTERNAL1V1);”, que activa una referencia interna del propio microcontrolador y la “analogReference(EXTERNAL);”, que permite fijar una referencia de voltaje entre los dos rangos límite comentados, siempre y cuando no fluctúen, sino podríamos destruir los pines analógicos. Esta placa también dispone de un botón de reinicio el cual siempre está conectado con una resistencia en configuración pull-up [fig1].

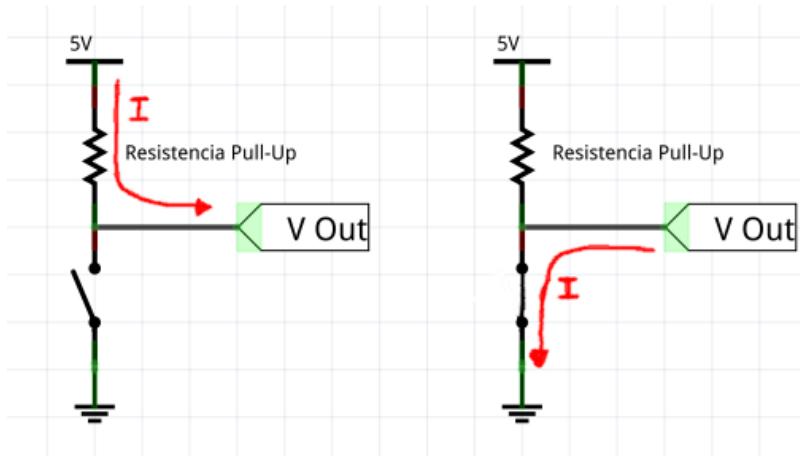


Figura 1: Aunque los dispositivos digitales den dos estados diferenciados HIGH o LOW, debido a factores como el ruido eléctrico o variaciones de voltaje en la fuente dan lugar a equivocaciones en la lectura del estado.

La placa puede alimentarse mediante un puerto USB tipo b, un conector de alimentación externo o los pines laterales de VCC y GND. En la placa se pueden encontrar una gran variedad de pines los cuales vamos a nombrar a continuación:

- 14 pines I/O digitales, de los cuales 6 se pueden utilizar como salidas PWM (Modulación por ancho de pulso, para simular una señal analógica), de 8 bits.
- 6 entradas analógicas, 14 E/S digitales capaces de suministrar una corriente máxima de 40 mA.
- Pines RX0 y TX1 para una comunicación serie. Estos corresponden a los pines digitales 0 y 1 respectivamente.
- Pines digitales 2 y 3 para utilizarlos como interrupciones hardware externas.
- Pines 4 y 5 de las salidas analógicas serían los correspondientes a los pines SDA y SCL respectivamente, los cuales son compatibles con la comunicación del protocolo I2C.
- Pines digitales del 10 al 13 los cuales corresponden a SS o CS, MOSI, MISO y SCK o CLK, para realizar una comunicación SPI.

El ATMega328p además dispone de tres temporizadores distintos y un cristal u oscilador de 16 MHz. A continuación, se incluye una la [\[fig2\]](#) detallando las diferentes partes de la placa UNO revisión R3.

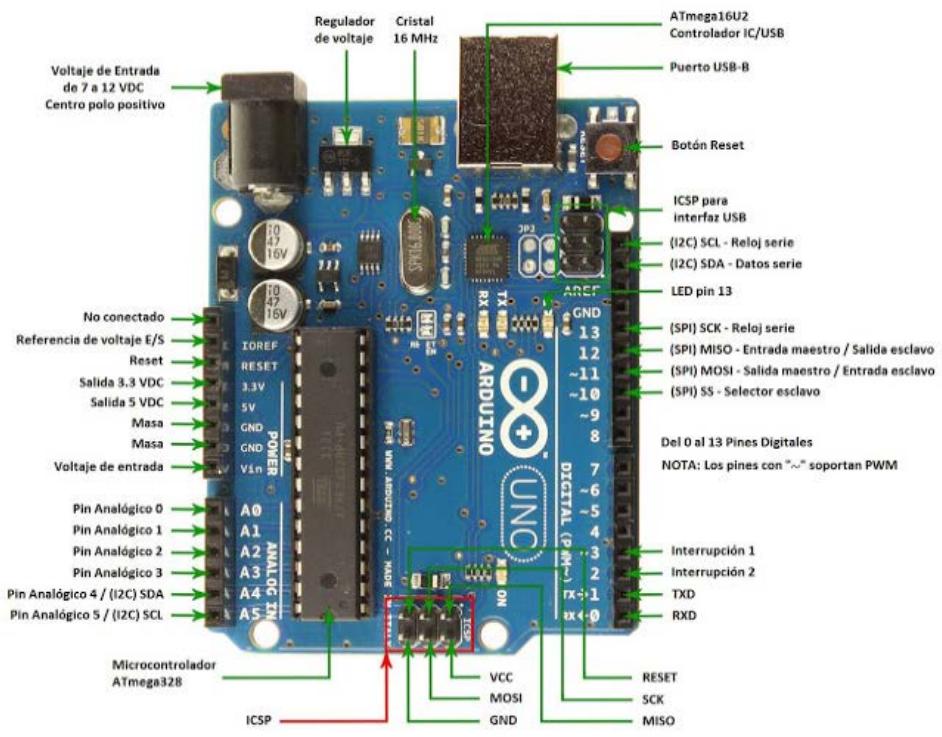


Figura 2: Arduino uno R3, donde se indican cada una de las partes descritas anteriormente.

También es importante nombrar la gran compatibilidad de hardware en el mercado disponible para esta placa, ya que es el modelo más extendido del mercado, llamados SHIELDS o módulos de expansión, utilizando los protocolos 1-Wire, el bus I2C y la comunicación SPI. Además de conocer y utilizar el gran abanico de hardware disponible, debemos utilizar un software específico, el cual también abarca un gran abanico de posibilidades, desde un entorno de programación en código, hasta programación en entorno visual, la cual antes de grabarlo en la placa, se traduciría a código. El software base usado mayoritariamente es el IDE de arduino, también siendo posible hacerlo con AVR Studio o Atmel Studio, software de la compañía fabricante del microcontrolador, o entornos visuales como Miniblock, Ardublock, Amici, etc.

### 5.1.1.2 *Raspberry Pi*

Este dispositivo es un ordenador de tamaño reducido similar al de una tarjeta de crédito. Este ha sido desarrollado por la fundación Raspberry Pi. El objetivo de esta plataforma open source es estimular la enseñanza de ciencia de la computación en las escuelas. Este puede utilizar lenguajes de programación de alto nivel como Python, C++ y java.

Esta placa dispone de un microprocesador ARM con una velocidad de hasta 1GHz integrado en un chip BROADCOM. Cuenta con 512 megas de RAM y una tarjeta gráfica Videocore IV. Para trabajar con esta placa se precisa de una memoria SD o micro SD. Las placas más modernas cuentan con hasta 4 puertos USB para conectar teclado y ratón, además de un conector HDMI con capacidad de reproducción de vídeo a 1080p y una conexión ethernet para conexión por cable a internet. Raspbian, Arch Linux, pidora u OpenElec se encuentran entre los sistemas operativos disponibles para esta placa. Esta placa se puede observar en la [\[fig3\]](#).



Figura 3: Muestra una de las versiones de Raspberry Pi, uno de los miniordenadores con mejores prestaciones y más usado en la actualidad.

#### 5.1.1.3 Nanode

Nanode es una evolución de Arduino desarrollada por Ken Boak del grupo Hackerspace. Para conectarse a éste, se hace necesario el uso de una API y puede incluso utilizarlo como servidor de páginas web simples, permitiendo al usuario configurar el dispositivo. Al igual que Arduino, se programa con el mismo entorno y es abierto pudiéndose programar desde cualquier sistema operativo. La única diferencia entre estos dos es el precio.

Esta placa no presenta apenas diferencia sobre Arduino UNO, ya que monta el mismo microcontrolador ATMega328p. Para describirlo, se exponen las diferencias:

- Este cuenta con un controlador ethernet ENC28J60 junto con un conector hembra para RJ45.
- Este trabaja a un voltaje de 3V3.
- Cuenta con un conector de miniUSB.
- Dispone de un puerto para agregar un módulo inalámbrico RF RFM12B.

A continuación, se muestra en la [fig4], una placa Nanode, donde se puede apreciar el enorme parecido con la placa Arduino UNO, además de portar el mismo microcontrolador que este, el ATMega328p.

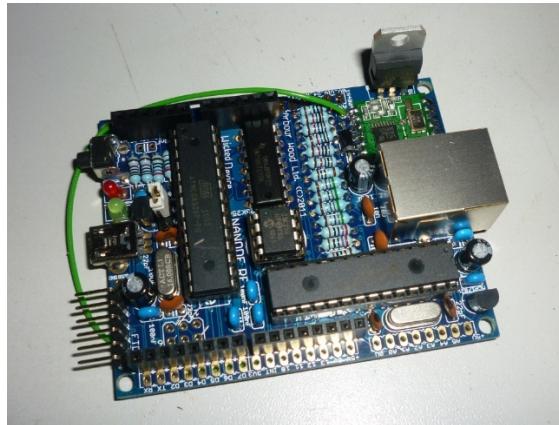


Figura 4: Placa Nanode, donde puede verse la enorme similitud con Arduino, respecto a la cual, presenta algunos cambios y evoluciones.

#### 5.1.1.4 *Libelium Wapsmote*

Es un dispositivo diseñado para crear redes inalámbricas de sensores con unos requerimientos bastante específicos y destinados a ser desplegados en un escenario real. Este dispositivo es comercializado por una startup española ubicada en Zaragoza. El parecido entre este dispositivo y Arduino es su entorno de desarrollo y el código que podemos desarrollar para Arduino, puede ser utilizado en este dispositivo con pequeñas variaciones. A continuación, se adjuntan sus características:

- Microcontrolador ATMega1281
- Frecuencia de trabajo 8MHz
- SRAM 8KB
- EEPROM 4KB
- FLASH 128KB
- SDcard de 2GB
- Consumo: en uso 9mA, modo sueño 62 $\mu$ A, sueño profundo 62 $\mu$ A e hibernación 0.7 $\mu$ A
- E/S digitales 8, 7 análogicas, 1 PWM 2 puertos de comunicación serie uno I2C y puerto USB.
- Tensión de funcionamiento de 3V3 a 4V2.
- Este dispone de un módulo XBee para comunicación inalámbrica mediante el protocolo IEEE 802.15.4 para comunicación entre sensores.

- En su parte trasera dispone de un zócalo para una pila.
- Dispone de un reloj RTC de 32KHz.

En la [\[fig5\]](#), se muestra la placa de Libelium Wasp mote, la cual, como Nanode y Arduino UNO, dispone de un microcontrolador de la familia Atmel, el ATMega1281.



Figura 5: Wasp mote de Libelium, utilizada para comunicación de sensores de forma inalámbrica mediante el uso de XBee.

### *5.1.2 Registradores de datos comerciales*

En este apartado se trata de profundizar con mayor medida en las soluciones comerciales de registrador disponibles en el mercado. Para ello se detallan los expuestos en la introducción.

#### *5.1.2.1 Termohigrómetro PCE-THB 40*

El termohigrómetro detecta la temperatura y humedad ambiental, así como la presión barométrica, y registra tales valores en una tarjeta SD. Con una amplia memoria con un máximo de 16GB, es sobre todo ideal en el registro de datos de larga duración en zonas industriales. El valor de medición actual se muestra directamente en la pantalla LCD y se registra simultáneamente en la memoria, lo que permite una lectura directa del valor, o un análisis en forma gráfica si se traspasan a un PC.

Los valores registrados en un fichero \*.xls en la tarjeta SD, pueden ser traspasados al PC, para su posterior análisis. También puede comprobar si los valores en las columnas sobrepasan cierto límite. El reloj interno con fecha permite asignar con precisión los resultados. La cuota de medición se puede ajustar libremente. Este dispositivo está protegido contra golpes. Sus características técnicas se exponen a continuación:

- Temperatura: De 0 a más de 50 °C con precisión de ±0.8
- Humedad relativa de 10 a 90% con precisión de ±4% del valor medido
- Presión barométrica de 10 a 1100 mbar con precisión de ±2 hPa a 1000 hPa
- Alimentación mediante 6x1,5 pilas AAA/9V
- Precio 191,18€

A continuación, se muestra una imagen del registrador descrito en la [\[fig6\]](#)



Figura 6: Este registrador, es usado en el sector alimentario, aunque también en el industrial, apreciando con facilidad los datos tomados, gracias a su pantalla LCD.

#### 5.1.2.2 *Logger de datos de temperatura y humedad Log110-Start*

Es ideal para el registro del desarrollo de temperatura en el sector del transporte y almacenamiento, y se utiliza también para el control de temperatura en múltiples sectores, registrando además el desarrollo de la humedad relativa.

El datalogger dispone de funciones útiles como la detección de punto de rocío a través de software y puerto para la posible conexión de un sensor externo, verifica la

temperatura en el control de calidad en los sectores de fabricación industrial, laboratorio y almacenaje.

#### Especificaciones técnicas:

- Temperatura interna: De -30 a 70 °C con resolución de 0,1 °C
- Temperatura externa: De -50 a 125 °C con resolución de 0,1 °C
- Humedad relativa interna: Rango de medición de 0 a 99% con resolución de 0,1%
- Dispone de conexión USB para volcar los datos al PC para su posterior análisis
- Alimentación mediante pila CR2032 de 3V
- Precio 228,69€

A continuación, se muestra una imagen del registrador descrito en la [\[fig7\]](#)



Figura 7: Este registrador, es usado en el sector del transporte y almacenamiento, apreciando con facilidad los datos tomados, gracias a su pantalla LCD.

#### 5.1.2.3 Sistema de telecontrol GSM/GPRS Hermes LC-2

Es un sistema de telecontrol y telemetría basado en tecnología GSM que permite monitorizar estaciones remotas de un modo sencillo y eficaz. El sistema de telecontrol Hermes LC-2 se alimenta directamente desde la red eléctrica. El sistema de telecontrol, cuenta con 8 entradas digitales, 2 salidas a relés, que se pueden activar mediante el servicio de mensajería corta SMS y sendas entradas para sondas de temperatura y humedad.

La funcionalidad básica del sistema de telecontrol es de un lado de transmisión de alarmas, temperatura o humedad, señales digitales activas, fallos de red, etc., y de otro lado el registro de datos de cualquiera de sus entradas, para enviarlas más tarde mediante llamadas de datos GSM o GPRS al centro de control.

#### Especificaciones técnicas:

- Alimentación: 220V AC
- Consumo nominal: 0,5W
- Consumo máximo 5W
- Memoria de programa: Flash de 256KB
- Memoria para almacenamiento de datos 64KB
- GSM: Siemens MC55
- Entradas digitales 8
- Sondas de temperatura 4
- Salidas a relé 2
- Puerto de conexión de datos USB
- Precio 336,86€

A continuación, se muestra una imagen del registrador descrito en la [\[fig8\]](#)



Figura 8: Este registrador, dispone de un modem GSM/GPRS integrado, que le dan soporte para telecontrol y telemetría.

#### *5.1.2.4 Estación meteorológica PCE-FWS20*

Equipo multifuncional que cubrirá las expectativas tanto en ámbito privado como profesional. Permite detectar de forma precisa la dirección del viento, la velocidad del viento, la temperatura, humedad relativa y la pluviosidad. Esta estación tiene la posibilidad de activar diferentes funciones de alarma en la estación meteorológica. Los valores meteorológicos se envían por radio a la base a una distancia máxima de 100 metros.

Esta estación meteorológica con la última tecnología en el análisis meteorológico, la pantalla táctil permite recuperar de forma muy sencilla los valores de la estación, el transmisor es alimentado por un módulo solar y dos pilas AAA recargables, el puerto USB permite transmitir los datos desde la estación a un PC, estos datos van acompañados de fecha y hora para poder analizarlos en periodo de tiempo más largo pudiendo memorizar los valores meteorológicos de forma ilimitada. Esta estación, incluye un software de análisis que permite analizar y comparar la fluctuación meteorológica mediante gráficos y diagramas a través de un tiempo prolongado.

#### Especificaciones técnicas:

- Interiores:
  - o Temperatura ambiental de 0 a 60 °C con resolución de 0,1 °C
  - o Humedad del aire del 1 al 99% con resolución del 1%
  - o Presión atmosférica de 919 a 1080 hPa con resolución de 0,1 a 1,5 hPa
- Intervalo de mediciones de 48 segundos
- Duración de alarma de 120 segundos
- Exteriores:
  - o Temperatura ambiental de -40 a 65 °C con resolución de 0,1 °C
  - o Humedad del aire de 1 a 99% con resolución del 1%
  - o Pluviometría de 0 a 9999mm con resolución del 0,1/1mm
  - o Velocidad del viento de 0 a 180 Km/h indicando la dirección del viento
- Transmisión de los datos vía radio hasta un máximo de 100 metros al aire libre
- Conexión de puerto USB
- Alimentación de estación base 3 pilas de 1V5 AAA
- Alimentación del transmisor con módulo solar y 2 pilas recargables
- Precio 104,6 €

A continuación, se muestra una imagen del registrador descrito en la [fig9]



Figura 9: Esta estación meteorológica, cumple casi con todas las premisas de partida, además de presentar un diseño muy elegante donde se pueden ver todos los parámetros ambientales en tiempo real en su gran pantalla táctil.

Como se ha podido apreciar, ningún modelo cumple con todas las premisas de partida, además de alejarse de los objetivos de bajo consumo, bajo coste y la necesidad de tener autonomía fuera del alcance de la red eléctrica general. Como se ha podido observar a lo largo de la descripción de los modelos, los dos primeros se hacen inviables, ya que son modelos portátiles, y el diseño final será colocado fijo en un emplazamiento. El siguiente no dispone de ningún dispositivo integrado para la medición, solo las entradas para conectarlos, además de la necesidad de conexión a la red eléctrica, lo que imposibilita su uso.

Como último ejemplo, se ha detallado la estación meteorológica, que se acerca más que cualquier otro modelo para solventar los problemas encontrados en este trabajo. Algo que no puede solventar es el hecho de tener un bajo alcance para el envío de los

datos, siendo necesario mayor alcance, ya que este trabajo se colocará lejos de cualquier núcleo urbano. Y por último el receptor debe estar en interior, mientras que el diseño final se ubicará en exteriores. Un factor común a todos es un costo medio-alto, incumpliendo otra premisa de este trabajo. Todo esto aleja estas soluciones comerciales como posible uso en el diseño final.

### 5.1.3 Datalogger del trabajo anterior

Esta placa fue construida en el trabajo anterior [1], recoge en un mismo diseño, el máximo de componentes necesarios para el funcionamiento del datalogger principal. El fin principal fue la mejora en resolución gracias al conversor ADC MCP3424, siendo tan fiable y precisa como un registrador de datos comercial de gran calidad y precisión. Este conversor presenta cuatro canales para realizar medidas, tomando consecutivamente sus valores de la entrada 1 hasta la 4, aunque pueden ser tomadas de forma independiente. Al realizar una toma de medida experimentales, se comprobó que los resultados obtenidos, fueron satisfactorios, con lo cual se consigue un ahorro de presupuesto con respecto a otro ya diseñados. Esta placa, para su uso, precisa de librerías para la gestión de la toma de temperaturas, el reloj y el conversor ADC, todo ello con Arduino UNO como se muestra en la [fig10].

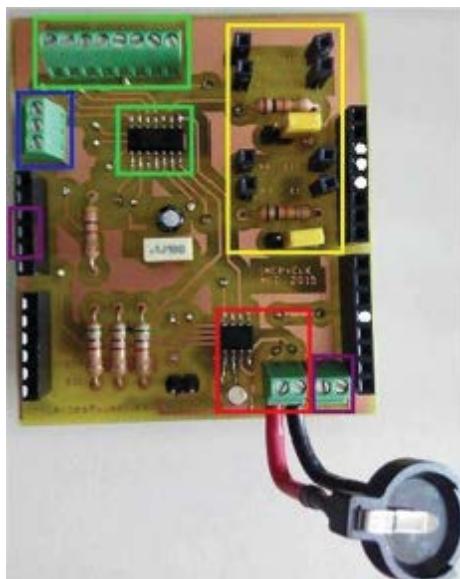


Figura 10: Datalogger que integra divisores de tensión (amarillo) para la medición de sensores analógicos que superen el rango de medida del ADC MCP3424 de 2.048V, el conversor ADC MCP3424 (verde), toma de medidas digitales de temperatura con el ds18b20 (azul), alimentación VCC-GND (morado), reloj RTC DS1307 (rojo) y blanco son las conexiones SPI SCK, MISO, MOSI y SS para la conexión del módulo de expansión de memoria microSD.

Una vez mostrado el conjunto de componentes necesario para la construcción del registrador de datos, el lector debe saber que, para el cumplimiento de los objetivos

finales, también debemos apoyarnos en un software óptimo para nuestro propósito, que realice las tareas de manera eficiente, y sea fiable a la hora de realizar nuestras funciones.

En la siguiente figura se muestra el prototipo final, conectado mediante una protoboard, donde se encuentra la placa Arduino UNO, y el datalogger junto con el módulo de expansión de memoria. A modo de tomar como partida para la explicación de nuestros objetivos finales, se realiza un resumen del software del registrador principal para tomar conciencia de las mejoras que se realizaran a lo largo de este trabajo [1]. En la [fig11] se muestra el prototipo final del registrador secundario.



Figura 11: Prototipo final para cumplir los objetivos del trabajo tomado como partida al aquí expuesto.

## 5.2 Funcionamiento del registrador

Como se ha comentado en el apartado [5.1.3], se muestra un resumen del diagrama de flujo del software necesario para la recogida de datos y presión de medidas. Este código se realizó mediante la IDE de Arduino que es una plataforma o entorno de desarrollo basado en processing/wiring, dos plataformas las cuales están pensadas para usuarios no conocedores a fondo en programación, el cual está pensado para desarrollar potentes herramientas a nivel “visual” y permitirnos crear prototipos electrónicos. Esta plataforma posee un compilador GCC para C/C++, dando la posibilidad de un gran abanico de posibilidades, ya que es un lenguaje muy asentado y con bastante tiempo de uso, el cual posee una gran cantidad de documentación.

Para comenzar, se va a explicar la estructura tipo de un entorno de desarrollo en Arduino, el cual siempre va a estar compuesto de funciones principales. Estas dos funciones son la void `setup(){};`, es la primera en la estructura y generalmente suelen contener la inicialización de las comunicaciones de los buses disponibles, indican si los pines digitales se comportarán como entrada salida, etc. Esta función solo se ejecuta al

inicio, es decir, justo al conectar la alimentación de Arduino, y una vez termina, da paso al void loop(){}, la cual su función es ejecutarse infinitamente, y es aquí donde se suele contener todo el código de programa que queremos que se ejecute en función de nuestras necesidades, ya sea un evento interno, externo, tareas automatizadas, etc.

Ahora ya explicado brevemente el entorno de desarrollo y como se realiza su ejecución principal, vamos a realizar el resumen del código del registrador de datos de partida [1]. Una vez se conecta la alimentación, en el setup se inicializan los sensores digitales, se reduce la frecuencia de operación de Arduino a 1 MHz y se comprueba si las placas de medida están disponibles, en caso de no estarlo, se devuelve un error y se enciende un led rojo, dando lugar a la finalización de la ejecución.

En caso de que la detección sea correcta, se entra en el loop, donde lo primero que se comprueba, es la existencia del fichero donde se guardan los datos de la toma de medidas, y en caso de que exista, se actualiza la hora y nos encontramos con la condición de que solo se comienza la toma de medidas, ya que como se establece en la normativa, la toma de mediciones se realiza cada 30s, por tanto, el código toma medidas en el segundo 0 y en el segundo 30, además previo y post a dicha toma, se produce el parpadeo de un led verde.

Una vez realizado el registro de medidas, el tiempo restante, el arduino, entra en modo sleep power down con las librerías jeelib incluidas. Al concluir las mediciones, se actualiza la hora y se encuentra la condición de si es la hora 00:00:00, en caso afirmativo se crea un fichero nuevo para la recogida de datos, esto quiere decir que se crea un fichero nuevo cada día. En caso de no cumplir la condición volvería a repetirse el loop. Mediante su testeo, se pudo comprobar que el código diseñado cumplía con los requisitos finales, pero aún se pretendía realizar ciertas mejoras. En la [fig12] se puede ver el fluograma del software utilizado, con todas las posibles mejoras que se decidió implementar.

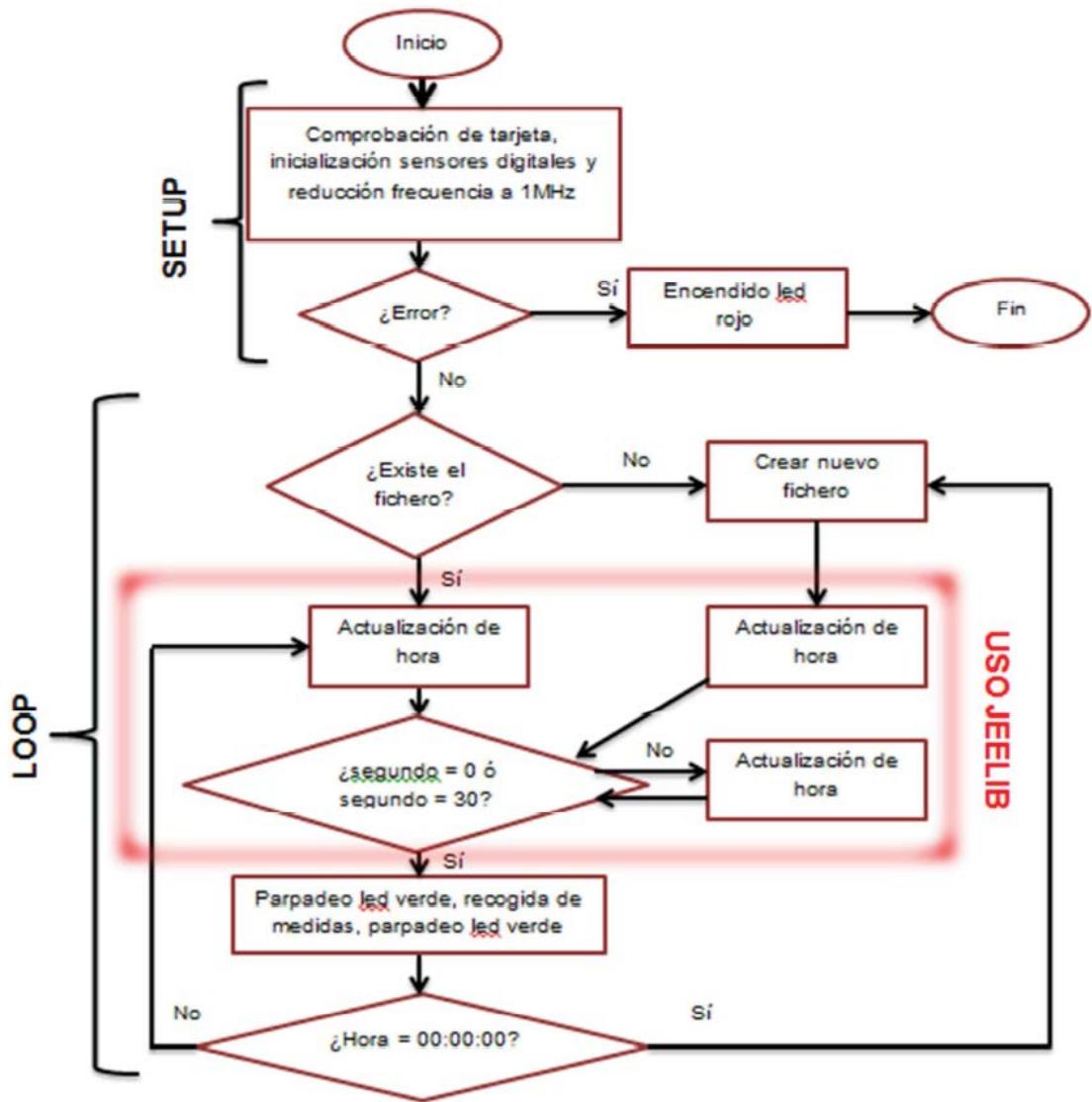


Figura 12: Diagrama de flujo del registrador de datos de tomado como partida.

### 5.3 Soluciones usando microcontroladores Atmel

Como se ha podido observar en el apartado [5.1.1], de las plataformas descritas, todas están basadas en microcontroladores de la familia Atmel, a excepción de Raspberry Pi. A pesar de esta excepción, se considera conveniente y necesario exponer con detalle las posibles soluciones a la problemática encontrada, haciendo mención de las técnicas utilizadas para optimizar ciertas premisas: como consumo, comunicación inalámbrica a larga distancia, etc.

#### 5.3.1 Hardware

Una vez comentada la introducción de los tipos de optimización existentes, hay que señalar el anexo [\[12.2\]](#), cuyo contenido muestra el montaje del microcontrolador con los componentes sueltos necesarios para su funcionamiento, sin el uso de placas comerciales. No obstante, puede hacer uso de cualquier shield aunque no se ajuste a la forma de la placa comercial. También se añade el uso de un regulador de voltaje DC-DC, ya que poseen una eficiencia de conversión de un 90% frente al 40% de los reguladores lineales, pudiendo caer hasta un 15% con facilidad, uso placas de bajo consumo, ya sean comerciales o alternativas.

#### *5.3.1.1 Tipos de microcontroladores*

En este apartado se esboza cuáles son las posibles soluciones de microcontroladores que hay en el mercado, que puedan solventar la problemática de trabajo. Para ello se adjunta tabla la [\[tabla 1\]\[30\]](#), donde se encuentran los chips más utilizados de la compañía Atmel. Un factor de todos ellos excepto del 2560p, es que el tipo de encapsulado es DIP, lo cual facilita su montaje en placas de prototipado.

Microcontroller	ATmega 16	ATmega 328P	ATmega 32	ATmega 644	ATmega 1284/1284P	ATmega 2560P
Max Frequency	16MHz	16MHz	16MHz	20MHz	<b>20MHz</b>	16MHz
Digital Pins	32	23	32	32	<b>32</b>	54
Analog input	8	6	8	8	<b>8</b>	16
SRAM	1k	2k	2k	4k	<b>16k</b>	8k
FLASH	16k	32k	32k	64k	<b>128k</b>	256k
EEPROM (Bytes)	512	1024	1024	2048	<b>4096</b>	4096
UART	1	1	1	2	<b>2</b>	4
Interrupts	3	2	3	3	<b>3</b>	8
PCINT Interrupts	no	23	no	32	<b>32</b>	54
PWM pins*	4	6	4	6	<b>6</b>	16

Tabla 1: Tabla comparativa de microcontroladores ATMega de encapsulamiento tipo DIP, excepto el 2560p que es del tipo QFP (encapsulado cuadrado plano) o encapsulado de circuito integrado utilizado para montaje superficial SMD.

#### *5.3.1.2 Placas de bajo consumo*

Otra de las alternativas disponibles en el mercado para la reducción de consumo, es el uso de placas de minúsculo tamaño, donde solo se dispone de los componentes necesarios para su funcionamiento. Además, se reduce su velocidad de procesamiento, dejando de lado los relojes externos, y el uso de placas alternativas a las comerciales, las

cuales deberemos estudiar en función de nuestras necesidades de uso, compatibilidad de shields, precio, etc. A continuación, se van a ir nombrando las placas comerciales y después las alternativas, dando una breve descripción de sus características de funcionamiento:

- Placas comerciales:

Arduino Pro Mini [\[40\]](#):

Esta placa viene en dos versiones, una que se alimenta a 5V/16Mhz y 3.3V a 8 MHz. Su ventaja es que tiene los mínimos componentes necesario para su funcionamiento, lo que disponga de un conversor USB-UART que esté consumiendo constantemente. En la [\[fig13\]](#) se muestra dicha placa.



Figura 13: Arduino Pro Mini. Destaca su reducido tamaño, donde se prescinde de patillaje de conexión y puerto usb y Jack de alimentación.

○ Arduino Lilypad [\[41\]](#):

Su reducido tamaño y escasez de componentes lo hacen ideal para reducir su consumo. Se suele utilizar en prendas y textiles y dispone de dos tipos de microcontroladores, el ATMega168V y el ATMega328V, de mayor rendimiento. Ambos trabajan a una frecuencia de 8 MHz, donde el primer modelo trabaja a 2.7V y el segundo a 5.5V. En la [\[fig14\]](#) se muestra el pequeño diseño de esta placa.

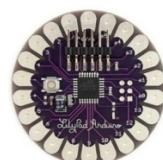


Figura 14: Arduino LilyPad, usado para llevar en prendas (aunque no es estricto en este ámbito), donde por necesidad, es imprescindible un bajo consumo, ya que no se puede dotar al usuario de baterías de gran tamaño.

- Placas alternativas:

- Arduino Moteino [\[42\]](#):

Trabaja a 3.3V y 16 MHz donde como en las versiones comerciales, anteriormente comentadas, su reducido tamaño, bajo voltaje de funcionamiento y solo el uso de los componentes necesarios, lo hacen idóneos para proyectos de bajo consumo. Está basada en el mismo microcontrolador que la placa Arduino UNO, el ATMega328p AU, empaquetado en versión de montaje superficial. Además, dispone de un chip para comunicación y programación inalámbrica Wireless. A continuación, se muestra esta pequeña placa en la [\[fig15\]](#).



Figura 15: Arduino Moteino, donde las características descritas, lo hacen idóneo para proyectos de bajo consumo.

- Arduino Mosquino [\[43\]](#):

Mosquino es una placa derivada de Sanguino y fabricada para diseños de bajo consumo. Su diseño está pensado para obtener los menores consumos posibles. No utiliza un microcontrolador convencional utilizado en placas comerciales, sino una variante de mayor tamaño y con ciertas ventajas al ATMega328p, montado en la placa Arduino UNO, y respecto a este, tiene mayor SRAM, EEPROM, FLASH y un mayor número de pines.

Dicho microcontrolador se llama ATMega644PA y las diferencias más subyacentes son que, puede trabajar con tan solo 1.8V y a ese voltaje, el reloj trabaja a 4 MHz, aunque tendría la posibilidad de hacerlo a 20 MHz. Para integrarlo con la IDE de arduino, es necesario realizar el procedimiento [\[12.2\]](#) detallado en anexos. Un dato curioso, lo que haría tener falta de compatibilidad con ciertos shields comerciales, es que los pines hembra para sus entradas/salidas, son de menor tamaño que los normales, en concreto 0.1 pulgadas como se puede ver en la [\[fig16\]](#), aunque puede hacerlo compatible con el resto de bareboard.

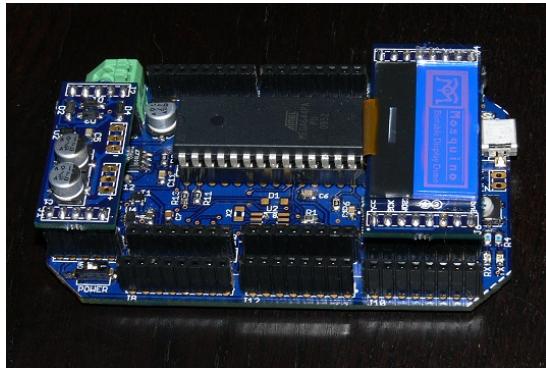


Figura 16: Placa Mosquino, diseñada con un mayor número de pines hembra, y como se aprecia de menor espacio.

Una vez realizado el análisis del apartado de optimización de consumo con las diferentes soluciones hardware, como cualquier buen diseño en el dispositivo, va acompañado de una buena implementación software que hagan realizar las tareas para las cuales se ha creado, de la forma más óptima y fiable posible, sin olvidar el consumo.

### 5.3.2 Software

Este apartado está dedicado a la optimización energética basada en software, para microcontroladores de ATMEL con arquitectura AVR 8-bits RISC (Reduced Instruction Set Computer o Computador con Conjunto de Instrucciones Reducidas), donde en nuestro caso, siguiendo el hilo argumental, se explica en base a el micro ATmega328p, pero como se acaba de comentar, es válido tanto para ATmega8, ATmega168, ATmega2560, etc., los cuales mantienen la misma arquitectura.

A continuación, se trata de explicar las soluciones vía software que abarcan desde la modificación de la frecuencia del reloj de CPU, timers, prescaler, interrupciones, modos durmientes de Atmel y el uso de librerías. Una nota antes de comenzar. Si el micro se programa a través de un programador ISP, no es necesario el uso de bootloader. Esto ahorra espacio en FLASH (memoria de programa) y se produce una carga más rápida del programa escrito. En el caso de usar la placa comercial UNO, se podría anular el programador USB de la placa, el ATmega16U2 [44], cortando la pista que lo alimenta, ya que el uso del bootloader solo es necesario en caso de la recepción de datos vía UART (serie).

En caso de optar por esta opción, debemos de pensar que no podríamos escribir el programa desde el puerto USB alojado en la placa, ni realizar una comunióñ serie desde ese conector, pero seguiría siendo posible en caso de usar cualquier tipo de

programador externo como por ejemplo un FT232, usbASP, o similar, a través de sus pines RX y TX, o incluso una placa de Arduino programada como un programador ISP.

### 5.3.2.1 Reducción de la frecuencia de oscilación

Los microcontroladores Atmel, suelen utilizar un cristal de cuarzo de 16 MHz. También es necesario a la hora de descargar los programas a la memoria flash del micro. Dependiendo de la configuración de los fusibles de configuración que se le indican al bootloader, puede utilizar este cristal externo, o puede usar su propio reloj interno, este de 8 MHz, siendo necesaria la previa modificación del bootloader indicando su uso.

A pesar de poder usar los dos osciladores, dependerá del cometido para el que se requiera usar, debido a que el cristal externo, presenta una mayor mejora en cuanto a tareas donde se requiera mayor número de operaciones, más rapidez y precisión. Este procedimiento se llevará a cabo de la siguiente manera, y es mediante el uso del prescaler o escalado dinámico del registro CLKPR y marcar un nuevo valor de frecuencia de trabajo. Para ello deberemos cambiar el archivo board.txt como se muestra en las [fig17] y [fig18] modificar la frecuencia de trabajo que tendrá el chip que se decida usar.

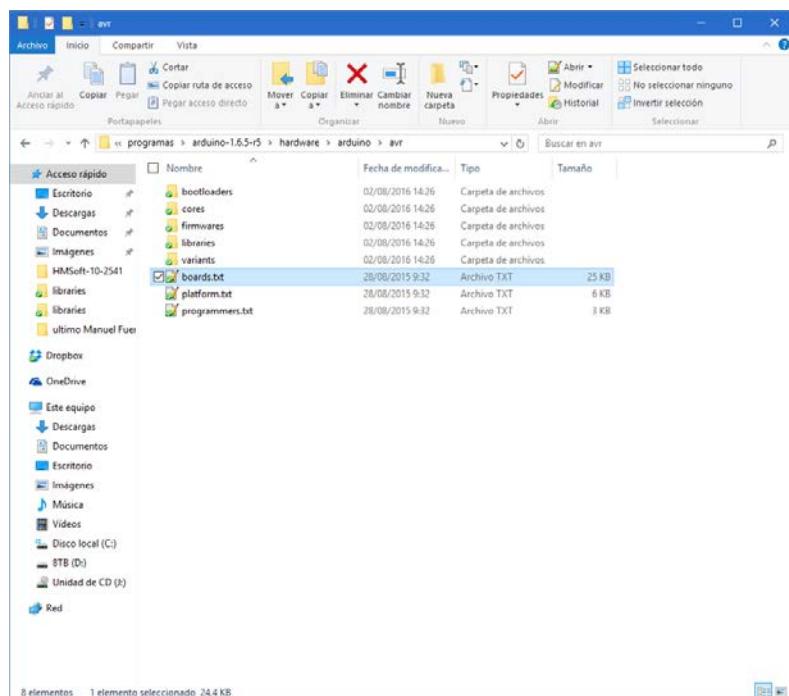


Figura 17: Ruta y fichero a modificar, para indicar la nueva frecuencia de funcionamiento, una vez cambiada la frecuencia por software a través del escalado dinámico.

```
TC:\Users\RafaBab\Dropbox\programación\arduino\programas\arduino-1.6.5\hardware\arduino\avr\boards.txt - Notepad -  
Archivo Editar Borrar Xista Copiar Seleccionar Configuración Lenguaje Configuración Macro Ejecutar Plugins Ventana I  
Archivos de texto [1]  
[Arduino Uno]  
uno.yield_time_us=1000000  
uno.build.heartrate=192_FDN  
uno.build.core=arduino  
uno.build.variant=yun  
uno.build.extra_flares=Build.usb_flares  
uno.usb_name=Arduino/Genuino Uno  
uno.vid=0x04D1  
uno.pid=0x0003  
uno.vid=0x04D1  
uno.pid=0x0001  
uno.vid=0x04D1  
uno.pid=0x0003  
uno.pid=0x0044  
uno.vid=0x04D1  
uno.pid=0x0243  
uno.upload_tool=avrduude  
uno.upload.protocol=avrdude  
uno.upload.maximum_size=32256  
uno.upload.maximum_data_size=2048  
uno.upload.speed=15200  
uno.bootloader=ATmega16U2  
uno.bootloader.lib_file=avr32FT  
uno.bootloader.lib_hex=avr32FT  
uno.bootloader_extended_fuses=0xD5  
uno.bootloader.unlock_bits=0x3F  
uno.bootloader.lock_bits=0x0F  
uno.bootloader_file=optiboot/optiboot_atmega32.hex  
uno.build.core=atmega32  
uno.build.f_cpu=16000000L // Esta valor se deberá modificar en función de la frecuencia de operación asignada, por ejemplo 10000000 si lo queremos a 10Mhz  
uno.build.board=AVR_JNC  
uno.build.core=arduino  
uno.build.variant=standards  
##  
## DUE/DUEMILASOLO  
diecimila.name=Arduino Due/Duemilasolo or Diecimila  
diecimila.upload_tool=avrduude  
diecimila.upload_protocol=arduino  
diecimila.upload.maximum_size=1024000  
diecimila.bootloader_hex=avr32FT  
diecimila.bootloader_lock_bits=0x3F  
diecimila.bootloader_unlock_bits=0x0F  
diecimila.bootloader_file=optiboot/optiboot_atmega328p.hex  
diecimila.build.f_cpu=16000000  
diecimila.build.board=DUE/DUMLASOLO  
diecimila.build.core=arduino  
diecimila.build.variant=standard  
## Arduino Due/Duemilasolo or Diecimila w/ ATmega328  
Normalized file  
length: 25134 lines: 843 Ln:74 Col:26 Sel: 9|1 UNX UFT-8 ING
```

Figura 18: Modificación del fichero boards.txt en concreto para el micro ATMe328p, y detallado como se debe actuar en caso de usarlo a 1 MHz.

Una vez modificado boards.txt, en el programa deberemos introducir unas líneas de código para realizar el escalado y fijado de frecuencia a 1 MHz, que sería la más baja posible:

CLKPR=(1<<CLKPCE);  
CLKPR=B00000100;

A continuación, se adjunta en la [\[tabla 2\]](#) la cual indica los valores del prescaler, el prescaler binario y como quedaría la nueva frecuencia, en función de la que se use por defecto.

Valor prescaler	Prescaler binario	Nueva frecuencia (MHz)
1	000	16
2	001	8
4	010	4
8	011	2
16	100	1

Tabla 2: Posibles valores de escalado dinámico CLKPR. Los últimos valores del registro, son los que indican la frecuencia de funcionamiento del microcontrolador.

### 5.3.2.2 Modos sleep microcontroladores ATMega AVR-8 bits

Los modos sleep en los micros, son los encargados de desactivar ciertas partes de él, en función de las que sean necesarias para su uso y, además, dependiendo de las necesidades de ahorro energético que se persigan. La autonomía de sistemas, donde la alimentación sea algo determinante, se ve muy afectada en función del diseño más o menos óptimo del software, donde un ejemplo muy sencillo es evitar el uso desmedido de la función `delay()`. Existen 6 formas de poner en modo durmiente, o más correctamente hablando, 6 formas en las cuales se desactivan ciertas partes internas del micro, en función de nuestras necesidades de ahorro energético. Antes de disponernos a su breve explicación, se adjunta la [\[tabla 3\]](#) indicando las distintas partes que se desactivan dependiendo del modo usado.

	Active Clock Domains					Oscillators		Wake-up Sources							
	clk <sub>CPU</sub>	clk <sub>FLASH</sub>	clk <sub>IO</sub>	clk <sub>ADC</sub>	clk <sub>ASV</sub>	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPI/MEEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X <sup>(2)</sup>	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X <sup>(2)</sup>	X <sup>(3)</sup>	X	X <sup>(2)</sup>	X	X	X		
Power-down								X <sup>(3)</sup>	X				X		X
Power-save					X		X <sup>(2)</sup>	X <sup>(3)</sup>	X	X			X		X
Standby <sup>(1)</sup>						X		X <sup>(3)</sup>	X				X		X
Extended Standby					X <sup>(2)</sup>	X	X <sup>(2)</sup>	X <sup>(3)</sup>	X	X			X		X

Tabla 3: Tabla que indica los modos sleep de Atmega, además de indicarnos que tipos de osciladores funcionan y formas de despertarlos en esos modos.

- SLEEP\_MODE\_IDLE: Como se puede apreciar, es el modo en el cual menos partes se desactivan, y, por ende, entre los modos sleep, es en el dónde más se consume.
- SLEEP\_MODE\_ADC
- SLEEP\_MODE\_PWR\_DOWN: Es el modo más agresivo de todos, pues como se puede observar, es el que menos partes activas mantiene, solo pudiendo ser sacado de este estado, mediante interrupciones externas hardware y el timer watchdog.
- SLEEP\_MODE\_PWR\_SAVE
- SLEEP\_MODE\_STANDBY
- SLEEP\_MODE\_EXTENDED\_STANDBY

Como hemos podido comprobar, estos micros pueden reducir drásticamente su consumo utilizando estos modos de funcionamiento, pero esto no serviría de nada sino pudieran ser sacados de ahí. Este proceso se conoce con el nombre de 'wake up' o despertar del modo durmiente. En la tabla aparece la forma que tiene cada modo para poder ser despertado. A continuación, se indica como poder sacar a los micros de la familia Atmel de estos estados.

- Wake up por medio de interrupciones externas.
- Wake up por medio de interrupciones por puerto serie (UART).
- Wake up por medio de interrupciones programadas y timers.
- Wake up por medio del timer watchdog o perro guardián.

#### *5.3.2.3 Wake up mediante timers e interrupciones programadas y hardware*

Todos los microprocesadores y microcontroladores poseen temporizadores que permiten controlar los ciclos de reloj transcurridos, apoyándose en los relojes que poseen. Se encargan de supervisar cuando se realiza una tarea o evento del sistema, sin necesitar estar supervisando el tiempo constantemente. Este tipo de interrupciones se conocen con el nombre de interrupciones programadas y su objetivo es dispararse una vez cumplido el tiempo o la hora fijada.

El ATMega328p, dispone de 3 tipos de timers distintos capaces de gestionar distintas funciones y pueden ser configurados con distintos modos de funcionamiento.

- Modo PWM: para generar de forma simulada, ondas sinusoidales, variando el ciclo de trabajo de la señal cuadrada.
- Modo CTC: Cuando el contador del temporizador, alcanza el valor del registro de disparo (CMR o Compare Match Register), el valor del timer se reinicia.

Estos timers, pueden ser modificados desde los registros. El primero es el timer0 de 8 bits, 256 valores, y es el encargado de gestionar las funciones delay(), millis(), micros() y delayMicroseconds(). El siguiente es el Timer1, el cual tiene 16 bits, es decir, 65536 valores, pero modificables para una mejor precisión. Por último está el timer2, también con 8 bits modificables y encargado de las funciones tone() y noTone(). Estos timer pueden ser modificados a través de unos registros, donde los más importantes, son los siguientes:

- TCNTx: Registro temporizador/contador. Almacena el tiempo actual.

- OCRx: Registro comparador de salida.
- ICRx: Registro capturador de salida.
- TIMSKx: Registro temporizador/contador más de interrupción. Permite habilitar y deshabilitar las interrupciones del timer. Esto se hace necesario en dos ocasiones concretas. La primera es cuando se están configurando los registros del timer para su duración y prescaler, cuando se está ejecutando la interrupción y en caso de tener más de una interrupción, para evitar que el evento o acción definida en la interrupción (ISR o interrupción con el servicio de gestión) se vea afectada por otra o ella misma.
- TIFRx: Registro temporizador/contador de flag de interrupción. Indica si existe alguna interrupción de timer pendiente.
- TCCR<sub>x</sub>: Registro temporizador/contador de control. Permite configurar los prescaler. Es uno de los más relevantes.

A continuación, se muestran en la [fig19] el funcionamiento de los registros descritos más arriba.

#### TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk <sub>I/O</sub> /(No prescaling)
0	1	0	clk <sub>I/O</sub> /8 (From prescaler)
0	1	1	clk <sub>I/O</sub> /64 (From prescaler)
1	0	0	clk <sub>I/O</sub> /256 (From prescaler)
1	0	1	clk <sub>I/O</sub> /1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Figura 19: Registro TCCR<sub>x</sub> 1A y 1B. Dispone de 16 bits repartidos entre los dos, 8 bits para cada uno. Los bits CS12, CS11 y CS10, permiten configurar los prescalers.

Para una mejor comprensión y seguimiento del hilo argumental, se explica brevemente que son los prescalers, debido a su posible uso tanto para los timers, como para la reducción de frecuencia de trabajo del micro.

Como se ha comentado con anterioridad, la placa ATMega32p, trabaja a 16 MHz, y teóricamente se podrían fijar interrupciones cada  $\frac{1}{16} \text{ MHz} = 62.4 \text{ ns}$ , tiempo que tarda un ciclo de reloj. Este valor no es útil debido a que es muy bajo, pero como hemos visto los registros de ATMega328p, tienen un número de bits configurables, los cuales se pueden usar para contar cada cuantos ciclos de reloj podría saltar la interrupción.

- Timer 0 y 2: 8 bits  $\rightarrow 2^8 = 256 \rightarrow \frac{256}{16000000} = 16 \mu\text{s}$ .
- Timer 1: 16 bits  $\rightarrow 2^{16} = 65536 \rightarrow \frac{65536}{16000000} \cong 4 \text{ ms}$ .

Estos nuevos valores nos facilitan mayor margen de tiempo para establecer una posible interrupción, sobre todo con el timer1. Aun así, este margen tampoco es muy válido en el caso de querer generar eventos cada hora o cada día. En este punto es donde se utilizan los prescalers o divisores de tiempo. Como se mostró con anterioridad, existen diferentes divisores de tiempo [fig19] controlados por los bits CS (Clock Selection), encargados de guardar el registro que coincidirá con el valor indicado para disparar la interrupción. Para tratar de aclarar al lector, se propone un ejemplo:

$$\left. \begin{array}{l} - 2s = 0.5 \text{ Hz} = f_{interrupción} \\ - CMR = \frac{f_{procesador}}{Prescaler * f_{interrupción}} \end{array} \right\} \rightarrow CMR = \frac{16000000 \text{ Hz}}{1024 * 0.5 \text{ Hz}} = 31250$$

Cuando obtenemos el CMR, hay que comprobar cuál de los posibles timers puede ser usado.

- $256 < 31250 < 65536$

Como se puede ver, timer 0 y 2 no son válidos debido a que el valor de CMR es mayor a su rango, por tanto, solo sería posible usarlo con el timer 1.

Las interrupciones programadas por timers y los divisores de tiempo, pueden ayudarnos a una optimización en el consumo energético de nuestro diseño y además aprovechar los intervalos en los que el datalogger no trabaje para controlar los consumos instrucciones distintas a `delay()`, que sume a nuestro micro es una espera donde no

puede realizar ninguna acción. De esta explicación y ejemplo, se puede abstraer la posibilidad del uso de librerías que traten sobre el aprovechamiento energético.

#### 5.3.2.4 Wake up mediante watchdog

El timer watchdog, conocido como perro guardián, está basado en un mecanismo de seguridad diseño para microcontroladores y microprocesadores. Este sistema, realiza un reinicio cada cierto tiempo en caso de bloqueo del hardware. Este timer, tiene un temporizador que cuando llega a cero provoca un reinicio, y debido a esto, dispone de su propio oscilador de 128 KHz. Este presenta varias utilidades como asegurar que el sistema siempre estará disponible además de sacar a nuestro micro de uno de sus modos durmientes.

Por medio del WDT\_vect y el ISR (Interrupt Service Routine o Rutina de Servicio de Interrupción) se puede despertar al micro e incluso volverlo a dormir. Hay que mencionar, que las funciones de servicio de interrupción, debido a que las interrupciones son algo que paraliza totalmente el hardware, ya que por disposición indican algo que debe ser ejecutado inmediatamente, estas rutinas deben ser breves y ejecutarse en el menor tiempo posible, como por ejemplo realizar un conteo, despertar o dormir la MCU, almacenar o extraer algún dato, etc. El watchdog puede ser utilizado como timer, reset o ambos como se ve en la [fig20]:

WDTON	WDE	WDIE	Mode	Action on Time-out
0	0	0	Stopped	None
0	0	1	Interrupt Mode	Interrupt
0	1	0	System Reset Mode	Reset
0	1	1	Interrupt and System Reset Mode	Interrupt, then go to System Reset Mode
1	x	x	System Reset Mode	Reset

Figura 20: Configuración del watchdog timer.

Al ser un timer como los descritos anteriormente, también tiene la posibilidad de utilizar un prescaler para obtener un mayor o menor tiempo de espera. Estas modificaciones las lleva el WDTCSR o registro de control del temporizador perro guardián, compuesto por 4 bits. Gracias a esto el micro puede realizar interrupciones entre un rango temporal de entre 8 a 16 ms. Este modo de wake up, es el más indicado según lo descrito para sacar a nuestro micro de su estado de sueño, pudiendo ser el

SLEEP\_MODE\_PWR\_DOWN, el modo más agresivo y donde se obtiene el mayor ahorro energético.

### 5.3.2.5 *Uso de librerías de bajo consumo*

Debido a la filosofía seguida en el desarrollo de placas comerciales de hardware libre, en la red existen un gran número de posibilidades, ya que cualquier usuario puede contribuir aportando sus conocimientos y diseños al gran abanico ya existente en la actualidad. Esto genera que entre esa gran cantidad de material se encuentre un gran número de ejemplos de diferentes diseños en función la tarea para la cual haya sido desarrollado ese programa y por ello en la web se puede encontrar un gran número de librerías para poder dar soporte a todo tipo de shields y componentes, que aun sin aparecer en algún tipo de shield, se podrían integrar en nuestro diseño teniendo ciertos conocimientos en diseño de placas de circuitería impresa o mismamente en protoboards.

En este apartado se proponen el uso de librerías para un manejo sencillo y rápido de todos los modos de funcionamiento para colocar nuestro micro en modo durmiente, y como no las instrucciones necesarias para poder despertarlo mediante wake up. A continuación, se van a nombrar varios ejemplos de un gran número de posibilidades de este tipo de librerías: JeeLib, Narcoleptic, Enerlib, Sleep\_n0m1, Low power Lighthead, la propia desarrollada por Atmel, etc.

El lector puede preguntarse el porqué del uso de otras librerías externas a las usadas por el fabricante, y esto es debido a una menor flexibilidad a la hora de utilizar los diferentes modos, por ejemplo, como se explicó anteriormente en el apartado [5.3.2.4], mediante el prescaler y el timer1, este nos permitía dormir nuestro micro durante un máximo de 16s. Esto es un factor bastante limitante si un desarrollo necesita dormir durante todo el tiempo que fuese necesario y despertarse solamente cuando se realiza cualquier evento externo mediante una interrupción hardware.

Debido a esto se realiza el uso de librerías diseñadas por terceros, ya que además de un uso mucho más sencillo, ya que el uso de las librerías del fabricante requiere de un vasto conocimiento a nivel de hardware, y además por la flexibilidad ofrecida en los tiempos pudiendo dormir nuestro micro de forma infinita, se realiza el uso de estas.

## 5.4 Comunicación

En este apartado, se tratará de explicar los diferentes modos desde los cuales ATMega1284p es capaz comunicarse hacia el exterior. No se tratarán los tipos de comunicaciones usadas por los sensores para comunicarse con él, debido a que solo se usan para ese fin, y solo se nombrarán las cuales permiten establecer la comunicación para la que cualquier persona sea capaz de visualizar esa comunicación.

Como en cualquier dispositivo, siempre ha sido necesario establecer un canal mediante el cual establecer una escucha de la información que puede ofrecer, como tanto para enviarle datos para que realice una consulta o hacerle una petición, y realice cualquier tarea. Esto no es necesario en algunos sistemas que son totalmente autónomos, lo que implica que no necesaria darle ninguna instrucción después de su puesta en marcha, ya que las tareas son prefijadas para que se realicen de manera constante. En este trabajo, el fin es grabar unas tareas que se ejecutarán siempre que el dispositivo esté activo, al cual le indicaremos que nos devuelva una información para realizar su posterior procesado y poder visualizarla para obtener lo que se necesite de ella.

### 5.4.1 *Comunicación con hilos*

En este apartado de trata de hacer un esbozo de los tipos de comunicación posible mediante cableado.

#### 5.4.1.1 *Comunicación Serie (UART)*

El puerto serie es la forma principal de comunicación de las soluciones expuestas en antecedentes con un ordenador. Este tipo de puertos tienen una gran cantidad de funcionalidades debido a que hace tiempo que están implantados, por ejemplo, mover el ratón o simular la escritura de un usuario en el teclado, controlar otros dispositivos, etc. Este tipo de puertos envía la información mediante una secuencia de bits. Para ellos se necesitan al menos dos conectores para realizar la comunicación, un TX (transmisión) y un RX (recepción). Se puede referirse a ellos como UART (universally asynchronous receiver/transmitter) que es una unidad que incorpora ciertos procesadores, encargada de realizar la conversión de los datos a una secuencia de bits y transmitirlos o recibirlas a una velocidad determinada.

En algunas ocasiones, también es posible que a este tipo de puertos se les pueda dar el nombre de TTL (transistor-transistor logic). Esto significa que la comunicación se realiza mediante variaciones en la señal entre 0V y Vcc, donde Vcc suele ser entre 3.3V o 5V, donde otros sistemas serie como RS-232 varían entre -Vcc y +Vcc, típicamente entre -13V y +13V. Prácticamente todas las placas poseen al menos un puerto UART. Esto significa que mediante el conversor USB-UART ATMega16U2 [44], sería posible una comunicación directa con un ordenador. Estos puertos, están físicamente unidos a distintas partes de la placa, y mientras son usados, no es posible usar estos pines asociados. En la [fig21] se muestra la comunicación serie de dos placas Arduino UNO.

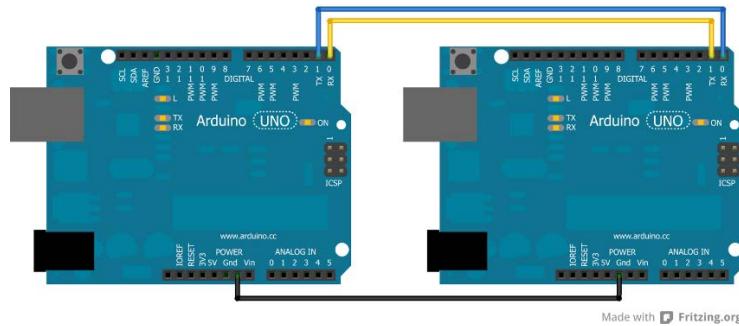


Figura 21: Ejemplo de comunicación serie entre dos arduinos.

Otro dato importante, es que además de los puertos físicos disponibles en cada microcontrolador, gracias a la librería “SoftwareSerial”, se puede convertir cualquier par de pines para realizar una comunicación serie, lo cual aumenta las posibilidades de comunicar con varios dispositivos.

#### 5.4.1.2 Comunicación Ethernet

Las placas comerciales de hardware libre vistas en antecedentes, son capaces de montar una conexión Ethernet gracias a la Ethernet Shield y con esto conectarse de forma local a una red e incluso a internet, con lo que conlleva un gran potencial de uso. Este tipo de comunicación utiliza un conector RJ45 para realizar la conexión entre la placa y el router que da acceso a red local o internet. Esta shield dispone de las siguientes características:

- Tensión de alimentación 5V
- Controlador Ethernet: W5100 con memoria de 16KB
- Velocidad de conexión: 10/100Mbps
- Conexión mediante puerto SPI

Esta shield está construida con el chip Wiznet W5100 que proporciona una red IP usando los protocolos UDP y TCP. Soporta hasta cuatro conexiones de socket simultáneas. Además, dispone de un conjunto de LEDs de estado para proporcionar información de los siguientes estados:

- PWR: led de alimentación
- LINK: indica que la shield se ha conectado a la red y parpadea cuando en la recepción/envío de datos.
- FULLD: indica que la comunicación es full duplex.
- 100M: indica que la conexión tiene una velocidad de 100Mbps
- RX: indica recepción de datos
- TX: envío de datos
- COLL: indica si se producen colisiones de paquetes en la red

El jumper soldado como “INT” puede ser conectado para permitir a la placa recibir notificaciones de eventos por interrupción desde el W5100. Esta placa también dispone de un slot de expansión de memoria microSD para almacenar archivos que puedan ser usados en la red. Este módulo se puede conectar a otro dispositivo mediante un cable de par trenzado de CAT5 o CAT6. En la [\[fig22\]](#) se puede ver el módulo ethernet.



Figura 22: Shield Ethernet, el cual dota a arduino de conexión de área local o internet.

Como hemos podido ver, esta shield dispone de un gran potencial para la problemática al envío de datos a internet, pero tiene el factor limitante de que tendría que usar un cable para poder comunicarse él. Esto podría solventarse si como intermediario se utiliza un AP (punto de acceso) inalámbrico el cual podría agregar una mayor distancia

y evitar el uso de cables hasta un punto de conexión a internet, solamente necesitando cable para la conexión de la shield al AP.

### **5.4.2     *Comunicación sin hilos***

La comunicación sin cables, presenta grandes ventajas frente a las que los utilizas en cuanto a que no existen cables físicos que entorpezcan la transitabilidad o que molesten estéticamente, reduce los costes, puesto que, a mayor distancia, por ende, se necesitarían una mayor cantidad de cable, es más sencilla su instalación debido a que no necesitan de conectores, permiten un gran alcance, dan más libertad de movimiento, etc.

Todas estas ventajas las hacen más atractivas frente a las desventajas que tienen como por ejemplo un menor ancho de banda (dependiendo de su uso), menos seguras (es cierto, pero muchas de estas tecnologías, disponen de librerías de encriptación de datos como base64, no muy buena, o AES, uno de los mejores sistemas de cifrado hasta la fecha, con lo cual podría implementarse una red muy segura con esta tecnología), etc. A continuación, se van a desglosar brevemente algunas formas de comunicación posible para ser usadas con estas placas y dependiendo de las necesidades de nuestro proyecto, usaremos una u otra, siendo siempre el más limitante la distancia.

#### **5.4.2.1   *Bluetooth***

En primer lugar, tratamos con la comunicación serie por bluetooth, por ser la más sencilla. Este tipo de conexiones son gestionadas mediante comandos de modem o AT. Para ello utilizamos como ejemplo el módulo HC-05, el cual tiene dos tipos de modos de funcionamiento llamados maestro o esclavo. El modo slave solo permite al módulo ser gestionado mediante otro dispositivo estando a la escucha de recibir órdenes, mientras que el master, permite al dispositivo también mandar instrucciones, lo que posibilita una comunicación bidireccional entre dos puntos.

Este tipo de módulos, suelen suministrarse soldados a una pequeña placa que adapta sus voltajes mediante un pequeño regulador a 3.3V ya que la gran mayoría de las soluciones mostradas suele trabajar a 5V, además de traer las conexiones necesarias para su gestión. Este módulo dispone de los siguientes pines:

- KEY: Este pin se utilizar para entrar en el modo de configuración para poder configurarlo mediante comandos AT. Para ello se debe establecer un nivel alto (+5V).
- RXD: Pin por el cual recibe la comunicación y hay que matizar que, para establecerla, siempre debe haber un cruce de pines entre el dispositivo que lo gestiona y el módulo bluetooth, por tanto, este pin iría conectado al pin TX.
- TXD: utilizado para enviar la comunicación serie.
- VCC y GND: se utilizan para alimentar al dispositivo.

Este tipo de dispositivos están organizados en tres clases que indican el rango de cobertura:

- Clase 3: 1 mW -> 0 dBm con un alcance máximo de ~1m
- Clase 2: 2.5 mW -> 4 dBm con un alcance máximo de ~5-10m
- Clase 1: 100 mW -> 20 dBm con un alcance de ~100M

Además, se clasifican también por versiones, las cuales indican régimen binario:

- Versión 1.2 con 1 Mbit/s de ancho de banda
- Versión 2.0 + EDR con 3 Mbit/s de ancho de banda
- Versión 3.0 + HS con 24 Mbit/s de ancho de banda
- Versión 4.0 con 32 Mbit/s de ancho de banda, está ya con los nuevos estándares BLE (Bluetooth low energy) de bajo consumo.

Como hemos podido ver, es una comunicación utilizada para distancias cortas, y que tiene una muy precaria seguridad, pero un punto a favor y es un bajísimo consumo con el estándar BLE. Como ejemplo, en la [\[fig23\]](#) se muestra el módulo maestro/esclavo HC-05.

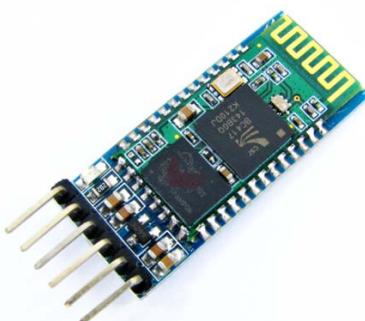


Figura 23: Módulo bluetooth HC-05, una iniciativa de comunicación inalámbrica a bajo precio.

#### 5.4.2.2 WiFi

Este tipo de comunicación, también se realiza mediante comandos de modem o AT. Para tratar de explicarlo haremos uso de uno de los módulos más famoso, el ESP8266. Es uno de los más baratos del mercado y es capaz de dar cobertura hasta una distancia aproximada de 20m en interiores y algo más en campo abierto. En la [\[fig24\]](#) se aprecia el montaje en protoboard de este módulo WiFi.

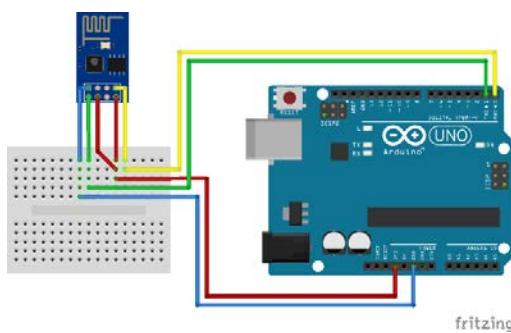


Figura 24: Módulo WiFi ESP8266, gestionado a través de comandos AT como el bluetooth.

Este tipo de módulo se alimentan a un voltaje de 3.3V al igual que el módulo bluetooth explicado anteriormente, y disponen del mismo de conexionado RX y TX solamente para gestionarlo. Hay poco que comentar de este tipo de comunicación, la cual trabaja como la anterior en la banda ISM (Industrial, científica y médica), pero esta dispone de una mayor seguridad en sus comunicaciones y despendiendo de la distancia, influirá su consumo al igual que bluetooth. Además, permite una comunicación full dúplex.

#### 5.4.2.3 RF

En el mercado, existen varios modelos de módulos de radiofrecuencia y con distintas características en su funcionamiento, donde lo habitual, es tener un emisor y un receptor, sin embargo, en este apartado vamos a tratar con el módulo NRF2401, el cual es un transceptor, esto quiere decir que es módulo puede tanto emitir como recibir, pero no al mismo tiempo lo que indica una comunicación dúplex, pero esto se puede modificar siempre que se quiera dándole unas sencillas instrucciones. A diferencia de las comunicaciones sin hilos anteriormente nombradas, este módulo se comunica con la placa mediante ISP, lo cual implica más pines de conexión y además el uso de una librería específica para él, al contrario que en los anteriores. Como las demás comunicaciones sin hilos nombradas, también es de bajo coste alimentado a 3.3V y

opera en la banda ISM a 2.4Ghz y posee una velocidad de comunicación de 250Kb/s, 1Mb/s o 2Mb/s configurables.

El alcance también es bastante pobre, entre ~20-80m determinado de la antena que posea, y si dispone de algún amplificador de señal, pero estirando estos factores un considerable aumento del consumo. A continuación, se muestra la [\[fig25\]](#) del módulo con su pinout, que tendrá unos pines distintos de conexión, dependiendo del módulo que se quiera usar.



Figura 25: Módulo RF NRF2401s, donde se indica su conexionado para conectarlo a la placa.

#### 6.4.2.4 GPRS

Existen en el mercado un gran número de módulos GSM/GPRS, pero aquí se va a tratar de explicar en base uno de los más sencillos y económicos, y el cual presenta gran facilidad de uso y manejo, y dispone de la menor cantidad de pines, solamente los utilizados para la gestión y alguno adicional más. En este caso se usa el módulo SIM800L el cual dispone de un abanico de funcionalidades totalmente sorprende, y el cual tiene una gran ventaja respecto a los otros y es el alcance debido a que la telefonía móvil tiene un gran auge en el presente y dispone de un gran número de antenas de gran potencia, lo cual nos da una gran cobertura a nivel mundial, siendo este el punto más atractivo de cara a la problemática del presente trabajo. Pasamos a detallar algunas características de este fantástico módulo:

- Voltaje de operación: 3.4V ~ 4.4V DC
- Consumo de corriente (max): 500 mA
- Consumo en modo sleep: 0.7 mA
- Interfaz serial UART
- Cuatribanda 850/900/1800/1900MHz a través de 2G
- Llamadas de voz y SMS
- Conexión de datos GPRS (TCP/IP, HTTP, FTP, etc.) a 85.6 Kbps

- Receptor FM
- Gestionado mediante comandos AT
- Reloj en tiempo real RTC
- Soporte A-GPS

Este módulo además de un pequeño consumo, dispone de un modo sueño configurable en modo automático, lo cual al terminar sus tareas se le indica que entre en este estado y una vez se necesite, solamente enviándole comandos de comunicación el despierta automáticamente. Su tamaño es muy reducido y tiene un bajísimo coste rondando los 5€.

Como hemos podido apreciar, este pequeño módulo dispone de unas grandísimas ventajas con respecto a sus competidores de comunicación sin hilos previamente explicados, lo que lo convierte en una de las mejores elecciones cuando se quiera dotar al proyecto de una comunicación inalámbrica a grandes distancias, con bajo consumo, bajo coste y simplicidad. La [\[fig25\]](#) ilustra el módulo SIM800L.



Figura 25: Módulo GPRS, SIM800L, utilizado para el envío de los datos al servidor web.

## 6 OBJETIVOS

Después de la problemática planteada en los objetivos y la pléyade de opciones mostradas en la sección de antecedentes, se pueden establecer unos objetivos principales que este trabajo debe cumplir para desarrollar un registrador de datos.

El primero es utilizar una plataforma de hardware y software libre que permita flexibilidad y robustez para adaptarse a los sistemas de generación de energía que se han expuesto en la introducción, pero, además adaptarse a toros de forma sencilla.

El segundo es optimizar el consumo de la solución elegida para el diseño del registrador, ya que su uso, en la mayoría de las ocasiones, tendrá lugar en lugares remotos y aislados, donde no habrá posibilidad de conexión eléctrica. Además, si se presenten medir sistemas fotovoltaicos autónomos de pequeña potencia, el sistema de medida no puede suponer una rémora energética para la instalación. Por ello un objetivo secundario sería el diseño de un sistema de alimentación autónomo para el registrador diseñado.

El tercero es que la toma de datos se envíe a distancia, para poder realizar un seguimiento de la instalación fuera de su ubicación original, al ser posible a tiempo real, además de contar con un sistema de copia de seguridad en una memoria, en el caso que la recepción de datos se pierda.

Enunciados los principales objetivos, a continuación, se exponen las soluciones adoptadas para lograr su cumplimiento, sin olvidar que el bajo coste es un concepto que está presente a lo largo de todo el manuscrito.

## 7 DESCRIPCIÓN DE LAS SOLUCIONES ADOPTADAS

En este apartado se realiza la exposición de las soluciones adoptadas. Se procede a presentar el análisis de acuerdo al orden que se ha seguido en los antecedentes. También se incluye un desarrollo del sistema de recogida, almacenamiento y envío de medidas. Este apartado estará compuesto por un conjunto de gráficas, medidas e imágenes sobre los datos obtenidos en cada una de las partes que componen este trabajo.

Para ello se presenta de forma esquemática el orden a seguir en el desarrollo:

- Equipamiento de medida, almacenamiento y envío de datos
  - Construcción del registrador de datos
  - Necesidad de toma de medidas
  - Circuito de medida
  - Puesta en hora del reloj
- Análisis de soluciones: Hardware
  - Diseño de bareboard
  - Implementación módulo GPRS
  - Diseño de SFA
- Análisis de soluciones: Software
  - Uso de librerías de bajo consumo y watchdog
  - Creación de aplicación de servidor para almacenamiento y procesado de datos y sincronización del reloj RTC
  - Presentación de los datos enviados

### 7.1 Equipamiento de medida, almacenamiento y envío de datos

Como en cualquier sistema, se considera esencial el uso de un sistema que pueda realizar lecturas de cualquier índole y almacenar esas lecturas para su posterior tratamiento. El sistema tiene que realizar dos funciones:

- Realización de medidas: Tomar lecturas es la parte más importante de este estudio,

y por ello deben ser de total confianza y siempre mantener precisión, rapidez y exactitud [1].

- Una vez realizada la toma de medias, tan importante es el almacenamiento de las mismas, con lo que se dispone a ello utilizando sistemas seguros, y a ser posibles en distintos lugares y formatos, para aumentar la posibilidad y flexibilidad a la hora de ser tratados y consultados.

Después de todas las opciones descritas en antecedentes, se ha visto que los registradores comerciales, no cumplían con todas las premisas de partida, ni cumplen los requisitos de trabajo, donde es de vital importancia, bajo consumo, autonomía, bajo coste, medidas fiables y precisas y la posibilidad de mandar los datos a largas distancias para realizar un monitorio constante de las mismas.

Otra posibilidad era el uso de plataformas comerciales de hardware libre para una posible implementación del registrador de datos usando alguna de ellas. De las cuatro mostradas, el Raspberry Pi, es una opción con alto consumo con lo cual queda totalmente descartada. Los tres restantes, tienen en común que comparten el microcontrolador el ATMega328p de la familia Atmel. A pesar de arrojar buenos resultados todos descritos a lo largo de los apartados contenidos en el apartado [5.3], estas disponen de varios componentes adicionales que, a pesar de disponer de disponer modos de bajo consumo, permanecen activos, con lo que aumentan el consumo final del trabajo, no siendo una opción viable a elegir.

Por último, se agregó la descripción del registrador de datos diseñado en el anterior trabajo [1], donde se pudo sacar como conclusión, que arrojó bastantes mejoras frente a la problemática de este trabajo, pero no llegaba a cumplir todas las premisas de este trabajo. A pesar de ello, se ha optado por reutilizar el anterior conversor ADC MCP3424, aunque realizando el diseño en una protoboard a modo de prototipo, donde se integran toda la batería de sensores y demás dispositivos del diseño general. Gracias al conocimiento de los buenos resultados conseguidos [1][23][29], que comprenden fiabilidad y exactitud, lo hacen único frente a otras soluciones comerciales.

### 7.1.1 *Construcción del registrador de datos*

Como se ha comentado con anterioridad, se quiere utilizar el mismo conversor ADC de datos empleado en los trabajos anteriores. Debido a esto, se van a necesitar los mismos componentes y para ello, se quiere construir otro que integre en una misma

placa, toda la batería de sensores, shields y el microcontrolador. Frente a todas las opciones vistas en antecedentes, como se comenta en el apartado [5.1], las soluciones que más se acercaban a solventar la problemática de este trabajo y pudiendo agregar en ellos todas las premisas de partida, son las plataformas comerciales de hardware libre, y como se pudo ver, todas ellas excepto Raspberry Pi, compartían el microcontrolador ATMega328p.

Como uno de los objetivos es ahorrar con el mayor grado posible en el consumo final, gracias a los detalles de cómo utilizar de manera funcional un microcontrolador de la familia Atmel usando para ello los mínimos componentes necesarios [ver apartado 14.2], se ha optado por usar un chip de esta familia usando esta técnica, ya que diseñando desde el principio el dispositivo, cumple todas las premisas de partida, y además, cumple con las expectativas de realizar un registrador de datos de bajo consumo y bajo coste.

Para el prototipo final del registrador de datos, se ha optado por utilizar el microcontrolador ATMega1284p debido a las siguientes características:

- En primer lugar, se ha tomado la decisión de elegir este chip por su tipo de encapsulamiento, siendo este tipo PDIP. Este tipo de encapsulamiento facilita su montaje tanto en placas protoboard tanto en una placa de circuito impreso para un diseño final, ya puede ser montado sobre un socket que facilite su montaje o desmontaje en caso de un posible fallo o rotura del mismo.
- La siguiente característica es su gran capacidad de memoria dinámica SRAM, lo que permite una mayor declaración de variables de uso global, y mejor manejo de instrucciones, ya que esta característica, es de vital importancia para el desempeño de las instrucciones programadas, con lo que tener gran cantidad, es un punto a favor.
- Memoria FLASH: debido a tener un gran número de dispositivos que implementar, por necesidad, aunque siempre tratando de realizar un código lo más óptimo posible, se necesita de un buen espacio para almacenar el código del programa, ya que, con una alta probabilidad, ocupara bastante espacio, donde es muy probable que los chips anteriores a este, no serán capaces de surtir las necesidades del proyecto.

- Interrupciones: Debido a los requerimientos en la declaración de 3 interrupciones hardware, dos para los sensores de lluvia y viento y la última y mayor importancia, la utilizada por el reloj DS3231, ya que es el encargado con sus alarmas, de despertar al microcontrolador de su estado de reposo, justo cuando se cumple la condición de medida, posibilitando que se dispare el proceso.
- Otro dato y uno de los objetivos a seguir, es el bajo coste, y este microcontrolador, se puede adquirir por unos 7 €, precio más que razonable, ya que dispone de unas grandes ventajas frente a sus competidores.

Para el montaje de los sensores digitales de temperatura (DS18B20) y humedad relativa (DHT22), el anemómetro Small Wind Transmitter, el pluviómetro Rain-O-Matic con salida de pulsos con relé reed, el reloj DS3231, el módulo GPRS SIM800L, tarjeta microSD y conversor ADC MCP3424 una placa de prototipado, exceptuando el microcontrolador ATMega1284p, que va montado en una placa de circuito impreso con los mínimos componentes para su funcionamiento, con la finalidad de evitar interferencia de ruido eléctrico o demás componentes del sistema. Teniendo en cuenta las pruebas realizadas, todo el diseño general podría ser contenido en una placa de circuito impreso, para tener un diseño más ordenado y ahorrar el cableado, lo cual queda más eficiente y mejor estéticamente, lo que también facilitaría su encapsulamiento en un compartimento estanco, para protegerlo de las condiciones climatológicas externas. Se adjunta en la [\[fig27\]](#) el montaje del prototipo final, realizado desde el software fritzing.

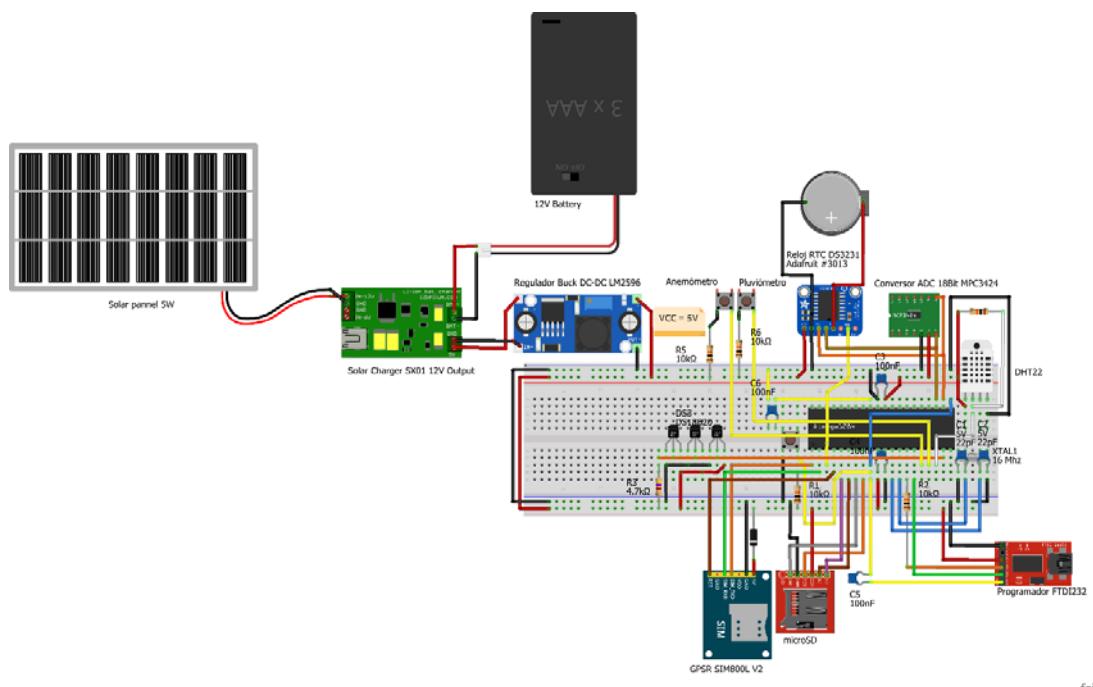


Figura 27: Prototipo del diseño final, utilizando una protoboard, mediante la herramienta fritzing

### **7.1.2 Puesta en hora del reloj**

Para concluir con la puesta a punto de nuestro dispositivo de media, debemos realizar la sincronización del reloj, ya que, por su cometido es de vital importancia y tanto como el corazón de este trabajo, el ATMega1284p. Esto es debido a que el reloj RTC DS3231, dispara el proceso de medida mediante unas interrupciones por hardware. Esto se realiza fijando dos alarmas internas, que son capaces de cambiar el estado del pin SQW/INT de alto a bajo. Una vez salta la interrupción debe ser desactivada para poder dispararse en la próxima alarma. Es muy importante previo a la fijación de las alarmas, que sea desactivada la salida SQW (square wave output), ya que por defecto esta salida genera una onda cuadrada de 1, 4096, 8192 y 32768Hz.

Es caso de quedar activa, estaría saltando constantemente la interrupción, lo que provocaría que ATMega1284p nunca entrara en SLEEP\_MODE\_PWR\_DOWN, que como se ha comentado anteriormente, solo es posible despertarlo mediante interrupción hardware, e interrupciones programadas por watchdog. Se ha observado a lo largo del test del diseño durante varios días, que el reloj perdía la sincronización de la hora, a pesar de tener la pila de 3.3V la cual evita esto.

Para solucionarlo y como una de las metas de bajo coste incluso no necesita la dependencia de la batería, debido a que se ha agregado al software la capacidad de sincronizar la hora del reloj desde internet, con una app PHP realizada en este proyecto, ubicada en el servidor. Este proceso de sincronización se ejecuta al inicio, con lo que logramos una regulación de la fecha y hora perfecta con la zona horaria en la que se encuentre ubicado nuestro circuito. En caso de cambiar de zona horaria, debe modificarse en la app PHP. Esto se consigue gracias al módulo GPRS SIM800L, que realiza una conexión HTTP y lee la página web como lo realizaría cualquier navegador, recogiendo la hora y entregándola al micro, después necesita ser procesada, ya que es recogida como tipo de dato string, y el reloj necesita tipo int para ser fijada.

## **7.2 Análisis de soluciones: Hardware**

Las soluciones hardware, están ligadas a cambios en el hardware de las placas comerciales, o en dispositivos que permitan reducir el consumo general. Como se ha podido esbozar en el apartado antecedentes, existen un gran número de formas para permitir el ahorro en el consumo energético.

En este trabajo, se ha utilizado dos de las tres expuestas, ya que como requisito se debe utilizar una placa diseñada con los mínimos componentes, la cual es en cuanto a nivel de hardware, la mejor solución orientada al ahorro en el consumo energético y, además, el cambio del regulador lineal que usan las placas comerciales, por otro que aumenta hasta casi en un 30% la eficiencia energética.

### 7.2.1 Sustitución regulador de voltaje

En este apartado se propone como solución hardware el cambio de regulador lineal utilizando como ejemplo la placa comercial de Arduino UNO R3. Su misión es la de proporcionar una salida de voltaje constante. Su gran desventaja, es que este tipo de reguladores, tienen muy poca eficiencia, siendo esta proporcional al porcentaje de voltaje salida/entrada, por ello podemos deducir, que a pesar de tener un rango de voltaje Vout de entre 1.5-12V, es necesario alimentarlo al menos con 2-3V por encima para suministrar los 5V en nuestro caso, si queremos dar corriente a la placa a través del conector jack, ya que si se decide suministrar la energía desde el puerto USB, no se usaría el regulador.

La ineficiencia energética de estos dispositivos, se debe a que ese voltaje de diferencia entre salida y entrada, se pierde en calor. En concreto el regulador de UNO, el AMS1117 5.0 [39] fig. [17], tiene un consumo de corriente en reposo de unos 10 mA, con un máximo de 0.8A de salida, suficientes para el conjunto de componentes que conforman la placa. A continuación, se muestra una figura ilustrativa del montaje básico del regulador montado en la placa Arduino UNO R3 [fig28], además del regulador que trae montado esta placa, en la [fig29].

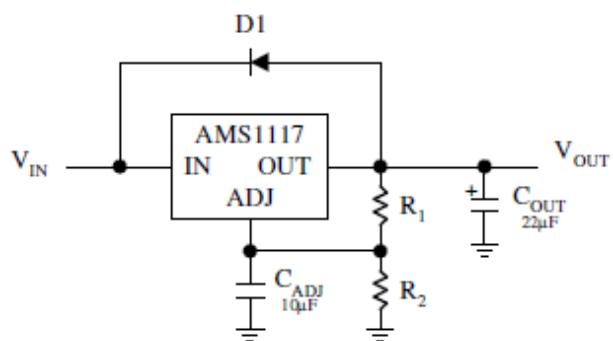


Figura 28: Montaje básico del regulador utilizado en la placa Arduino UNO R3, con R1 y R2 para fijar un voltaje, ya que este regulador es capaz con estas dos resistencias, sacar voltajes estables de 1.5V, 1.8V, 2.5V, 2.85V, 3.3V y 5.0V

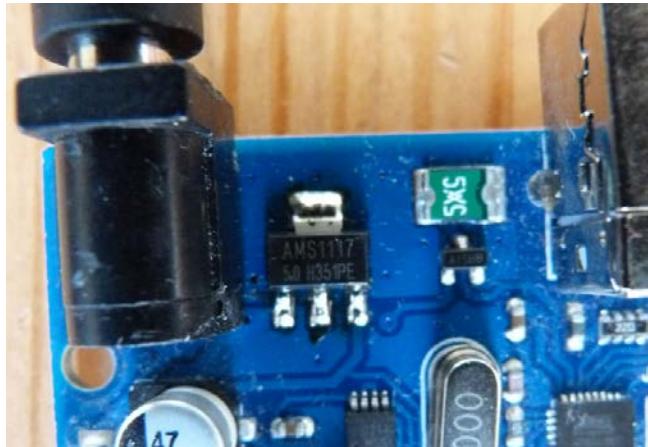


Figura 29: Regulador AMS1117 5.0, en placa arduino UNO R3.

Como opciones de sustitución, estarían el LM337, en su encapsulado D2PAK, con una corriente casi un 200% superior de 1A, y 0.4mA de corriente de reposo, o el LM317, con 1A de corriente de salida, y 1mA de corriente de reposo. Uno de los más indicados, por tener un voltaje de entrada comprendido entre 1.2-37V, y baja corriente de reposo sería el LM337 en su encapsulado SOT-223.

Se puede apreciar, que existen mejores alternativas al regulador que trae la placa comercial, pero puede variar en función del tipo de alimentación que queramos conectar. Esto indica que no se debería usar este tipo de reguladores en caso de querer conectar nuestra placa a fuentes de alimentación continua, como pueden ser una placa solar, o cualquier tipo de baterías. En este apartado también se puede comentar la posibilidad de deshacerse del regulador de voltaje de la placa, o quitándolo directamente [fig30], o cortando las pistas de salida y conectar una fuente de alimentación externa al conectar Jack que trae de serie la placa.

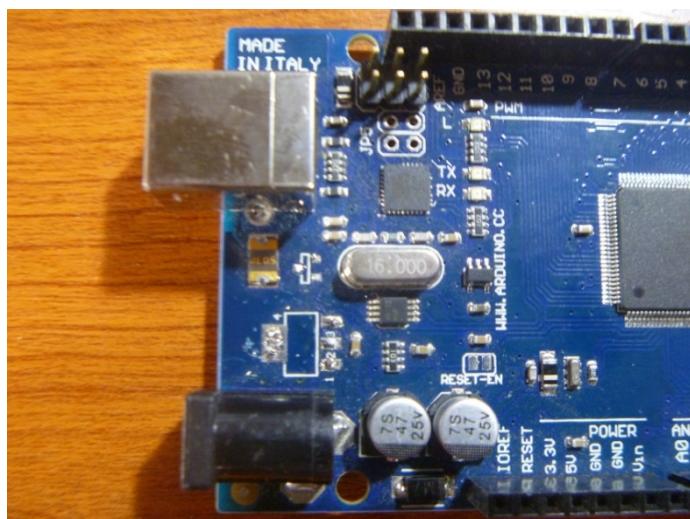


Figura 30: Regulador de voltaje eliminado de la placa.

Hay dos opciones bastante más idóneas desde este conector y es la posibilidad de usar una fuente de alimentación conmutada, con un 90% de eficiencia y cortando la corriente si no hay conducción de energía. La [\[fig31\]](#) ilustra el diseño de una fuente de alimentación conmutada.

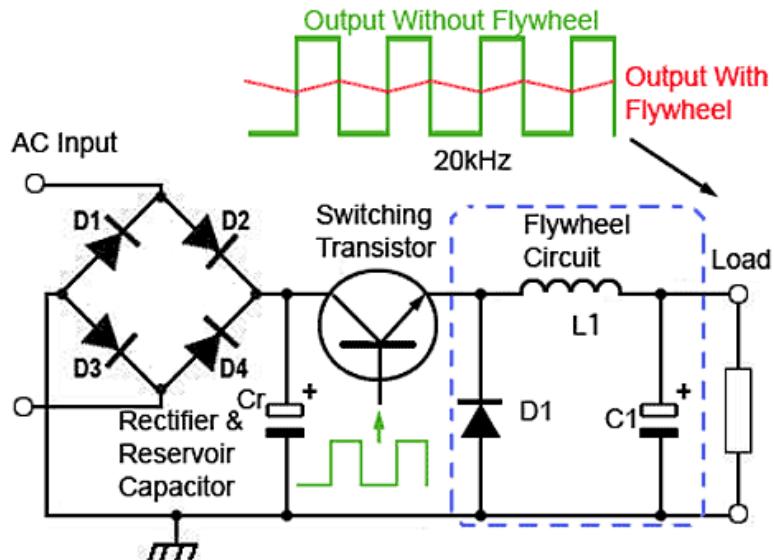


Figura 31: Fuentes de alimentación conmutadas, donde es posible regular el voltaje de salida aplicando una señal cuadrada a la entrada como la que podemos encontrar en las salidas PWM de varias placas de hardware libre.

En caso de querer usar fuentes DC, se puede usar un conversor DC-DC tipo step up/down, en función del voltaje a conectar. Los tipo step up, aumentan el voltaje con respecto al voltaje de entrada, al contrario que los step down, que lo reducen. Un regulador a destacar por su gran versatilidad, fiabilidad, eficiencia y como no precio, es el 2596 [\[10\]](#), donde su mayor ventaja, con respecto a los anteriormente nombrados, es su eficiencia energética de hasta un 95%. A continuación, se muestra en la [\[fig31\]](#) de un circuito regulador DC-DC Buck (step down o reductor de tensión) que es posible adquirir por tan solo 1€.



Figura 32: Regulador de voltaje Buck DC-DC 2596, ideal cuando queremos alimentar un diseño con fuentes de corriente continua.

### 7.2.2 *Diseño de bareboard*

En este apartado se explica con detalle, como se realiza el diseño de una placa funcional con el microcontrolador ATMega1284p, utilizado en nuestro diseño, para trabajar con el mínimo de componentes necesario. Para ello primero es necesario saber esa lista de componentes. Como es un prototipo, se decide montar en placa protoboard para una facilidad ante las futuras pruebas. Para el diseño se necesitan, una resistencia de  $10\text{K}\Omega$  y  $1/4\text{W}$ , para utilizarla en la patilla 9 de RESET, donde se encuentra en configuración pull-up, lo que indica que irá conectada a VCC, que, en este caso, ya que todos los dispositivos del sistema funcionan a 5V, será de este voltaje, aunque según el datasheet del micro, podría funcionar hasta un voltaje mínimo de 1.8V, lo que reduce más el consumo general, pero no siendo posible por lo que se acaba de comentar.

Además, se necesitan 2 condensadores de poliéster de  $0.1\mu\text{F}$ , utilizados a ambos lados del micro, conectados entre los pines de VCC y GND, y AVCC y GND, usados a modo de desacople. Dos condensadores de 18 o 22 pF, conectados en cada patilla del cristal de 16MHz y GND de tipo lenteja, y el cristal de 16MHz. En este caso se usa un cristal con dicha frecuencia debido a que el bootloader usado, el “optiboot”, no puede trabajar a mayor frecuencia, pero es capaz de utilizar un cristal de hasta 20MHz. Como apunte, hay que decir que es posible usar este cristal, lo que aumentaría la velocidad para realizar las tareas de nuestro dispositivo, pero no es necesario, puesto que no se utilizan rutinas demasiado complejas como para demandar más velocidad.

En caso de querer usarlo, no podría utilizar el bootloader, lo que conlleva utilizar un programador por ICSP o ISP, ya que el encargado de escuchar la comunicación USB-TTL entre el PC y el micro, colocando el programa que se graba en la FLASH a continuación de él, y disparando el inicio una vez que termina su escritura, o en caso de reinicio realiza la misma función.

En este trabajo por comodidad, se ha optado por utilizar un cristal de 16MHz y el bootloader optiboot, para grabar con mayor rapidez el programa mientras se implementaba el código de los distintos dispositivos del proyecto. A continuación, se muestra la [\[fig33\]](#) de cómo quedaría el diseño de la bareboard, montada en una placa de prototipado:

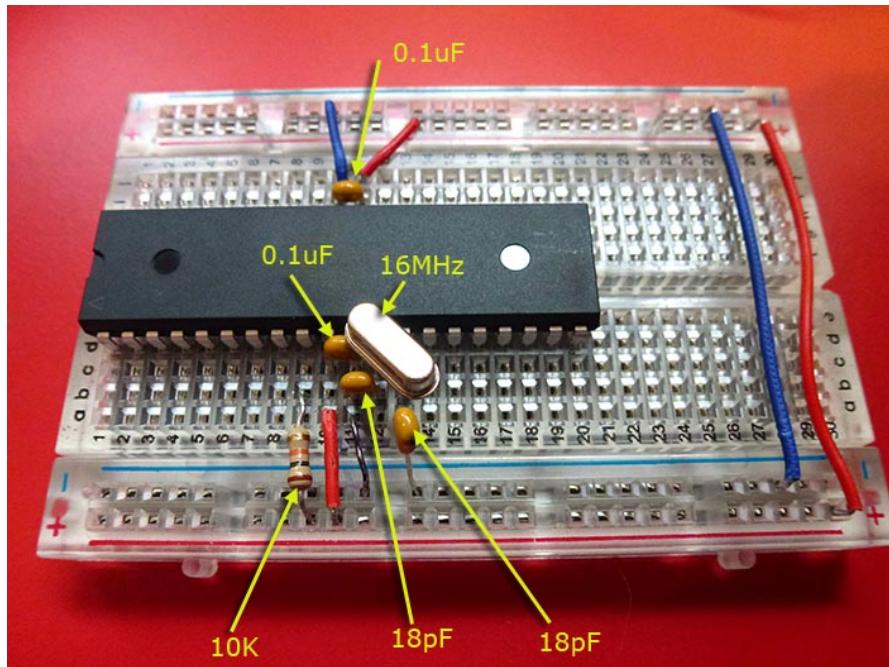


Figura 33: Microcontrolador ATMega1284p, montado con los mínimos componentes necesarios para su funcionamiento.

Una vez realizado su montaje, hay que grabar el bootloader necesario para poder descargar el código a la memoria FLASH del microcontrolador. Para ello lo primero es agregar la compatibilidad de la placa con el entorno de desarrollo y utilizar un programador ISP para escribir el bootloader, o como se propone en dicha web, el uso de una segunda placa Arduino, programada como programador ISP. Este procedimiento está detallado en antecedentes en el apartado [14.2]. Una vez agregada la compatibilidad, en este caso, en la web adjunta en la bibliografía “Atmega bootloader programmer”[30], debemos descargar unas librerías específicas para la programación del ATMega1284p, ya a través de ellas, modificar el siguiente apartado del código del sketch “Atmega\_board\_programmer”:

```
0xFF,      // fuse low byte: external clock, max start-up time
0xDE,      // fuse high byte: SPI enable, boot into bootloader, 1024 byte bootloader
0xFD,      // fuse extended byte: brown-out detection at 2.7V
```

Por el siguiente:

```
0xF7,      // fuse low byte: external clock, max start-up time
0xDE,      // fuse high byte: SPI enable, boot into bootloader, 1024 byte bootloader
0xFD,      // fuse extended byte: brown-out detection at 2.7V
```

Esto es debido a que fuse low byte (fusible de configuración de estado bajo), tiene una mala configuración, lo que imposibilita la descarga del software a la FLASH. Una vez realizado este procedimiento, y realizado el conexionado, se abre la consola Serial,

donde se indica la opción de realizar el grabado del bootloader. El proceso tarda apenas un segundo, y una vez terminado, tendremos listo nuestro ATMega1284p para escribir el software en él. Antes de terminar este apartado, es necesario decir que, al intentar escribir software en el microcontrolador, se presentaron problemas en la escritura debido a interferencias, desembocando en un fallo ya que terminaba el proceso de escritura de manera premeditada.

Para solventarlo, y ya que en este trabajo se usa como opción de programador uno a través de USB-TTL, el FTDI232, es necesario agregar una resistencia de 10KΩ en serie con la patilla TX programador y RX micro (hay que recordar que para la comunicación Serial, es necesario cruzan los pines RX-TX), y un condensador en serie de 0,1 µF a la patilla DTR del programador, y RESET del micro, encargada de realizar un reset al MCU una vez descargado el software en la FLASH. A continuación, se adjuntan dos fotografías, una del programador utilizado [fig34], y otra del montaje con el programador [fig35], listo para descarga cualquier código en la memoria:

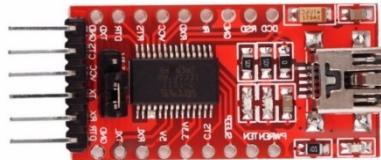


Figura 34: Programador FTDI232, utilizado para grabar los sketch en la FLASH del ATMega1284p a través de USB-TTL.

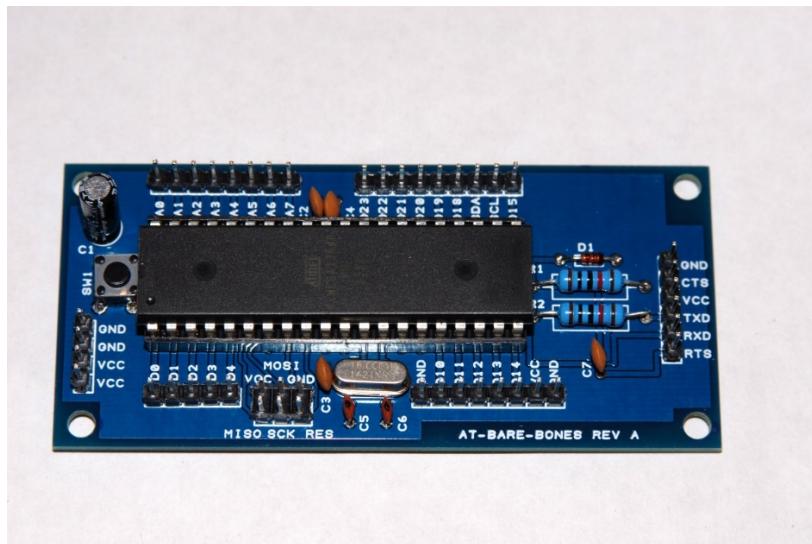


Figura 35: ATmega1284p montado en un socket sobre una placa de circuito impreso con los mínimos componentes necesarios.

A pesar del posible uso de la protoboard para el montaje del micro con los mínimos componentes, para evitar el posible ruido generado por el resto de componentes del circuito, se ha decidido por realizar una placa de montaje superficial, que además de evitar estas interferencias, le da un uso más sencillo, debido al uso de pines para un conexiónado con cableado, y un uso más intuitivo, ya que como en las placas comerciales, están numerados y nombradas todo su patillaje. Una vez terminado este proceso, está preparado para recibir cualquier sketch del programador y listo para comenzar con el diseño del circuito registrador de datos y posterior envío de los mismos.

### 7.2.3 *Implantación módulo GPRS*

Debido a la imperiosa necesidad de disponer de los datos recogidos por el registrador, y visto en los antecedentes la diferentes soluciones para el envío de los mismos desde ATMega1284p hasta internet, donde se hace más cómodo el tratamiento y visionado mediante gráficos, se ha optado por el uso de una shield GSM-GPRS, en concreto el módulo con chipset SIM800L de la compañía SIMCOM. Para justificar su uso, se exponen los distintos argumentos para escoger esta opción en lugar de las restantes expuestas.

- En primer lugar, quedan descartadas la comunicación con hilos, ya que por la situación geográfica donde se implantará este trabajo, lejos de cualquier núcleo urbano, debido a que la ubicación de los campos solares, está lejos de los mismos, la comodidad ofrecida por las comunicaciones inalámbricas y en el ahorro en infraestructura, todos los cables que unen el equipo receptor y transmisor, perdida de señal cuando los cables tienen grandes distancias, lo que implicaría instrumental para regeneración de señal hacen que se descarte totalmente este tipo de comunicaciones.
- Dentro de las comunicaciones inalámbricas siguiendo el orden de los antecedentes, aunque el bluetooth en su última generación, dispone de tecnologías de un ultra bajo consumo, no las hacen viables para este proyecto, como consecuencia de una baja seguridad y poca distancia en la comunicación. Las demás tecnologías inalámbricas, se han descartado por la sencilla razón de la distancia a la cual es posible realizar la comunicación, dando lugar a grandes equipos para mandar grandes potencias para abarcar una mayor zona de cobertura.

- Otra desventaja es que, para estas soluciones inalámbricas, se hace necesario el uso de un equipo receptor y transmisor, donde el transmisor ya tendría acceso a internet donde se enviarían los datos al server de destino. Esto conlleva un mayor desembolso económico, cosa que está fuera de nuestro objetivo de bajo coste.

Por los motivos expuestos, se toma la decisión del uso de la comunicación por GPRS debido a que, con poca potencia de emisión, podemos enviar nuestros datos, ya que existe una gran cobertura de telefonía móvil a nivel mundial, siendo solo necesario el uso del equipo transmisor, siendo en este caso el módulo SIM800L pudiendo colocarlo en cualquier parte del mundo.

Expuestos los motivos de nuestra opción, hemos podido apreciar las grandes ventajas que brinda la comunicación de telefonía móvil y por ello, será la elegida para nuestro proyecto. Hay que tener precaución a la hora de gestionar los tiempos de respuesta después de ordenarle una instrucción mediante comandos AT al módulo GPRS, ya que, si no son escuchados y se continúa mandando instrucciones, puede dar lugar a una saturación del buffer de datos, dando lugar a errores en la comunicación.

#### *7.2.4 Diseño de SFA (Sistemas Fotovoltaico Autónomo)*

Un sistema fotovoltaico autónomo o aislado (SFA) [\[35\]](#), convierte la energía proveniente del sol en energía eléctrica, almacenándola en una batería para su posterior uso. Este tipo de instalaciones no requieren de una conexión a la red eléctrica general, trabajando de forma autónoma para proveer energía a los equipos. Estos sistemas están diseñados para adaptarse con facilidad a lugares sin conexión a la red, donde hay un bajo consumo de energía y un buen recurso solar.

Estos sistemas son ideales para aplicaciones donde se requieran pocos recursos, o en cuyo caso de una mayor demanda energética, se suelen combinar con otros sistemas como generadores eléctricos con motor de combustión interna, turbinas eólicas o la combinación con la red eléctrica mediante inversores de corriente que transformar la corriente DC generada por los módulos solares, en corriente alterna. Este tipo de sistemas se llaman sistemas híbridos, y son para los cuales, se ha desarrollado este trabajo, para la monitorización de estos campos solares. Los sistemas híbridos tienen la ventaja de reducir el coste inicial de la inversión y asegurar un suministro de energía constante.

En su forma básica un SFA consiste en:

- Generador: el módulo fotovoltaico que convierte la radiación solar en energía eléctrica.
- Acumulador: la batería donde se almacena la energía proveniente del módulo.
- Regulador: un elemento que controla la energía producida por el módulo, la carga de la batería, y la energía consumida en los equipos.
- Carga: los equipos que consumen la energía, en este trabajo se colocará el regulador DC-DC Buck LM2596 [\[10\]](#), el cual alimenta al circuito que actúa en este caso, junto con el regulador como carga.

Nuestro SFA, será exclusivamente de corriente directa proveniente del módulo de 5W policristalino, y almacenada en una batería. A continuación, se muestra en la [\[fig36\]](#) el diseño necesario para sistemas donde es posible el uso de corriente continua y alterna:

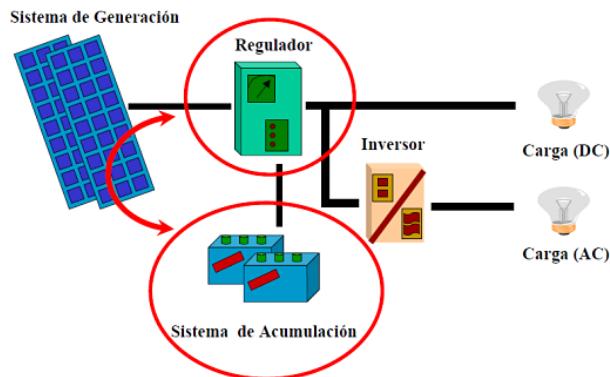


Figura 36: Sistema fotovoltaico autónomo preparado para abastecer a circuitos de DC-AC con la ayuda de un inversor para este último.

En nuestro caso no será necesario inversor, ya que nuestro diseño, se alimenta exclusivamente con corriente continua. Estos sistemas son capaces de suministrar energía durante años de forma eficaz y económica. Sin embargo, se hace de vital importancia realizar un correcto dimensionamiento del sistema para una larga vida útil. Con esto vamos a procurar a la hora de realizar el diseño seguir unas pautas para ello:

- Sencillez: se buscará la solución más sencilla para alcanzar nuestro objetivo
- Planificación: si se dedica tiempo, se tendrá más probabilidad de éxito

- Bajo coste: ya que es una de nuestras metas, donde debemos poner en una balanza calidad y precio.

Una vez realizado una pequeña introducción sobre estos sistemas, se procede a el cálculo de la instalación.

#### *7.2.4.1 Cálculo de la instalación (Dimensionamiento)*

Para el diseño del mismo, como se ha comentado en el apartado anterior, es necesario de tres dispositivos, el módulo solar, el regulador de carga y una batería que almacene la energía que suministra el módulo. Hay que comentar que el regulador de carga solar, puede ser construido, pero como la finalidad de este trabajo, es bajo coste, en base a los ejemplos de circuitos posibles para su fabricación [\[ejemplo\]](#), haciendo un evaluación del diseño, ajustarse a nuestras necesidades, ya que según el datasheet del módulo GPRS, componente que más consumo demanda en nuestro circuito, puede tener picos de consumo de hasta 2A, sin embargo, en las mediciones de consumo realizadas hemos podido comprobar a lo largo de una tanda de 98 mediciones seguidas, no ha tenido picos mayores de 380mA, por lo que por seguridad, para ajustar el regulador a nuestras necesidades, se intenta buscar un dispositivo que pueda entregar una corriente de 3A, como el que se ha decidido usar. Además, el SX01 [\[11\]](#), produce un consumo de 2mA en vacío, y no superior a los 8-10mA en funcionamiento haciéndolo idóneo para este trabajo.

Otra necesidad implícita tomando la opción de un diseño propio, es que debe tener una monitorización de los tres dispositivos que tiene conectados, módulo solar, batería y carga, necesitando de un PIC o ATMega que haga de lógica de control tomando las medidas de corriente y voltaje necesarias para llevar una adecuada gestión, y con una buena eficiencia. Para hacer esto se necesita de un programa que pueda llevar todo el control, lo que implica un mayor tiempo.

Todas estas necesidades son cubiertas por este regulador, donde además posee de un display de 7 segmentos que nos da información del estado de los dispositivos conectados con la posibilidad de ser apagado cuando no está en uso, diferentes modos de funcionamiento, compatibilidad con un gran número de baterías, regulación de carga mediante PWM (modulación por ancho de pulso) lo cual ayuda a regular la intensidad de carga de la batería hasta su carga total y su voltaje, ayudando a una menor pérdida de

energía y evitando sobrecalentamientos, etc, son todas las posibilidades que nos puede ofrecer este dispositivo, además de su reducido precio, es lo que ha llevado a decantarse por la opción del regulador comercial SX01.

Una vez aclarado por qué sobre la decisión tomada, se procede a detallar el cálculo para el dimensionamiento de nuestro SFA, para obtener el número de módulos y tipo de batería que se deberá usar.

- En primer lugar, es necesario hacer un estudio del consumo demandado por nuestro diseño. Este se llevó a cabo en el apartado [8.3], por tanto, se aconseja dirigirse aquí para mayor detalle. En este estudio, se pudo determinar después de una tanda de 98 medias sacadas de 98 procesos completos de medida (recordar que en cada proceso se toman 300 medidas de corriente), se hizo la media global, donde el consumo estimado del circuito en cada ciclo de medida y sueño, ha sido de 36 mAh, con lo que este será nuestro dato de partida, junto con el conocimiento de que el voltaje de la batería y el regulador de cargar solar serán de 12V y el de nuestro circuito de 5V.
- El siguiente paso, será calcular los W/h día que consumirá el diseño, para calcular la batería necesaria, partiendo de que el tipo de batería es VRLA (batería de ácido-plomo regulada por válvula) con un voltaje de 12V. Para realizar el siguiente cálculo, habrá que tener en cuenta las pérdidas de conversión generadas por el regulador Buck LM2596, que son del 20%, lo que nos da un 80% de eficiencia. Por tanto:

$$36mAh * 5V * 1.2(\text{factor de pérdidas}) = 0.216 Wh$$

Esta potencia nos da cuenta de la potencia que consume con exactitud a la hora nuestro circuito, sin embargo, se debe tomar el peor caso, sobredimensionando por seguridad en su autonomía:

$$36mAh * 12V = 0.432 Wh$$

Este dato obtenido, será el de partida para comenzar con el cálculo del SFA, donde en este caso para el cálculo, será redondeado a 0.43 Wh. Se calcula el consumo en Wh/día, por tanto:

$$0.43Wh * 24h = 10.32 \frac{Wh}{día}$$

- Según la normativa, se recomienda dotar al sistema de una autonomía de entre 3 – 5 días, donde en este caso se opta por 4 lo que nos da:

$$4días * 10.32 \frac{Wh}{día} = 41.28 Wh$$

Una vez obtenidos los vatios necesarios para cubrir la demanda, procedemos al cálculo de la batería. Para esto es necesario comentar el DOD (Depth of discharge) o máxima profundidad de descarga, siendo este el máximo de descarga permitido de la batería sin causarle daños a su vida útil. Como en la universidad disponemos de 4 posibles modelos para usar en esta instalación todas disponen de una DOD máxima de un 65%, lo que deja un 35% de su máximo a lo largo de esos 4 días.

Otro dato a conocer el factor de rendimiento carga/descarga, siendo este el voltaje mínimo al que puede llegar con respecto al voltaje pico de la batería que en este tipo de batería de 12V suele llegar a unos 14.2V. Esto nos indica que la batería debe comenzar su carga y cortar el suministro en caso de estar entregándolo al circuito una vez llega a  $14.2V * 0.86(\text{factor de carga/descarga}) = 12.212V$ . El margen de entre 12.212V-14.2V será el que debe ajustarse en nuestro regulador de carga solar, para que actúe y cargue la batería. Con esto tenemos los datos necesarios para saber su amperaje:

$$\frac{41.28Wh}{0.65(DOD) * 0.86(\text{factor de redimiento carga/decarga})} \approx 73.85 Wh$$

Donde como sabemos nuestra batería es de 12V, por tanto:

$$\frac{73.85Wh}{12V} \approx 6.16 Ah$$

Finalmente se obtiene los Ah que debe tener nuestra batería. Como siempre por seguridad, se debe tomar un valor un poco más alto, y para esto, se elige entre las distintas baterías proporcionadas por la Escuela Politécnica superior de Linares, la

batería de plomo-ácido de ciclo profundo de 12V-7,2Ah, AGM. Con esto se concluye el cálculo de la batería necesaria.

- El siguiente objetivo es el cálculo de la cantidad de módulos solares necesarios, donde partimos que todos los disponibles en la Escuela Politécnica Superior de Linares son de 5W de potencia nominal policristalinos. Para realizar el cálculo, es importante elegir la zona geográfica donde se colocará el diseño, ya que se debe conocer la radiación solar dado en kWh/m<sup>2</sup>/día necesario para la obtención del HSP (hora solar pico). Esto nos indica el número de horas de sol en media diaria a una radiación de 1000 W/m<sup>2</sup>, equivalente a la energía total diaria incidente sobre una superficie horizontal en kWh/m<sup>2</sup>/día. Este dato, se obtiene para el peor caso, y en este caso, valga la redundancia, se obtiene para el “mes peor”, donde para Linares se estima 3.5 HSP. Con estos datos solo queda conocer las pérdidas globales del módulo solar disponible, que en este caso es de 25%. Por tanto, ya se puede calcular el número de módulos necesarios para cubrir las necesidades del diseño:

$$Nº(\text{módulos}) = \frac{10.32Wh}{5W * 3.5HSP * 0.75(\text{pérdidas globales})} \approx 0.79$$

Lo que nos indica que solo es necesario de un módulo para cubrir la autonomía del circuito diseño.

### 7.3 Análisis de soluciones: Software

En este apartado de exponen las soluciones software o modificaciones a nivel de código para obtener el objetivo de mejorar el rendimiento energético de nuestro registrador sin la necesidad de usar hardware adicional. Es probable el paralelismo de ciertas soluciones software con el hardware, ya que como siempre, deben ir de la mano para conseguir el mejor rendimiento. Hay que aclarar que no todas las soluciones software expuestas en los antecedentes, han sido utilizadas, debido que en su lugar se han sustituido por otras opciones que se ajustan mejor al perfil que sigue el funcionamiento general del dispositivo.

### *7.3.1 Soluciones no contempladas*

A continuación, se cree conveniente la explicación del por qué no usar todas las soluciones comentadas en los Antecedente:

- La utilización de timers y sus registros queda omitida debido al uso de los modos sleep de ATMega. Esto se debe a que, en los modos de bajo consumo, en concreto el SLEEP\_MODE\_PWR\_DOWN, se realiza la desconexión de la mayor parte de componentes integrados en el chip, quedando solo a la escucha de las interrupciones hardware y el watchdog timer. Debido a esto, se hace imposible el uso de timers y registros por quedar inactivos durante este modo sleep.
- La utilización de reducción de la frecuencia también queda descartada, debido a que, en la realización a las pruebas pertinentes, se ha observado una mayor lentitud en la realización de las tareas. Esta opción queda más orientada en la realización de tareas de gran sencillez, que no requieran de una gran rapidez de procesamiento. En este caso, es preferible realizar las tareas con la mayor brevedad posible, ya que el mayor modo de ahorro energético vía software, se obtiene colocando nuestro microcontrolador en modo SLEEP\_MODE\_PWR\_DOWN, por lo que se pretende tomar las medidas de los sensores y su posterior envío en el menor tiempo posible, para el resto del tiempo hasta la siguiente tanda de medidas, el micro este en modo bajo consumo.

Aclarado el porqué de no adoptar todas las posibles soluciones descritas en el apartado de antecedentes, se dispone a mostrar las soluciones adoptadas para la recogida de medias y ahorro en el consumo energético vía software.

### *7.3.2 Software del registrador de datos*

El software desarrollado abarca todas las partes del proyecto, excepto la toma de medidas de corriente, que se realiza desde una segunda placa de arduino, por el motivo de que mientras el microcontrolador se encuentra en modo sleep, no podría monitorizar la corriente que está consumiendo en esos momentos. Este software, sincroniza la hora desde internet, inicializa y establece la conexión de datos desde el módulo GPRS. Una vez fijada, queda listo para empezar la tanda de medidas, que compone la lectura de la hora y los sensores, el grabando de los datos en la microSD, la lectura de los mismos y su posterior envío al servidor web. Este proceso por normativa se realiza cada 30 segundos. A continuación, se adjunta un flujograma de la [fig37], donde se muestra el funcionamiento del software general.

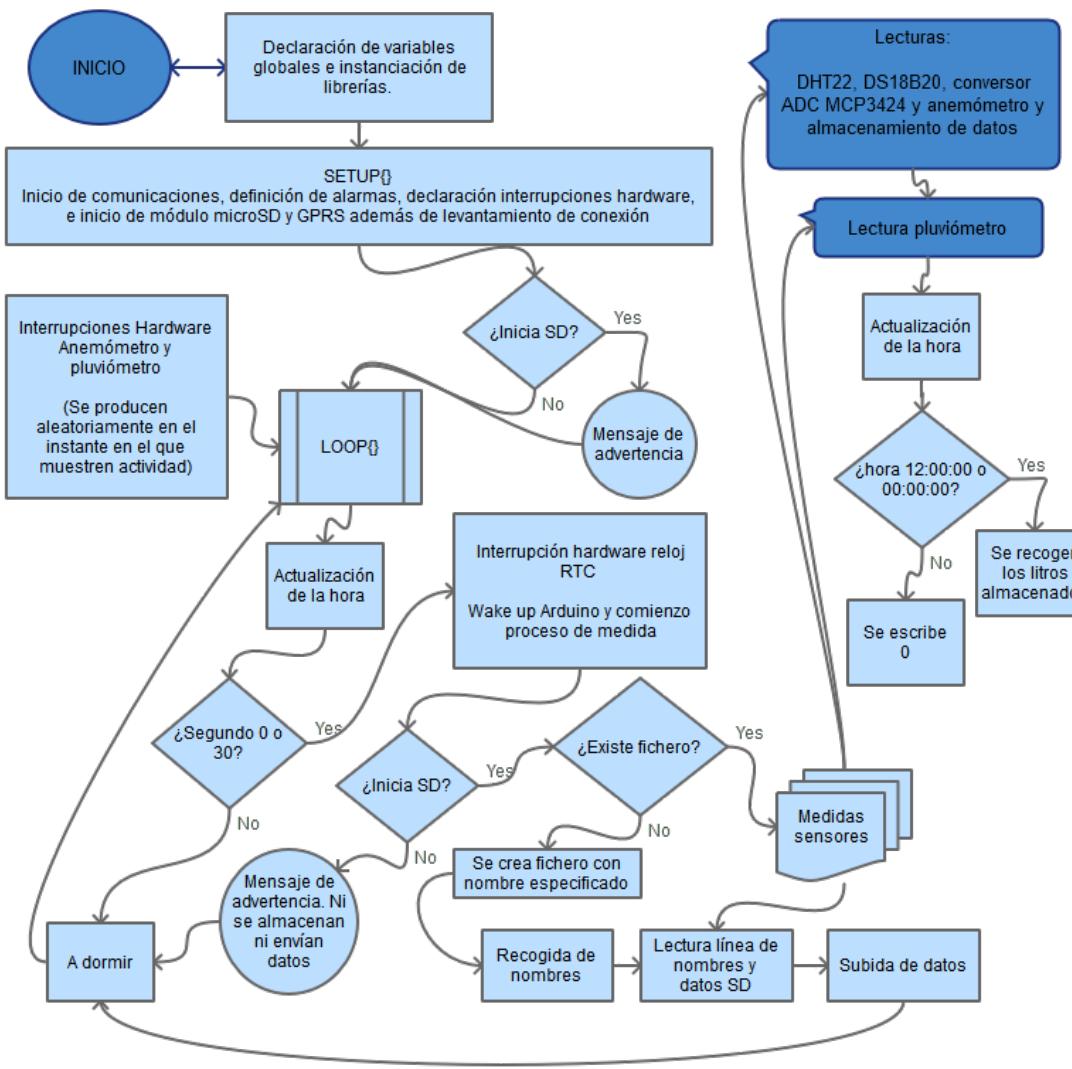


Figura 37: Diagrama de flujo del software de todo el proyecto, muy orientativo para una comprensión visual del mismo.

### 7.3.4 Creación de aplicación de servidor para almacenamiento y procesado de datos y sincronización del reloj RTC

Esta aplicación ha sido creada con el fin solucionar una problemática expuesta en los antecedentes, de disponer de los mismos datos recogidos por nuestro circuito, con lo que se crea una copia de seguridad idéntica, lo que además posibilita con una mayor sencillez, el tratamiento de los datos, como por ejemplo para ser representados de forma automática en gráficas y poder llevar una monitorización en tiempo real desde cualquier parte del mundo.

Para comenzar con la explicación, hay que comentar que el servidor que aloja nuestras aplicaciones programadas en PHP, puede ser montado por el propio usuario,

pero este caso se ha elegido optar de un hosting gratuito para evitar la instalación y configuración del servidor. Aun tomando la opción más sencilla y rápida, se describe brevemente, el proceso a seguir en caso de querer montar un servidor propio.

Una posible opción es utilizar el software xampp o wamp, los cuales alojan soporte para web, ftp, php, base de datos, etc. Una vez instalado, para poder acceder desde internet a nuestro servidor, y en el caso de este trabajo también desde el módulo GPRS, sería necesario disponer de una IP pública estática suministrada por el proveedor ISP que se tenga contratado, donde este servicio tiene un coste adicional.

En caso de como la mayor parte de la comunidad, se dispone de IP pública dinámica. Para esto es necesario del uso de servicios de dns dinámico como Nolp, y para ello se necesita de un software adicional suministrado por dicha web, o muchos router ya cuentan con la opción de DDNS, que en todo momento estaría facilitando a esta web, la cual suministra un dns, la ip del router, para poder llegar al server. Además, la ip de la máquina que contiene el servidor, debe ser fija, y realizar un forwarding, para reenviar las peticiones realizadas al router por el puerto 80, y estas se redirijan a la máquina que aloja el servidor, posibilitando después de todas estas configuraciones, el acceso al mismo y las aplicaciones en PHP.

Para evitar todo este proceso, se toma la opción de registrarse en un hosting gratuito, que en este caso ha sido Hostinger, el cual da un soporte más que suficiente y de calidad, para las tareas sencillas que queremos desempeñar. Una vez realizado el registro y elegido el nombre de nuestra web, solo queda copiar las aplicaciones desarrolladas en su carpeta raíz para comenzar a utilizarlas.

#### *7.3.4.1 Aplicación de procesado de datos*

El propósito de esta aplicación, es la de generar un fichero idéntico al que se está generando en la microSD del circuito, para después estos datos enviarlos a una página web que nos permitirá almacenarlos en una base de datos y poder generar gráficas para su monitorización en tiempo real. Esta aplicación es sencilla y realiza pocas tareas, con lo que, para una rápida explicación, se adjunta un diagrama de flujo en la [\[fig38\]](#) donde se detallan las operaciones que realizan este pequeño software. El lenguaje elegido ha sido PHP.

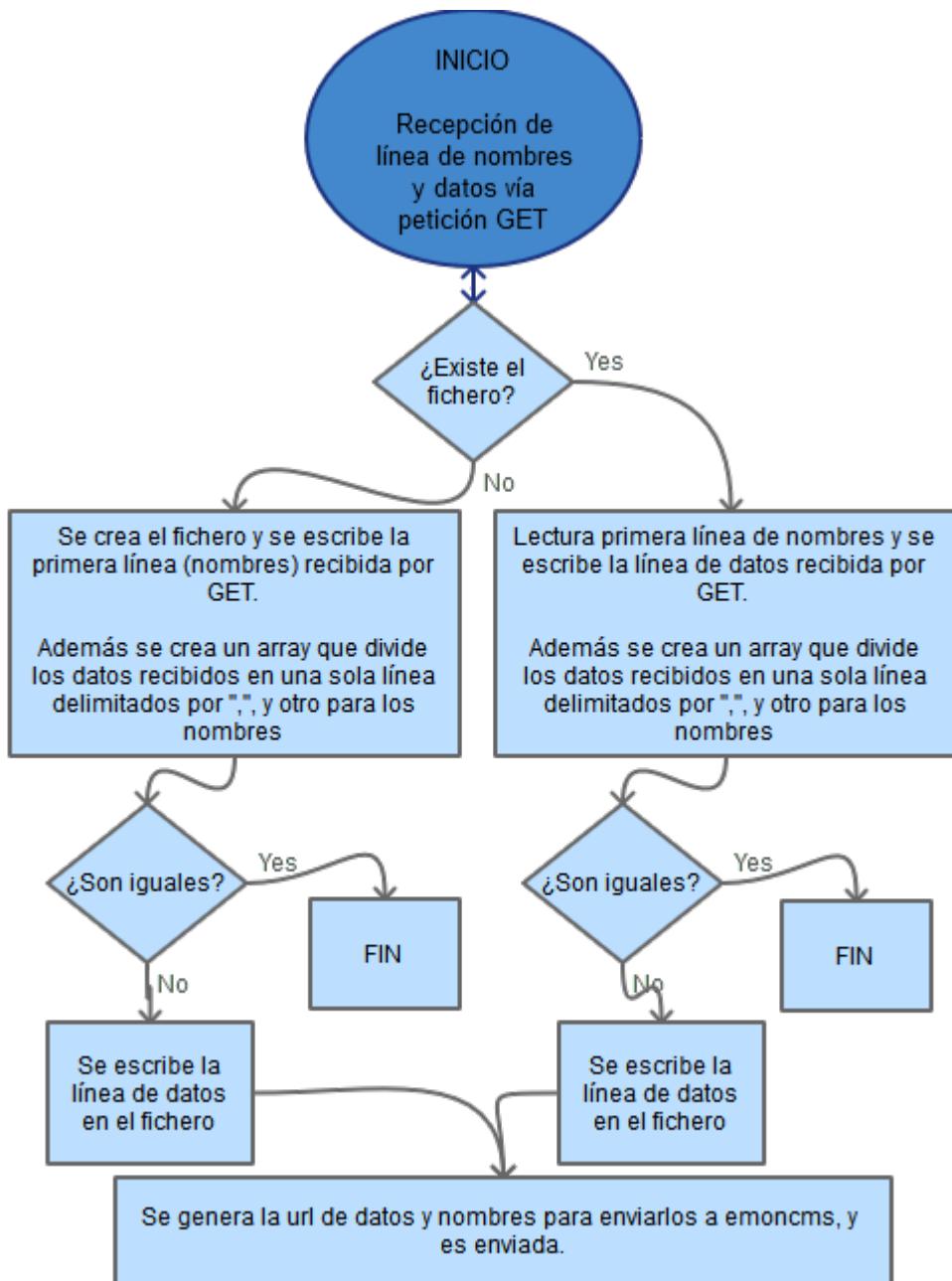


Figura 38: Diagrama de flujo del software PHP que recoge y escribe los datos y los reenvía a la web de emonCMS, para realizar las gráficas.

Como se puede apreciar, el software es básico. La primera decisión tiene en cuenta todas las posibilidades, las cuales son:

- En primer lugar, puede no existir el fichero en nuestra microSD, lo que la línea de nombres y datos enviada sería la misma. Si en esta situación en el server existe, las compara y al ser iguales, no escribe ningún dato y además termina la ejecución.

- Otra opción es que no exista en el servidor y si en la tarjeta, lo que provoca la creación del fichero y la escritura de las dos líneas en él.
- Por último, existe la posibilidad de la primera ejecución en las dos partes, por lo que no existiría el fichero en ninguna de las dos. Esto crea el fichero en la tarjeta microSD, y la línea de nombres y datos sería la misma con lo que, en el servidor, generaría el fichero, también escribiría la línea de nombres y acabaría la ejecución.

Una vez esta decisión, se genera la URL que contiene los datos y nombres que son entregados a emoncms mediante URL para una vez almacenados allí en su base datos, poder generar los gráficos que nos permiten verlos.

#### *7.3.4.2 Aplicación para sincronización del reloj RTC*

Esta aplicación no precisa de una gran explicación, ya que simplemente sincroniza el reloj con la zona horaria que se le ha establecido. A continuación, con la instrucción “echo”, mostramos en el navegador web la hora. El módulo GPRS, leerá esta página web, donde solo se devuelve la hora siguiendo el formato, “año,mes,día,hora,minuto,segundo” y después se procesa para sacar los datos separados y poder fijar la hora leída en el reloj RTC DS3231[20].

#### *7.3.5 Presentación de los datos enviados*

Como detalle adicional a este trabajo, se ha optado por una presentación elegante y que posibilita una mejor apreciación de los datos obtenidos desde ATMega1284p, los cuales nos permiten una monitorización en tiempo real de los mismos. Para ello, se ha optado por utilizar una API de software libre llamada emoncms. En primer lugar, se trata de explicar que es emoncms y que funciones nos ofrece.

Esta aplicación, como su propio nombre indica, es un CMS (Content Management System o Sistema de Gestión de Contenidos) como lo pueden ser Drupal o WordPress, pero en este caso dirigido a sistemas para la gestión sensores. Este software libre ha sido desarrollado por <https://openenergymonitor.org/emon>, la cual dispone de un conjunto de sensores fabricados por esta empresa, para la gestión energética, como podemos observar en la [fig39].

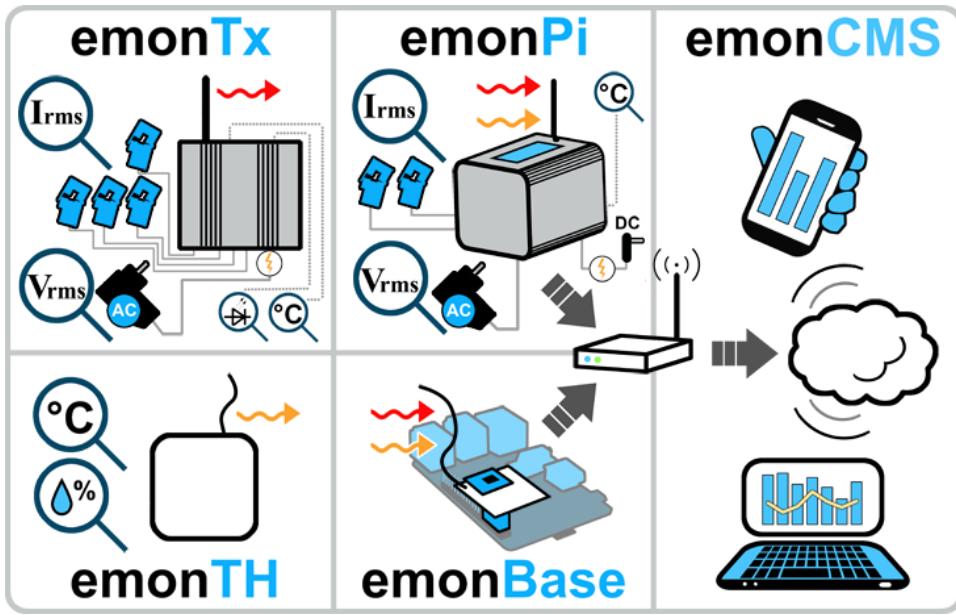


Figura 39: OpenEnergyMonitor se compone de estos dispositivos para la monitorización desde emonCMS.

A pesar de tener su propio hardware, es posible mediante la utilización de su API en base como se especifica en su web, seguir la monitorización de cualquier otro tipo de sensor, el estado de algún dispositivo, etc. Para su uso, es tan sencillo realizar como el registro en su web, y seguir las instrucciones facilitadas en el apartado <https://emoncms.org/site/api> donde se indican varias formas para realizar el envío de los datos. Una vez que se ha elegido el método para enviar los datos, hay que realizar un envío de una primera tanda de datos, para generar los feeds, los cuales son los datos que leeremos para generar nuestras gráficas. Hecho esto, solo queda diseñar el dashboard con el tipo de gráfico y datos que se quiera. A continuación, se muestra el resultado del dashboard generado para este trabajo, y como queda el resultado final que muestran los datos en la [fig40].



Figura 40: Resultado final del dashboard que aloja las gráficas para mostrar los datos recogidos por los sensores de nuestro ATMega1284p.

## 8 JUEGO DE PRUEBAS

### 8.1 Necesidad de toma de medidas

Una vez diseñado y montado el circuito. Para conocer si nuestro propósito es cumplido por el registrador, es necesario conocer dos parámetros relevantes, uno es la intensidad y otro el voltaje. El voltaje en este caso, es constante para todo nuestro diseño, fijado por el regulador de voltaje DC-DC LM2596[10], el cual nos fija una VCC para alimentar el circuito de 5V. Hay que tener en cuenta para un posterior desarrollo del consumo final que, según su hoja de características, este regulador tiene una eficiencia energética de un 80%, por lo que habrá que dimensionar el cálculo para obtener el consumo final. A continuación, se explica los datos a medir:

- Medida de corriente: En este caso, para realizar las lecturas de la corriente consumida por el sistema, se ha optado por un sensor digital que es capaz de medir corriente, varios voltajes y potencia. Este sensor, el ina219[41], se utiliza desde un segundo arduino, que conectados los dos mediante una comunicación Serial, el dispositivo a medir, envía un carácter, para así medir los tiempos que utiliza para realizar la toma de medidas y el tiempo que se encuentra en modo sleep.

Este sensor mediante sus librerías y haciendo uso de sus métodos, puede ajustar su resolución hasta 12 bits, ocupando 532  $\mu$ s a dicha resolución, lo cual nos permite detectar rangos de variación de 0.8 mA, pudiendo medir una entrada diferencial de corriente hasta  $\pm 3$  A y +26Vcc. Modificando el rango de medida y sin utilizar la ganancia del amplificador interno, la corriente máxima que es capaz de media es de  $\pm 400$  mA, lo que nos da un rango de variación de 0.1 mA. En la [fig41] aparece dicho sensor.

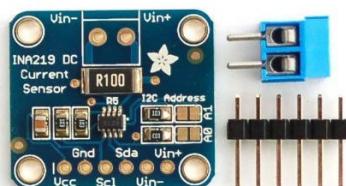


Figura 41: Sensor de corriente digital, INA219B, utilizado para medir el consumo de nuestro circuito.

Gracias a esto, se pueden obtener con gran precisión, en caso de tener corrientes muy pequeñas y además mucha resolución debido a poder tomar tantas medidas en un intervalo tan pequeño, esto nos permite construir una gráfica con gran detalle para observar posibles fluctuaciones en el consumo.

## 8.2 Circuitos de medida

Como se ha aclarado en el apartado anterior, el único circuito de medida necesario, es una shields comercial, un sensor de corriente, el ina219B. A continuación, se incluye un esquema en la [fig42] del diseño realizado mediante fritzing, para ilustrar el montaje necesario para medir la corriente.

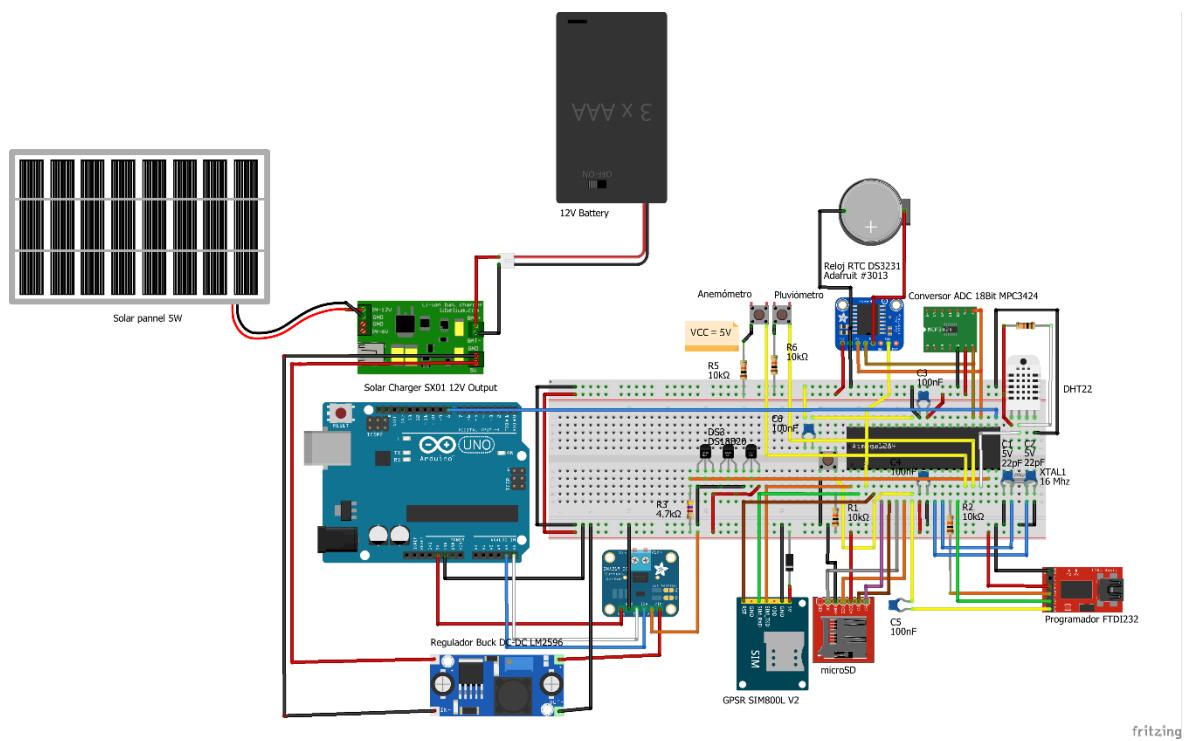


Figura 42: Circuito ilustrativo, de la toma de corrientes. Como se puede observar, la corriente consumida, circula a través del sensor, posibilitando su medida

Será la segunda placa, Arduino UNO, la encarga de ir mostrando mediante la consola Serial, todas las medidas durante el proceso de medida y cuando se encuentra en modo sleep nuestra placa. Como se puede ver en la imagen, hay un cable azul, que va desde ATMega1284p al pin 8 de arduino UNO. Será por aquí, cuando reciba un estado a nivel bajo cuando comience el proceso de medida y así obtener la corriente consumida en cada intervalo.

Para guardar estas medidas para su posterior análisis, se ha utilizado un programa llamado CoolTerm, el cual almacena en un fichero de texto los datos que manda Arduino a través del puerto serie. Con esto evitamos la necesidad de utilizar una shield de expansión de memoria para almacenar los datos en la tarjeta SD. Este proceso solo precisa de hacer una vez, tomando como no, varias lecturas, para obtener con la mayor exactitud posible, el consumo total de nuestro circuito, estrictamente necesario para el posterior dimensionamiento del SFA (Sistema Fotovoltaico Autónomo), que será el encargado de alimentar a nuestro circuito.

### 8.3 Registrador de datos con librería Jeelib

La librería Jeelib ha sido desarrollada por el equipo Jeelabs. Está compuesta por un conjunto de cabeceras y clases sencillas para funcionar con el IDE de Arduino. Viene implementada con varias clases como Port, MilliTimer, PortI2C o Sleepy. En nuestro caso, será la clase Sleepy la que nos facilite el ahorro en el consumo energético que necesitamos para nuestro propósito.

La clase Sleepy nos permite trabajar en modo bajo consumo a través del Watchdog Timer y el modo Powerdown. Para poder utilizarlo correctamente, debemos utilizar un vector para manejar la interrupción de watchdog. Esta clase contiene los siguientes métodos:

- Byte Sleepy::loseSometime(Word msec): Es utilizada para estar un tiempo dado en milisegundos en el modo SLEEP\_MODE\_PWR\_DOWN. Para usar este método debemos incluir en el código una definición de la interrupción WDT. La forma más sencilla de hacerlo es incluyendo la línea ISR(WDT\_vect){Sleepy::watchdogEvent();}. Esta función sustituye a la función delay() e induce a nuestro microcontrolador al modo Powerdown durante el periodo de tiempo indicado
- Void Sleepy::powerDown(): Este método como el anterior, nos permite entrar en el modo SLEEP\_MODE\_PWR\_DOWN. Sin embargo, utilizando esta función, como no se utiliza el timer watchdog, y hay que recordar que, desde este modo de bajo consumo, el microcontrolador solo puede ser despertado a través de interrupción

programada vía watchdog o interrupciones hardware, será necesario utilizar interrupciones hardware para poder despertarlo.

El código utilizado en esta solución software de bajo consumo ha sido los dos, debido a que en la inicialización del módulo GPRS se necesita cierto tiempo para realizar la conexión a la estación base y establecer la conexión de datos, se mantiene el microcontrolador dormido con la opción de delay() de bajo consumo contenido en el método Sleepy, además de utilizar el modo Powerdown, el cual es utilizado en el funcionamiento general del código debido a que la tanda de medidas son disparadas gracias a la interrupción hardware generada por el reloj RTC DS3231 en los segundos 0 y 30. Por tanto el código siguiente es el que se ha utilizado:

```
//Inclusión de la librería
#include <JeeLib.h> // sensor library from https://github.com/jcw/jeelib
//Inicializa el vector WDT para hacer uso del timer watchdog
ISR(WDT_vect) {
    Sleepy::watchdogEvent();
}
/*
Es recomendable este pequeño intervalo de 100 ms, debido a que se ha observado
que, al utilizarlo, por ejemplo, antes de mostrar caracteres por Serial, que genera
caracteres extraños imposibles de leer.
*/
delay(100);
//Esta instrucción como se acaba de comentar, simula la instrucción anterior delay()
Sleepy::loseSometime(14900);
/*
Esta instrucción duerme de manera infinita nuestro micro mientras no sea despertado
desde una interrupción hardware, que el caso de este proyecto, puede venir desde el
anemómetro, pluviómetro o el reloj RTC DS3231, encargado de disparar el proceso de
medida.
*/
Sleepy::powerDown();
```

Una vez incluida la librería y las instrucciones en nuestro programa general, se han realizado una campaña de medidas, concretamente 98 medidas de todo el proceso completo el cual dura 30 segundos. Durante el proceso se han tomado medidas de corriente cada 100 milisegundos lo cual nos genera 300 medidas. Para obtener la corriente basta con realizar la suma de todas las muestras extraídas y dividirlo entre el nº total de muestras. Esto nos daría la corriente consumo a la hora. Como las muestras extraídas están en mA el resultado obtenido se da en mAh. A continuación, se adjunta la [tabla 4] de las medidas realizadas en mA.

40,10	33,34	44,92	32,74	34,59	34,86	34,13	37,28	32,97
36,03	37,81	32,72	33,34	36,92	35,26	33,78	35,07	43,49
34,63	36,81	31,41	40,53	32,75	31,48	32,97	42,68	32,57
41,97	39,16	29,94	37,62	34,13	35,72	37,21	43,06	41,45
34,70	39,68	44,71	34,91	36,41	33,72	35,53	37,66	40,34
37,63	38,36	33,99	33,73	35,85	32,62	34,53	34,24	33,92
35,41	40,69	33,68	35,30	33,68	34,46	32,55	35,03	31,06
37,88	44,57	34,26	31,76	34,87	35,56	33,73	34,31	31,20
33,85	41,84	39,21	34,63	34,91	34,29	33,43	31,52	30,71
33,47	34,04	49,22	39,28	33,69	42,31	34,13	40,07	37,21
34,66	41,99	33,75	33,87	31,97	33,09	36,97	32,94	32,97
								35,96945

Tabla 4: Tabla con las medias

Estas 98 medidas, son las medias obtenidas de 98 procesos completos, donde cada uno contiene 300 medidas, ya que se toman medidas de corriente cada 100ms. El dato detallado en azul, es la media obtenida de todas las medias, lo que nos da un consumo de 36 mAh (miliamperios/hora). Para poder apreciar con detalle el consumo del circuito a lo largo del proceso completo de toma de medidas, almacenamiento de datos, lectura de datos, envío de datos y entrar en modo sleep, se adjunta la [fig43] donde aparece un gráfico que contiene el tránscurso de los 30 segundos que dura el ciclo completo.

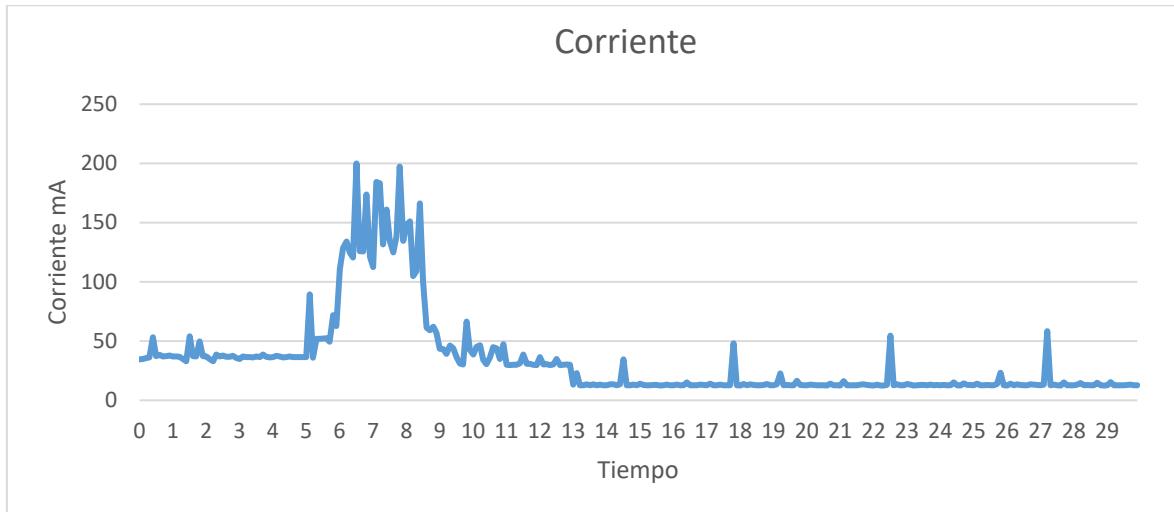


Figura 43: Gráfico con las medidas de corriente obtenidas cada 100 ms, tomadas a lo largo del proceso completo entre medidas de 30 segundos de duración.

Vamos a explicar lo que se puede apreciar en la [tabla 4]. El proceso de medida comienza una vez se cumple la condición de que es el segundo 0 o 30. Previo a esto, el microcontrolador ha sido despertado vía interrupción hardware desde el reloj y da comienzo el ciclo. En primer lugar, se toman las medidas y se almacenan los datos durante unos 5 segundos como se puede ver en la gráfica. Una vez terminado, se leen

los nuevos datos que acaban de ser guardados y el módulo GPRS establece la conexión HTTP donde son enviados a través de una petición GET.

Este proceso como se puede observar es breve, de unos 3-4 segundos. Una vez acaba se le ordena entrar en modo sleep, pero a diferencia de ATMega1284p, este modo sleep es un modo “Autosleep”, ¿qué quiere decir esto?, que el módulo tarda unos 3 segundos en darse cuenta de que no está recibiendo comandos AT, con lo que entra en modo sleep quedando el consumo mínimo en unos 12,5-13 mA. Con esto logramos tener más de la mitad del tiempo que dura todo el ciclo en bajo consumo, logrando consumir muy poca energía.

Para realizar una comparación con el trabajo predecesor [\[1\]](#), se incluye la [\[fig44\]](#), que muestra un gráfico con los resultados obtenidos usando las diferentes librerías de bajo consumo y la reducción de frecuencia de ATMega328p en la [\[tabla 5\]](#), usado en el trabajo anterior, a 1 MHz.

9:44:00	175,484375	1445,4375		0,1016795	13,298025
9:44:10	172,78125	1453,01463		0,0940414	13,3677346
9:44:20	173,453125	1450,3125		0,09593988	13,342875
9:44:30	175,514625	1449,1875		0,10176498	13,332525
9:44:40	173,671875	1451,04688		0,096558	13,3496313
9:44:50	173,78125	1449,73438		0,09686705	13,3375563
9:45:00	175,859375	1444,90625		0,10273912	13,2931375
9:45:10	173,9375	1448,54688		0,09730856	13,3266313
9:45:20	173,890625	1450,40625		0,09717611	13,3437375
9:45:30	175,875	1443,35938		0,10278327	13,2789063
9:45:40	173,828125	1449,10938		0,09699951	13,3318063
9:45:50	173,90625	1453,39063		0,09722026	13,3711938
9:46:00	175,03125	1444,39063		0,10039912	13,2883938
9:46:10	173	1448,85938		0,09465951	13,3295063
9:46:20	173,6875	1449,51463		0,09660215	13,3355346
9:46:30	175,453125	1443,98438		0,1015912	13,2846563
9:46:40	173,75	1450,5625		0,09677875	13,345175
9:46:50	173,625	1449,32813		0,09642554	13,3338188
9:47:00	175,5625	1443,25		0,10190025	13,2779
9:47:10	173,59375	1449,4375		0,09633724	13,334825
9:47:20	173,014625	1447,34375		0,09470081	13,3155625
9:47:30	175,71875	1440,51463		0,10234176	13,2527346
9:47:40	173,734375	1447,03125		0,0967346	13,3126875
9:47:50	173,765625	1445,875		0,0968228	13,30205
9:48:00	175,96875	1445,21875		0,10304818	13,2960125
9:48:10	174,014625	1446,3125		0,09752649	13,324475
9:48:20	172,171875	1449,10938		0,09231951	13,3318063
9:48:30	175,609375	1446,4375		0,10203271	13,307225
9:48:40	173,828125	1446,82813		0,09699951	13,3108188
9:48:50	172,75	1450,82813		0,09395309	13,3476188
9:49:00	175,921875	1443,6675		0,10291572	13,281925
9:49:10	173,9375	1446,17188		0,09730858	13,3047813
9:49:20	173,953125	1448,92188		0,09735271	13,3300813
9:49:30	175,921875	1442,95313		0,10291572	13,2751688
9:49:40	173,984375	1446		0,09744101	13,3032
9:49:50	173,96875	1445,34375		0,09739686	13,2971625
9:50:00	175,9375	1443,67188		0,10295984	13,2817813
9:50:10	173,703125	1448,79688		0,09664641	13,3289313
9:50:20	173,75	1446,34375		0,09677875	13,3063625
9:50:30	175,75	1438,3125		0,10243006	13,232475
9:50:40	173,09375	1444,40625		0,09492441	13,2885375
9:50:50	173,828125	1445,64063		0,09699951	13,2988938
9:51:00	175,953125	1432,65625		0,10300403	13,1804375
9:51:10	173,8125	1443,26563		0,09695533	13,2780438
9:51:20	173,53125	1444,26563		0,09616061	13,2872438
9:51:30	175,796875	1442,28125		0,10256252	13,2689875
9:51:40	173,703125	1441,375		0,09664641	13,26065
9:51:50	173,796875	1445,14625		0,0969112	13,2953455
9:52:00	175,6875	1437,98438		0,10225346	13,2294563
9:52:10	173,8125	1440,78125		0,09695533	13,2551875
9:52:20	173,75	1443,9375		0,09677875	13,284225

Tabla 5: Tabla con las medidas de corriente obtenidas cada 10s del consumo del registrador de datos principal, tomadas desde el registrador de datos secundario. Estos implementan las nuevas mejoras software para reducir el consumo.

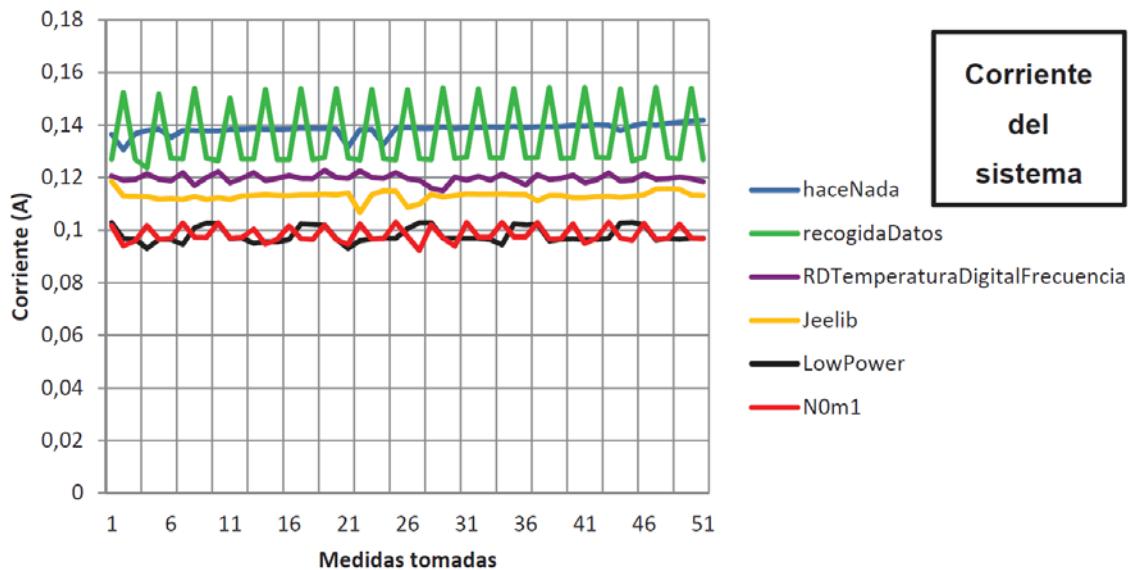


Figura 44: Gráfica de las medidas de corriente del trabajo predecesor.

La corriente fue tomada de un registrador de datos principal al que se le mejoró el software para tomar las medidas, y se sustituyeron los sensores de temperatura analógicos por digitales. Estas medias fueron tomadas cada 10 segundos, desde un registrador de datos secundario. Como se puede observar, el menor consumo obtenido se consigue con la librería LowPower llegando a unos 98 mAh. Con respecto a los 36 mAh obtenidos en este trabajo, se ha conseguido reducir un 63,27% el consumo total del registrador, recalmando que se han agregado una mayor cantidad de sensores, un SFA y la posibilidad de enviar los datos a internet, para tener una monitorización en tiempo real de los mismos.

Como apunte final a este apartado, se puede afirmar que se ha conseguido una reducción del consumo si lo comparamos con el anterior trabajo. Hay que resaltar que estas mejoras han sido obtenidas de forma paralela agregando soluciones hardware y software, tal cual fue dejado como líneas de futuro en el trabajo anterior, lo que ha desembocado en un ahorro energético muy superior. Con esto no solo se ha logrado alargar la vida operativa del diseño cuando se alimenta desde nuestro SFA, sino que cumple y mantiene nuestros objetivos primarios: crear un sistema de bajo consumo y bajo coste autónomo para poder ser utilizado en lugares con deficiencias energéticas y/o económicas.

## 9 DISCUSIÓN DE RESULTADOS

Este apartado es de vital importancia en el desarrollo del trabajo porque es aquí donde se observa la calidad del montaje. Para ello el sistema se ha mantenido el sistema tomando medidas. Para ello se adjunta una muestra de las medidas tomadas por el registrador una vez ejecutado el diseño final, mostrado en la [\[fig45\]](#).

```
Fecha,(DHT22)Humedad,Temperatura,(DS18B20),Temperatura_S1_0x07,S2_0x09,S3_0x09,MCP3424(CH1),(CH2),(CH3),(CH4),Anemometro(m/s),Pluviometro(L/m2_dia),
31-08-2016_00:13:30.25.30.30.80.30.62.30.56.44.06.-0.36.-0.37.-0.39.-0.41.0.00.0,
31-08-2016_00:14:00.18.30.30.60.30.50.30.37.44.00.-0.41.-0.41.-0.39.-0.39.0.00.0,
31-08-2016_00:14:30.18.40.30.50.30.56.30.50.44.00.-0.41.-0.44.-0.41.-0.39.0.00.0,
31-08-2016_00:15:00.18.40.30.40.30.50.30.44.43.94.-0.36.-0.37.-0.41.-0.41.0.00.0,
31-08-2016_00:15:30.18.50.30.40.30.44.30.37.43.94.-0.39.-0.37.-0.41.-0.41.0.00.0,
01-09-2016_09:17:30.31.10.31.00.31.12.31.12.45.56.1200.91.1200.89.1200.98.1200.31.0.00.0,
01-09-2016_09:18:00.31.10.31.00.31.12.31.19.45.56.958.91.958.33.958.83.958.72.0.00.0,
01-09-2016_09:18:30.32.10.31.00.31.19.31.19.45.50.897.48.850.84.797.47.762.64.0.00.0,
01-09-2016_09:19:00.34.60.31.00.31.25.31.19.45.50.1127.70.1155.92.645.55.62.86.0.00.0,
01-09-2016_09:19:30.34.20.31.00.31.19.31.19.45.50.1202.50.1202.53.1202.00.1201.70.0.00.0,
01-09-2016_09:20:00.33.00.31.10.31.19.31.19.45.50.244.25.123.53.235.91.866.48.0.00.0,
01-09-2016_09:20:31.30.31.10.31.25.31.19.45.56.914.81.913.23.918.23.917.91.0.00.0,
01-09-2016_09:21:00.31.30.31.10.31.25.31.19.45.50.932.14.932.06.932.56.932.59.0.00.0,
01-09-2016_09:21:30.31.20.31.10.31.25.31.19.45.50.932.58.932.36.931.98.932.59.0.00.0,
01-09-2016_09:22:00.31.20.31.10.31.19.31.19.45.50.931.92.931.81.932.56.932.41.0.00.0,
01-09-2016_09:22:30.31.20.31.10.31.19.31.19.45.44.932.69.932.48.931.80.932.47.0.00.0,
01-09-2016_09:23:00.31.20.31.10.31.25.31.19.45.44.932.08.932.31.932.48.931.98.0.00.0,
01-09-2016_09:23:30.31.20.31.10.31.19.31.19.45.44.932.27.932.39.932.06.932.34.0.00.0,
01-09-2016_09:24:00.31.10.31.10.31.25.31.19.45.38.932.05.932.33.932.34.932.22.0.00.0,
01-09-2016_09:24:30.31.20.31.10.31.25.31.19.45.38.932.34.932.28.932.00.931.80.0.00.0,
01-09-2016_09:25:00.31.20.31.10.31.25.31.19.45.38.931.89.932.38.932.42.929.88.0.00.0,
01-09-2016_09:25:30.31.30.31.10.31.25.31.19.45.38.932.19.932.22.932.14.930.92.0.00.0,
01-09-2016_09:26:00.31.20.31.10.31.25.31.19.45.31.932.11.931.84.932.22.932.33.0.00.0,
01-09-2016_09:26:30.31.30.31.10.31.25.31.19.45.31.932.16.932.05.931.73.932.16.0.00.0,
01-09-2016_09:27:00.31.30.31.10.31.25.31.19.45.31.931.56.932.11.932.16.931.56.0.00.0,
```

Figura 45: Tanda de medidas recogidas con el registrador de datos, cada una con 30s de diferencia.

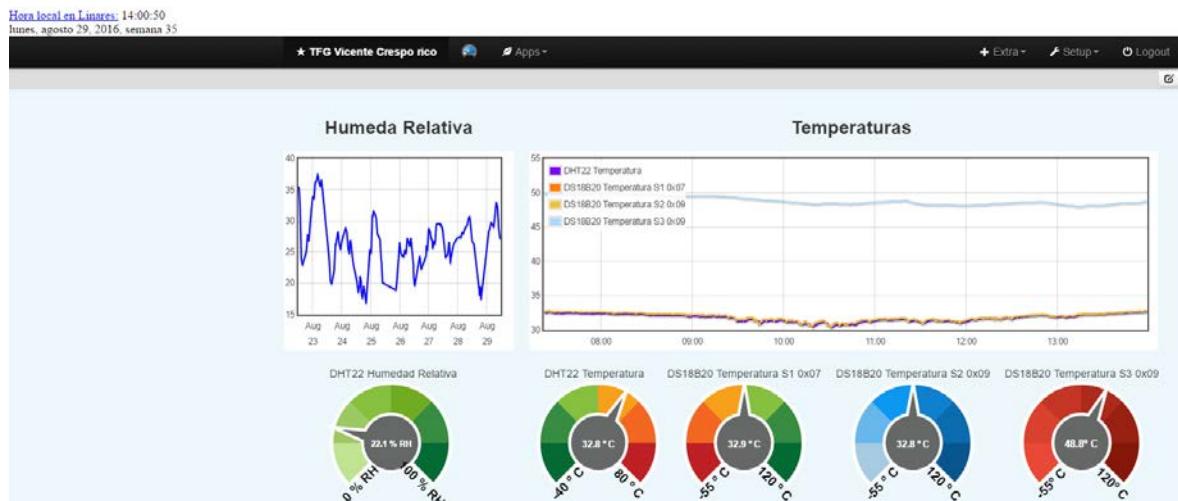
Aclarando un poco los valores dados, hay datos que aparecen extraños como el anemómetro y pluviómetro, ya que una vez implementados, las entradas permanecen en vacío hasta su colocación final. Cabe recordar que, aunque las interrupciones por hardware siempre serán medidas cuando se producen, la lectura del pluviómetro se va almacenando en la variable volatile en la SRAM además de en la tarjeta microSD como seguridad en caso de corte de suministro o reinicio por fallo en el envío de las medidas. Al realizar la subida de los datos cada 30 segundos, se ha decidido enviar constantemente 0 hasta que llegan las 0 o 12 horas en punto, que como se recuerda, como norma general, los datos tomados sobre la cantidad de lluvia suelen ser dados cada 12 horas.

También, puede apreciarse pequeños intervalos donde no hay medidas. Esto se debe a que, en ciertas ocasiones, por fallo en la cobertura con la estación base de telefonía o por la propia disponibilidad del servidor que aloja las aplicaciones PHP donde se almacenan los datos, como se ha impuesto, puede tener un máximo consecutivo de 2 fallos en el envío de los mismos, después de esto, el mismo se provoca un reinicio por hardware donde se ha estimado una pérdida entre 1-2 lecturas en la tarjeta microSD y 3-4 en el servidor, debido al intervalo de tiempo que toma el dispositivo en reiniciarse y sincronizar la hora con el servidor. Una vez reiniciado el sistema se sobreponer, y

continúa tomando y enviando las medidas con total fiabilidad. Esta situación no es frecuente, donde tras varios días de pruebas, han sucedido días completos sin ningún fallo en el envío de los datos, con lo que se ha conseguido un sistema fiable y preciso.

Es importante comentar que los factores de cobertura y disponibilidad, son factores totalmente ajenos a este trabajo, siendo el nuestro, el asegurar una correcta toma de las medidas y asegurar en la mayor probabilidad posible el envío de los datos al servidor para su posterior monitorización. A nuestro juego de pruebas se agrega el resultado de la monitorización que proporcionan las gráficas en tiempo real, apreciando con mejor detalle los cambios en nuestro sistema en la [fig46].

#### Página web de Vicente Crespo Rico



#### Conversor analógico/digital MCP3424 para sensores analógicos



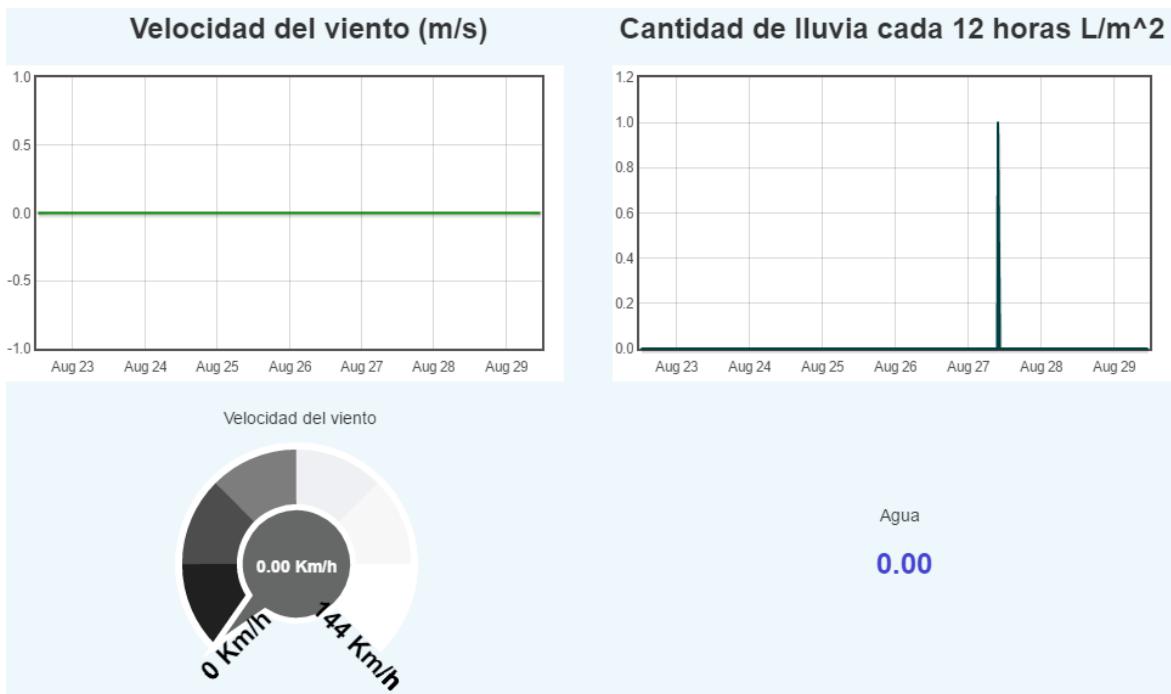


Figura 46: Página web creada para el presente trabajo, donde se muestran las gráficas generadas para cada elemento que compone el registrador.

Este sería el resultado final desde que el dispositivo sincroniza la hora desde internet, utilizando un sistema parecido al protocolo NTP utilizado para sincronizar relojes de los sistemas informáticos a través del enruteamiento de paquetes, pero en este caso con una mayor sencillez, la toma de medias y su almacenamiento, lectura de las mismas y posterior envío desde el módulo GPRS mediante HTTP en una petición GET, almacenamiento y procesado de los datos en el server, generando una copia de seguridad “prácticamente idéntica” (se entrecilla sugiriendo al lector, los posibles fallos en el envío de las medidas) al generado en la microSD, generación de la URL que envía los datos a la plataforma de contenido emonCMS, y finalmente como se muestra en las gráficas anteriores el resultado.

## 10 CONCLUSIONES

Uno de los objetivos de este trabajo ha sido la optimización energética, buscando una máxima reducción en el consumo y la autonomía de un registrador de datos de bajo coste. Para ello, se ha realizado un estudio previo de los problemas a unas necesidades de diseño y la implementación de los distintos métodos estudiados en el análisis previo. En base a los objetivos expuestos frente a la problemática encontrada en este manuscrito, se han llegado a las siguientes conclusiones.

En primer lugar, frente a la problemática de no encontrar soluciones comerciales que se adecuen a los requisitos del diseño, se ha tomado la opción de utilizar plataformas de hardware y software libre que surten de mayor flexibilidad, utilizando para ello el uso de bareboard como solución hardware, y utilizando el entorno de desarrollo de Arduino, haciendo uso de librerías que den compatibilidad y apoyo al hardware del diseño, como solución software.

Los objetivos de reducir el consumo respecto al anterior trabajo, se han conseguido, llegando a una reducción del 65% durante la medida, y cercano al 90% cuando el sistema se encuentra en modo sleep mientras espera a la siguiente toma de medidas. Esto se ha logrado haciendo uso de las técnicas más agresivas en el apartado hardware utilizando los mínimos componentes necesarios, y en el apartados software, desarrollando un código capaz de adentrar el microcontrolador ATMega1284p utilizado en este trabajo, en el modo sleep. Ya que el componente que mayor consumo presenta es el módulo GPRS, también se hacía necesario poder reducir su consumo y se ha logrado utilizando un comando AT que lo induce a ese modo, logrando reducir drásticamente el consumo desde unos valores pico de 350mA hasta 12,5mA, lo que consigue un drástico ahorro energético.

Por último, frente a la necesidad de almacenar los datos a modo de copia de seguridad y de enviarse para una monitorización en tiempo real fuera del emplazamiento original, se ha conseguido utilizando para ello, una shield de expansión de memoria microSD a modo de backup, y para el envío de los datos, la mejor opción frente a las posibilidades expuestas en antecedentes, ha sido el módulo GPRS, el cual permite una comunicación con un ámbito de cobertura mundial tan solo insertando la sim del operador de la zona donde se encuentre ubicado el diseño, logrando así, enviar los datos de manera satisfactoria y dotando al sistema de detección de fallos en el envío, gracias al

conocimiento de las respuestas tanto del módulo como del servidor donde se envían los datos, lo que obliga a un autoreinicio del mismo para evitar la mínima pérdida de estos.

Como apunte final, durante este diseño, se ha mantenido la línea de que todo el material usado debía ser de bajo coste, sin dejar de lado la calidad y procurando que estos tengan el menor consumo posible o pueden inducirse el modo sleep en ellos.

Gracias al trabajo realizado a lo largo de este proyecto, se han conseguido alcanzar todas las metas, haciendo posible la construcción de un registrador de datos de bajo consumo y bajo coste autónomo, para monitorizar sistemas fotovoltaicos híbridos, basado en plataformas de hardware y software libre.

## 11 LÍNEAS FUTURAS

En cuanto a las líneas futuras, se pasa a describir las más importantes de cara a una continuación del trabajo aquí expuesto.

Una posible línea de mejora sería la de usar una petición POST en lugar de la usada en este trabajo mediante GET, en caso de querer ocultar la información, ya que, al pasar los datos por navegador, la información se podría visualizar e incluso modificar por algún tercero. Si se quisiera además codificar la información ya que tanto en GET como en POST, los datos se envían en texto plano, se podría usar el protocolo de encriptación AES, usando una librería creada para el microcontrolador usado, lo que resultaría un aspecto muy interesante en el tema de seguridad.

Otra opción, sería usar un protocolo de comunicaciones llamado mqtt (message queue Telemetry Transport) desarrollado por IBM, usado en comunicaciones M2M (máquina a máquina) en el ámbito IoT (internet de las cosas). Este protocolo está destinado a comunicar sensores, debido a que presenta un reducido ancho de banda y se utiliza en dispositivos que presentan unos bajos recursos (CPU, memoria, etc.). Este protocolo sigue una topología en estrella, donde existe un nodo central o bróker, encargado de gestionar la comunicación entre un gran número de clientes. Además, una característica es la posibilidad de cifrar las comunicaciones, lo que añade seguridad al envío de los datos.

Ya que se ha decidido usar para el visionado el administrador de contenido emonCMS, existe la posibilidad de programar la aplicación del microcontrolador ATMega1284p para realizar el envío directamente desde él. Esta opción necesita de una nueva programación, pero evitaría utilizar el server con las apps en PHP como intermediario.

## 12 BIBLIOGRAFÍA

- [1] Burgos Borrego, José Miguel, Estudio teórico-práctico para la optimización de consumo de un registrador de datos de bajo coste destinado a monitorización para aplicaciones en países en desarrollo, 2015, Escuela Politécnica Superior de Linares.
- [2] [Datasheet Arduino UNO R3](#)
- [3] [Datasheet SIM800 mas comandos AT \(comandos de modem para el chipset de GSM/GPRS\) y comandos AT SIM800L](#)
- [4] [Datasheet INA219 sensor de corriente y manual de uso](#)
- [5] [Datasheet ATMega1284p](#)
- [6] [Datasheet sensor de temperatura y humedad relativa digital DHT22](#)
- [7] [Datasheet sensor de temperatura digital DS18B20](#)
- [8] [Datasheet reloj RTC DS1307](#)
- [9] [Datasheet conversor ADC MCP3424](#)
- [10] [Datasheet convertidor reductor de voltaje \(Buck\) DC-DC LM2596](#)
- [11] [Regulador de carga solar PWM 3A 6V-12V SX01](#)
- [12] [Datasheet anemómetro 4.3515.61.100](#)
- [13] [Datasheet pluviómetro Rain-O-Matic Small Rain Gauge](#)
- [14] Batería (<http://es.rs-online.com/>)
- [15] Módulo fotovoltaico (<http://es.rs-online.com/>)
- [16] [IDE Arduino y varias librerías con detalles de uso.](#)
- [17] [Librerías para usar modos sleep en arduino de Jeelib](#)
- [18] [Librerías para conversor ADC MCP3424 y ejemplos](#)
- [19] [Modificación de una dependencia del paquete de librerías SD, donde se añade el pinout del ATMega1284p](#)
- [20] [Librería reloj RTC DS3231, datasheet y ejemplos.](#)
- [21] [Librería DHT11/DHT22 y ejemplos](#)
- [22] Librerías [OneWire](#) y [DallasTemperature](#) para sensor de temperatura digital ds18b20 y ejemplos [1-2](#)
- [23] Barquero Moreno, Gala, Red inalámbrica para la monitorización de Sistemas Fotovoltaicos Autónomos – SFA – en países en vías de desarrollo, PFC, Universidad de Jaén.

[24] [Identificación de sensores conectados al bus 1-Wire que en este caso ha sido para identificar los ds18b20 y toma de medidas](#)

[25] [Ejemplos de procesos de lectura/escritura SD](#)

[26] [Interrupciones por hardware para arduino](#)

[27] [Comandos AT para realizar una conexión HTTP para enviar datos mediante una petición GET](#)

[28] [Modo sleep chip GSM/GPRS SIM800L](#)

[29] Burgos Borrego, José Miguel, Detector de purificación de agua mediante SOLWAT, PFC, Universidad de Jaén.

[30] [Construcción de arduino con mínimos componentes, programación de bootloader, librería para añadir compatibilidad de ATMega1284p con el IDE de arduino y tabla comparativa microcontroladores ATMega más usado, con encapsulamiento tipo DIP.](#)

[31] [Protocolo mqtt para IoT](#)

[32] [Api emoncms](#)

[33] Alonso Abella, Miguel, Dimensionado de Sistemas Fotovoltaicos autónomos, Centro de investigaciones energéticas, medioambientales y tecnológicas.

[34] [Procedimientos de lectura/escritura de ficheros de texto plano](#)

[35] [Oliver Style, Energía solar autónoma. Planificación, dimensionado e instalación de un sistema fotovoltaico autónomo, Primera edición, mayo 2012](#)

[36] [Temporizador Watchdog](#)

[37] [Datasheet ATMega328p](#)

[38] [Arduino to Breadboard y ficheros necesarios para hacer compatible el IDE de Arduino con la placa](#)

[39] [Datasheet AMS1117 5.0](#)

[40] [Arduino Pro Mini](#)

[41] [Arduino Lilypad](#)

[42] [Arduino Moteino](#)

[43] [Arduino Mosquino](#)

[44] [ATMega16U2](#)

[45] [Cuatro alternativas a Arduino](#)

[46] [Raspberry Pi](#)

[47] [Nanode](#)

[48] [Libelium Wasp mote](#)

[48] [Termohigrómetro PCE-THB 40](#)

[48] [Logger de datos de temperatura y humedad Log110-Start](#)

[48] [Sistema de telecontrol GSM/GPRS HERMES LC-2](#)

[48] [Estación meteorológica PCE-FWS20](#)

## 13 PRESUPUESTOS

Presupuestos montaje por personal cualificado				
Elemento	Personal necesario	Precio por hora (€/h)	Horas de trabajo (h)	Precio Total (€)
BareBoard (Diseño y montaje) más reloj y conversor ADC	Ingeniero	32.55	15	709.3
	Técnico	27.63	8	
Expansión de memoria	Técnico	27.63	0.5	13.82
Módulo GPRS	Técnico	27.63	0.5	13.82
Sensores de temperatura y humedad relativa	Ingeniero	32.55	1	32.55
Encapsulado del sistema	Ingeniero	32.55	2	65.1
Anemómetro	Técnico	27.63	0.5	13.82
Pluviómetro	Técnico	27.63	0.5	13.82
Conversor DC-DC LM2596	Técnico	27.63	0.5	13.82
Sistema fotovoltaico autónomo (SFA) compuesto por batería, módulo solar y regulador de carga solar	Ingeniero	32.55	4	130.2
	Técnico	27.63	1	27.63
Precio total personal de trabajo: 1033.83 €				

Este presupuesto se ha diseñado para contener el presupuesto de la mano de obra del personal cualificado incluyendo desde el diseño de la bareboard, montaje de la misma, montaje de los módulos para el envío de datos, medida de temperatura y humedad relativa, velocidad del viento, agua, el bloque del SFA que incluye batería, módulo solar, regulador de carga solar y conversor DC-DC para adaptación de voltajes hasta su puesta a punto en el terreno.

Presupuesto de los componentes				
Elemento	Composición	Unidades (u)	Precio por unidad (€/u)	Precio total (€)
BareBoard más componentes	AtMega1284p, 1 condensador 47µF, 2 de 22pF, 4 de 100nF, 2 resistencias de 10KΩ, 1 diodo 1N4002, 1 socket para montaje, cristal	1	18.64	18.64

	16 MHz, tira pin 36w y placa fotosensibilizada positiva			
Conversor ADC MCP3424	Placa comercial montada	1	15.1	15.1
Reloj RTC DS3231	Reloj DS3231 más pila CR2032	1	2.09	2.09
Regulador de carga solar SX01	Placa comercial más compartimento estanco	1	7.29	7.29
Módulo GPRS SIM800L	Módulo GPRS SIM800L	1	5.8	5.8
Convertidor DC-DC 3 A regulable LM2596	Convertidor DC-DC 3 A regulable LM2596	1	1.45	1.45
Módulo expansión de memoria microSD	Módulo expansión de memoria microSD	1	1.17	1.17
Sensor de temperatura ds18b20	Sensor de temperatura ds18b20	2	1	2
Sensor de temperatura ds18b20	Tipo sonda	1	1.4	1.4
Sensor de temperatura y humedad relativa DHT22	Sensor de temperatura y humedad relativa DHT22	1	3.56	3.56
Batería plomo-ácido de ciclo profundo 12V-7,2Ah, AGM	Batería plomo-ácido de ciclo profundo 12V-7,2Ah, AGM	1	43	43
Módulo solar 5W	Módulo solar policristalino	1	17	17
Anemómetro Small Wind Transmitter de Thies Clima 4.3515.61.100	Anemómetro Small Wind Transmitter de Thies Clima 4.3515.61.100	1	112	112
Pluviómetro RAIN-O- MATIC de salida de pulsos con 1 relé reed	Pluviómetro RAIN-O- MATIC de salida de pulsos con 1 relé reed	1	35	35
Encapsulado del sistema	Caja IP65	1	7.77	7.77
Conexiones	Cable 22 – 26 AWG 1m	1	1	1
Honorarios por diseño e implementación del sistema: 150 €				
<b>Precio total de componentes: 424.27 €</b>				

Este último presupuesto está compuesto por los dispositivos que componen el sistema al completo sobre el terreno en el cual, al final se agregan unos honorarios

impuestos en caso de una posible réplica. En el futuro este presupuesto podría variar en función de si el cliente quiere agregar más componentes al diseño final.

## 14 ANEXOS

En el siguiente anexo, se tratan de explicar el montaje del diseño final del bareboard junto con el resto de shield del diseño final, además de las mejoras realizadas por software para conseguir la reducción máxima del consumo frente al trabajo predecesor [1], e impuesto como meta.

### 14.1 Manual de usuario

A continuación, se procede al desarrollo de un manual de usuario, debiendo seguir este en caso de una posible réplica del diseño de este proyecto, necesario para su funcionamiento y puesta a punto. En este se detallan las siguientes partes:

- Conexionado de los distintos sensores y shields necesarios para la toma de medidas y almacenamiento de los datos.

Para llevar a cabo el conexionado de los componentes y shields, solamente es necesario observar la primera parte del código desarrollado donde se declaran las librerías y definen las variables globales usadas, para poder realizar el correcto conexionado de las diferentes partes. Hay que matizar que se pueden agregar más sensores de temperatura ds18b20 al mismo pin, ya que el software los identifica y ordena para su posterior lectura y procesado.

- Instalación del entorno de programación de Arduino, instalación de librerías y agregar la compatibilidad de la placa con el IDE.

Una vez descargado e instalado el entorno de programación de Arduino, para agregar las librerías necesarias para este trabajo, basta con dirigirnos al directorio de instalación donde se encuentra una carpeta llamada “libraries”, en la cual hay que copiar las carpetas de las librerías usadas. Para agregar la compatibilidad del microcontrolador ATMega1284p, basta con seguir las mismas indicaciones que el apartado [14.2] de este trabajo.

- Carga del software e inicio del proceso, junto con la puesta en hora del reloj RTC.

Una vez agregada la compatibilidad del entorno IDE de Arduino para nuestro ATmega1284p, se procede a escribir el software desarrollado en su memoria FLASH, y previo a esto, para poner en hora el reloj, parte crucial en este trabajo, ya que es el encargado de disparar el proceso de medida y almacenamiento y envío de los datos, basta con quitar los comentarios del software de la línea “clock.setDateTime(\_\_DATE\_\_, \_\_TIME\_\_);” donde el compilador toma la hora en la que se escribe el software en la FLASH y queda fijada en el RTC. Hay que advertir, que una vez fijada la hora hay que volver a comentar esta línea y volver a escribir el programa en el microcontrolador, ya que, de no ser así, en un posible corte de suministro, el reloj volvería a tomar la misma hora en la que se escribió el programa.

- Comprobación del correcto funcionamiento y puesta en marcha.

Para comprobar el correcto funcionamiento, basta conectar nuestra placa al PC mediante USB, y mediante el monitor Serial, visualizar como el módulo GPRS envía los datos al servidor, y si los datos se suben correctamente, el servidor responderá con HTTP 200 OK, síntoma de que los datos se han subido satisfactoriamente. Hay que advertir que, aunque los datos no se envíen al servidor web, los datos se grabarán correctamente en la tarjeta microSD. Para poder comprobarlo, solo es necesario ver el monitor Serial del IDE de Arduino, ya que previo al envío de los datos, estos han debido de leerse de la tarjeta después de guardarlos en ella, con lo que se podrán visualizar por pantalla, indicando que se guardaron correctamente.

- Instalación de la aplicación web para procesado y almacenamiento de datos.

Este apartado es muy sencillo. Para poder utilizar la aplicación web programada en PHP, basta con copiarla en el directorio raíz del servidor web que hayamos instalado, o de igual forma si se contrata algún hosting donde alojar nuestra aplicación. En el servidor, los datos se almacenarán siguiendo el mismo criterio que se usa para almacenarlos en la tarjeta microSD, por lo que si son enviados correctamente (hay que recordar que el envío de los datos depende de la cobertura móvil que tenga el módulo GPRS SIM800L y de cómo se encuentre el servidor donde se aloja nuestra aplicación web), el fichero donde se guardan debe ser una copia exacta al de la tarjeta microSD de Arduino.

- Instalación de API para el almacenamiento de los datos en una base de datos para su posterior visionado en gráficas, o de forma opcional a la instalación de la API que genera las gráficas, es posible registrarse en la web del creador de la

API, omitiendo su instalación, ya que solo es necesario el registro y el posterior diseño de la gráfica en función de los datos recibidos.

En este trabajo, se ha optado por la segunda opción, para tratar de optimizar tanto en el tiempo de instalación de la API, como en evitar la posterior configuración de la misma. Para ello hay que dirigirse a la página web <https://www.emoncms.org>. Una vez en ella hay que registrarse para poder obtener la APIKEY, una clave para permitir el proceso de lectura/escritura de los datos, para una vez almacenados en la base de datos, poder generar las gráficas para su posterior visionado. La APIKEY cambia con respecto a cada usuario, por lo que esta debe ser cambiada en la aplicación web.

## 14.2 Uso de bareboard

Para tratar de explicar de la forma más entendible posible, se realiza un estudio de la placa Arduino UNO, por ser la más extendida. Esta placa tiene el microcontrolador ATMega328p [37]. A pesar de ser uno de los primeros modelos (ya que su predecesor es el Duemilanove y anterior a este y primero de todos el Diecimila), dispone de una gran cantidad de funcionalidades y buses de comunicación.

El datalogger del cual se parte en este trabajo, tiene la revisión 3, donde su mayor diferencia es el cambio del controlador USB-UART(serie), del ATMega8U2 que llevaban las revisiones 1 y 2, a el ATMega16U2 el cual su ventaja es una mayor FLASH de 8KB en la versión anterior, la ATMega8U2 a 16KB, lo que nos permitiría cargar programas más grandes en menor tiempo. Otro de los cambios es la adición de dos nuevos pines para evitar tapar las entradas analógicas A4 y A5 en caso de usar la comunicación I2C, que en concreto es usada por el registrador para el reloj RTC [8] y el conversor ADC MCP3424 [9]. La [fig47] muestra los pines utilizados por el registrador secundario.

## ATmega328 Pin Mapping

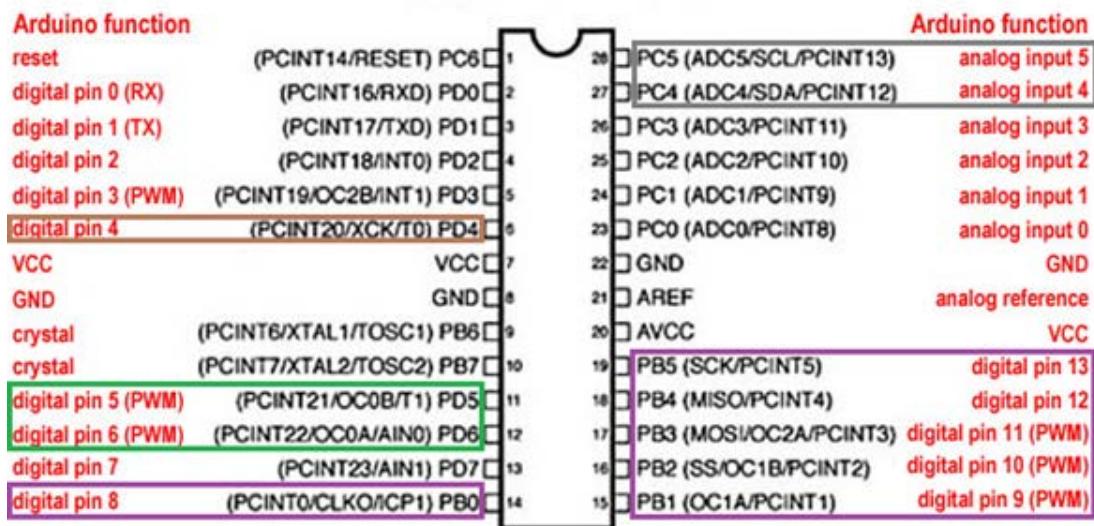


Figura 47: La siguiente imagen, muestra el pinout del ATMega328 usado por el registrador, diferenciado por colores para identificar los pines usados. Verde LEDS, marrón para temperaturas digitales, morado I/O digitales y módulo de expansión para microSD y gris para comunicación I2C para reloj DS3231 y conversar ADC MCP3424.

Esta medida de realizar el diseño con los componentes sin ensamblar, tiene como primera meta la reducción del consumo, ya que algunos como el regulador de voltaje lineal del que dispone Arduino, en concreto en esta revisión es el AMS1117 5.0, consume entre 5-10 mA en reposo, el conversor USB-UART ATMega16U2 con 4mA en modo inactivo, y varios componentes más que componen la placa, dando una suma de decenas de miliamperios, que aun siendo montados de forma independiente por ejemplo en una protoboard, tienen un consumo neto mucho más bajo. En la propia web de Arduino.cc [\[16\]](#) se detalla el procedimiento para realizar la extracción y montaje del microcontrolador, junto con los mínimos componentes necesarios para su funcionamiento [fig. \[48\]](#):

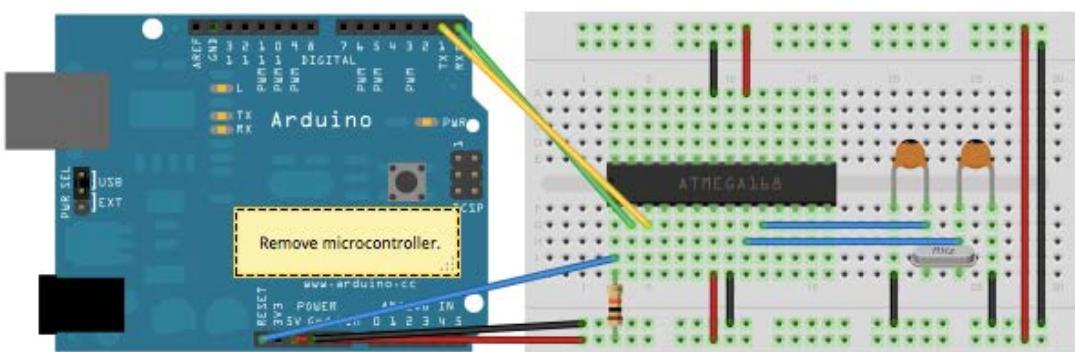


Figura 48: Montaje en protoboard del ATMega168 con los mínimos componentes, usando reloj externo, siendo necesario el uso de la placa de Arduino para la carga de los sketch, u otro programador FTDI o SPI.

Como se puede apreciar, se hace necesario del uso de la propia placa para poder cargar los programas a utilizar y alimentar el microcontrolador y el resto de componentes, o cualquier otro tipo de programador como uno por FTDI o SPI. Además de esta

configuración es posible realizar otra la cual se desprende del reloj externo de 16MHz, pero siendo necesario el cambio del bootloader y agregando a la IDE de Arduino la carpeta para reconocer la placa. Se procede a la descripción detallada del procedimiento de grabación del bootloader e instalación de los ficheros necesarios para el reconocimiento de la placa por parte del IDE de Arduino [38], dependiendo de la versión de programa usado.

El primer paso es el montaje con los componentes necesarios, y debido a que debemos de grabar un nuevo bootloader, es necesario de un programador SPI, ya que es el mejor método para poder grabarlo, donde el programador más usado es otro Arduino programado (valga la redundancia) como programador SPI. La siguiente figura ilustrativa nos indica el montaje en protoboard como para la grabación del bootloader [fig. \[49\]](#):

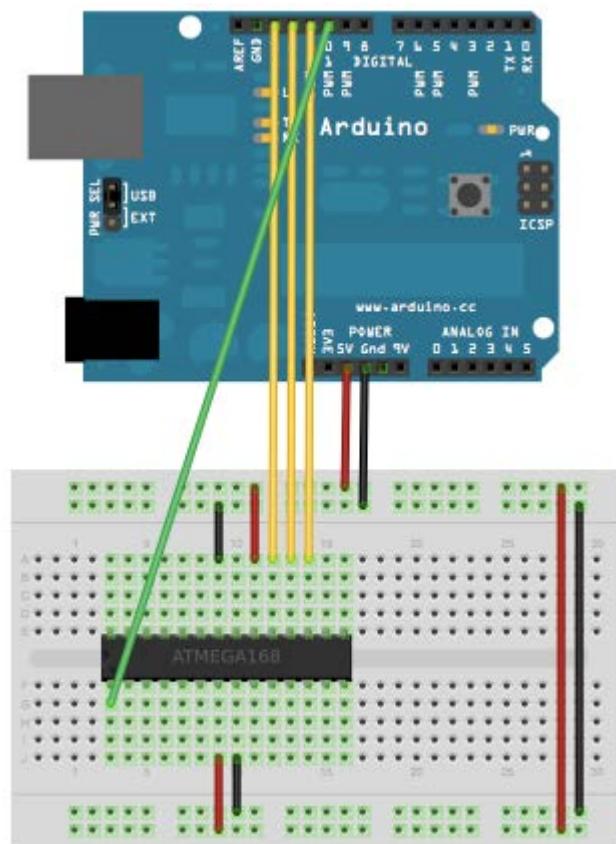


Figura 49: Montaje en protoboard del ATMega168 con los mínimos componentes, usando su propio reloj interno de 8 MHz.

Lo siguiente es la programación de la placa Arduino como programador SPI [fig. \[50\]](#).

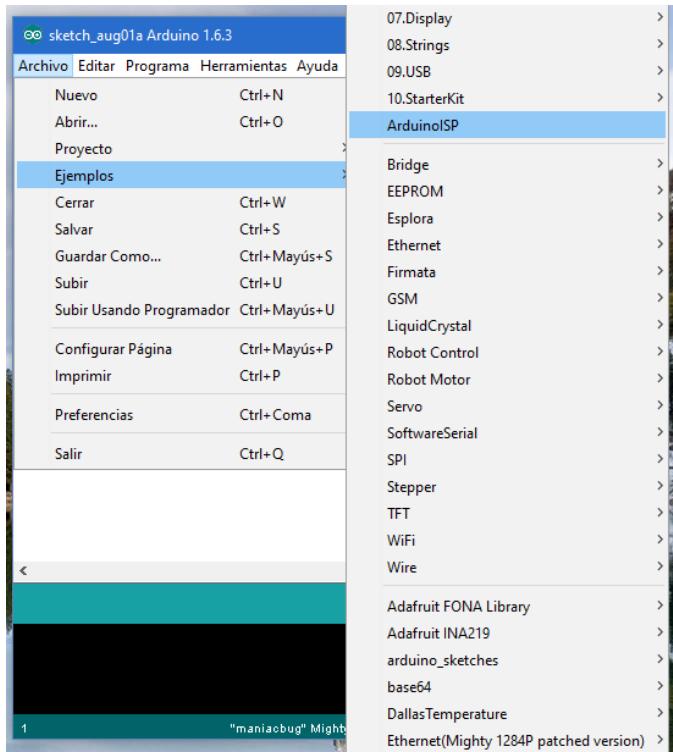


Figura 50: Se programa la placa de Arduino, como un programador SPI para escribir el bootloader en el montado en la protoboard.

Previa utilización del software, en función de la versión del IDE, deberemos descargar un fichero que agrega la placa al IDE, debido a que por defecto no dispone de esta versión. Para ello, nos tendremos que dirigir al directorio de instalación, y copiar la carpeta descargada, llamada “breadboard”, en la carpeta “hardware” del directorio raíz de instalación, y ya podremos hacer uso de ella [fig. \[51\]](#).

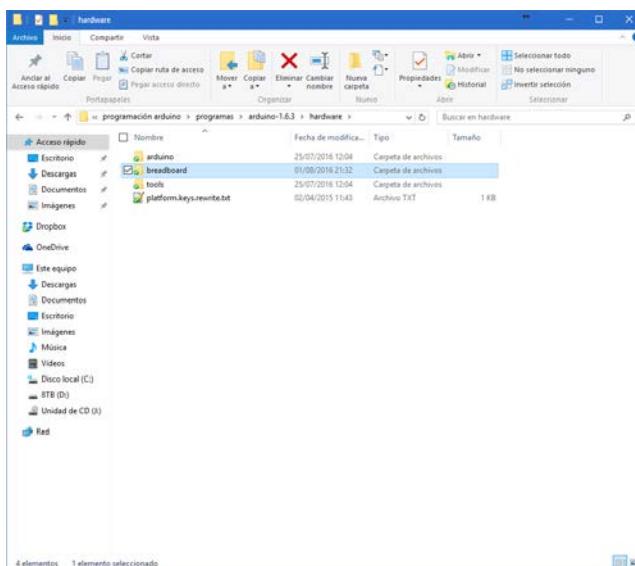


Figura 51: Ruta donde colocar la carpeta para agregar la compatibilidad del ATmega328p al IDE de Arduino, usando su reloj interno.

Por último, en la pestaña herramientas, quedaría seleccionar el tipo de placa que se va a programar, el tipo de programador usado y pulsar el botón de “Quemar Bootloader” para comenzar el proceso. Una vez finalizado está listo para subir cualquier programa desde cualquier programador FTDI o el mismo Arduino usado para grabar el bootloader como SPI. Si queremos usar el chip ATMega16U2 de la placa Arduino, primero debemos retirar el ATMega328p de la placa, ya que, si no, entrarían en conflicto [fig. \[52\]\[53\]\[54\]](#).

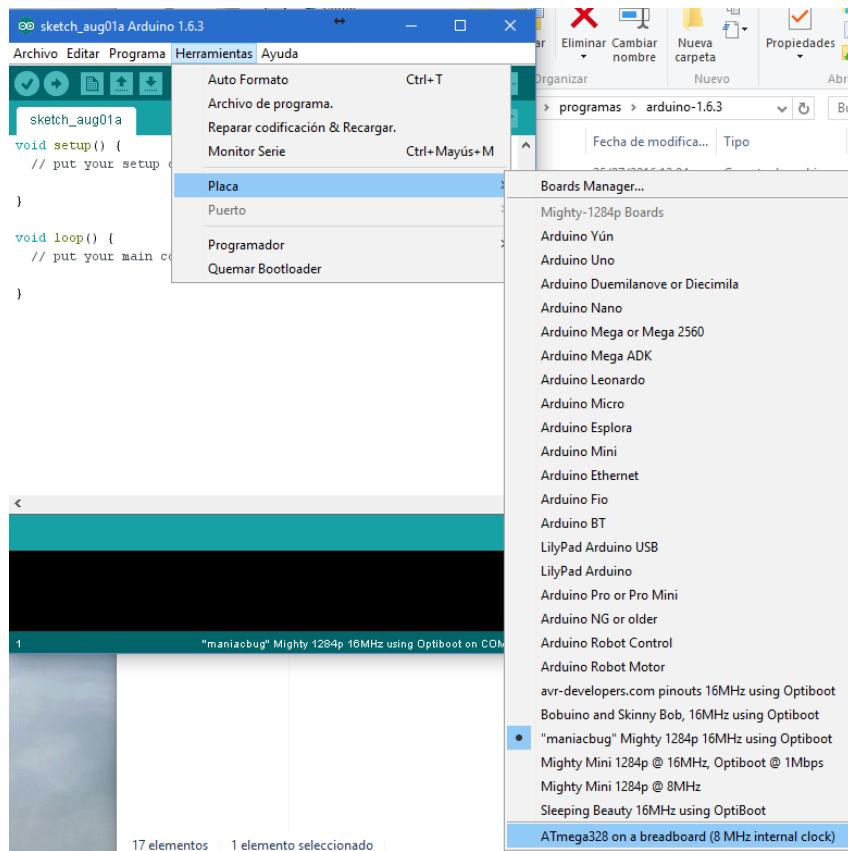


Figura 52: Selección de la placa, o en este caso del ATMega328p en protoboard, al que se realiza la grabación del bootloader.

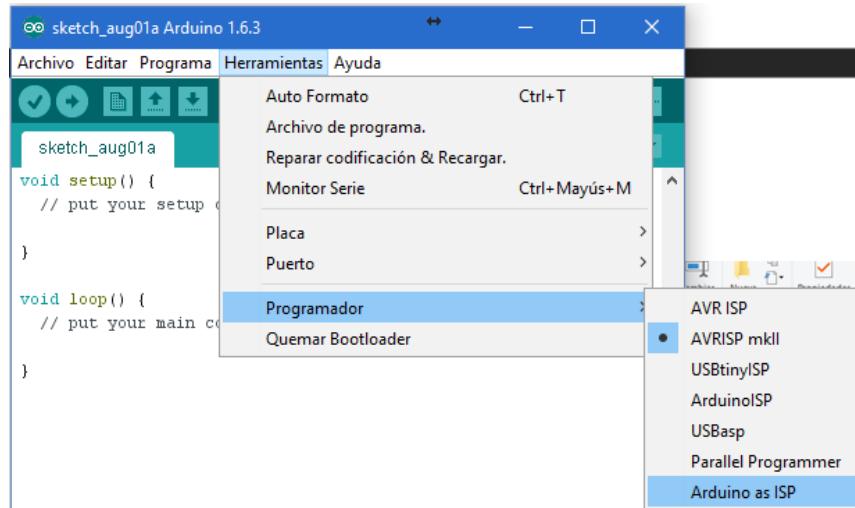


Figura 53: Se debe seleccionar el tipo de programador usado para quemar el bootloader.

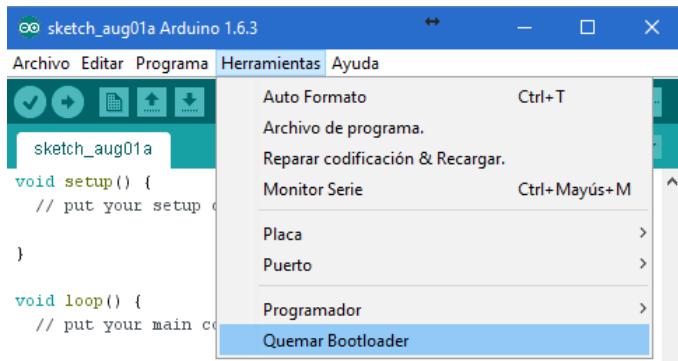


Figura 54: Inicio del proceso de grabación del bootloader.

Este tipo de solución hardware además de su fin más inmediato de reducir el consumo frente a las placas comerciales, también tiene como otros objetivos subyacentes el de reducir costes, ya que necesitamos menos componentes, o el de realizar proyectos mucho más personalizados en función de nuestras necesidades.

### 14.3 Desarrollo de hardware

En este apartado se expone un esquemático del diseño completo del proyecto utilizando el software proteus mostrado en la [fig55].

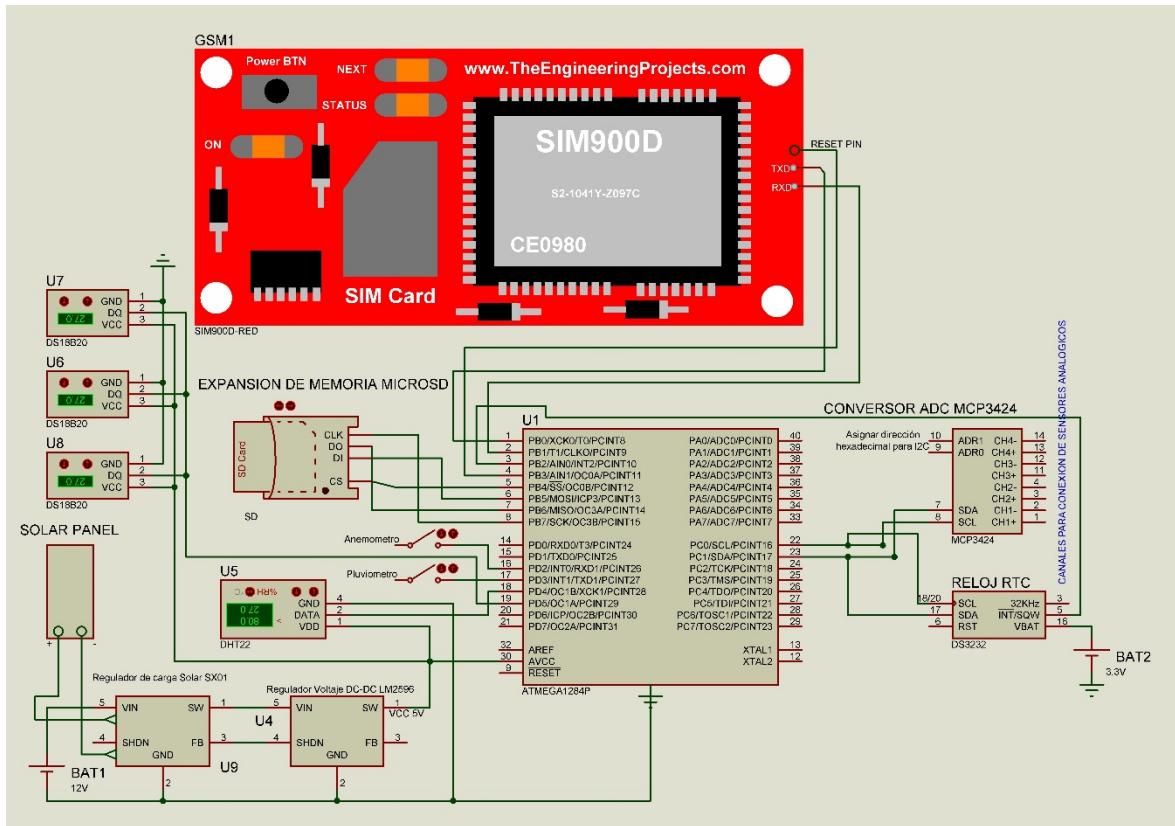


Figura 55: Esquemático completo del diseño del final.

Se puede apreciar que no se encuentran agregados los sensores analógicos en este boceto, ya que no son de importancia debido a que no necesitan de implementación software debido a que se toman sus valores de voltaje mediante el MCP3424 y posteriormente pueden ser usados una vez implantado en un trabajo real, como por ejemplo colocando un sensor de irradiancia, etc. Debido a la falta de librerías no se ha podido utilizar el módulo SIM800L en el boceto del diseño, sin embargo, se puede sustituir por el mostrado, debido a que utiliza una gran cantidad de comandos AT similares.

A continuación, se adjunta otro esquemático más real, montado en una protoboard, de cómo sería el diseño completo, utilizando para ello la herramienta fritzing. El diseño se puede apreciar con detalle en la [fig56].

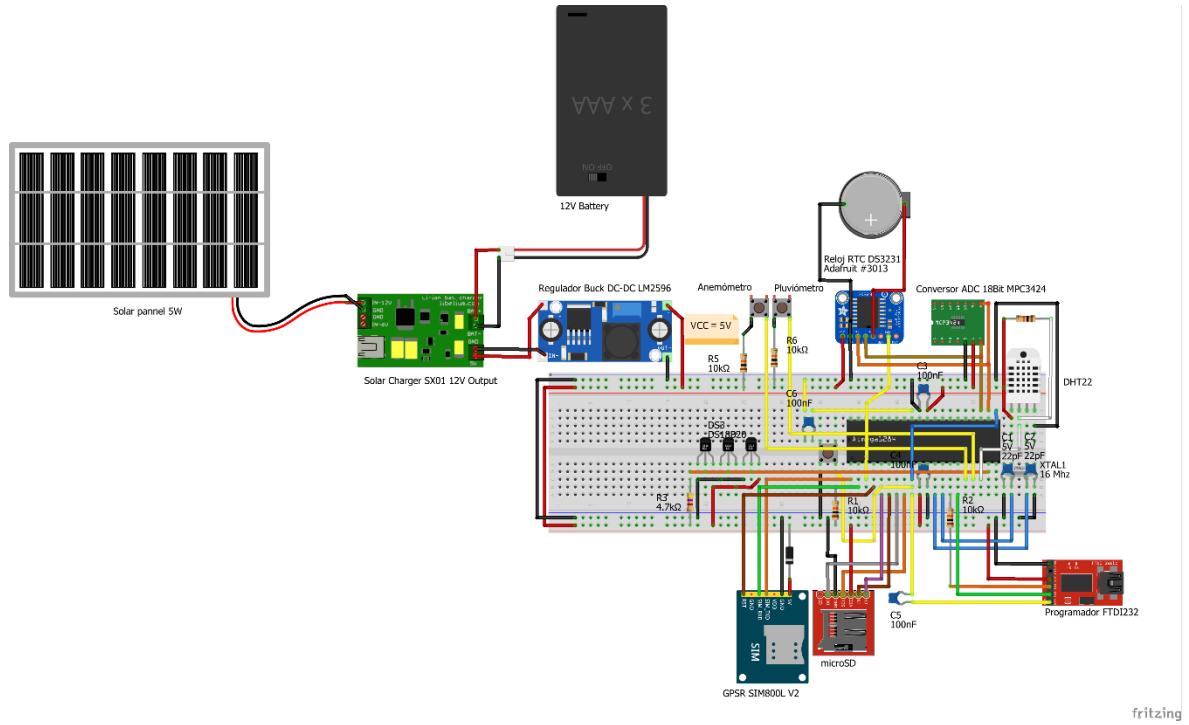


Figura 56: Prototipo del diseño final, utilizando una protoboard, mediante la herramienta fritzing

## 14.4 Desarrollo de software

### 14.4.1 Diagrama de flujo del software desarrollado

En el siguiente apartado, se adjunta un diagrama de flujo, tratando de facilitar la comprensión de forma visual del comportamiento del software desarrollado, ya que el mismo se encuentra estructurado por funciones, de forma que el `loop{}` (función que se ejecuta de forma infinita en ATMega1284p) quede lo más sencilla y clara posible como se muestra en la [fig57].

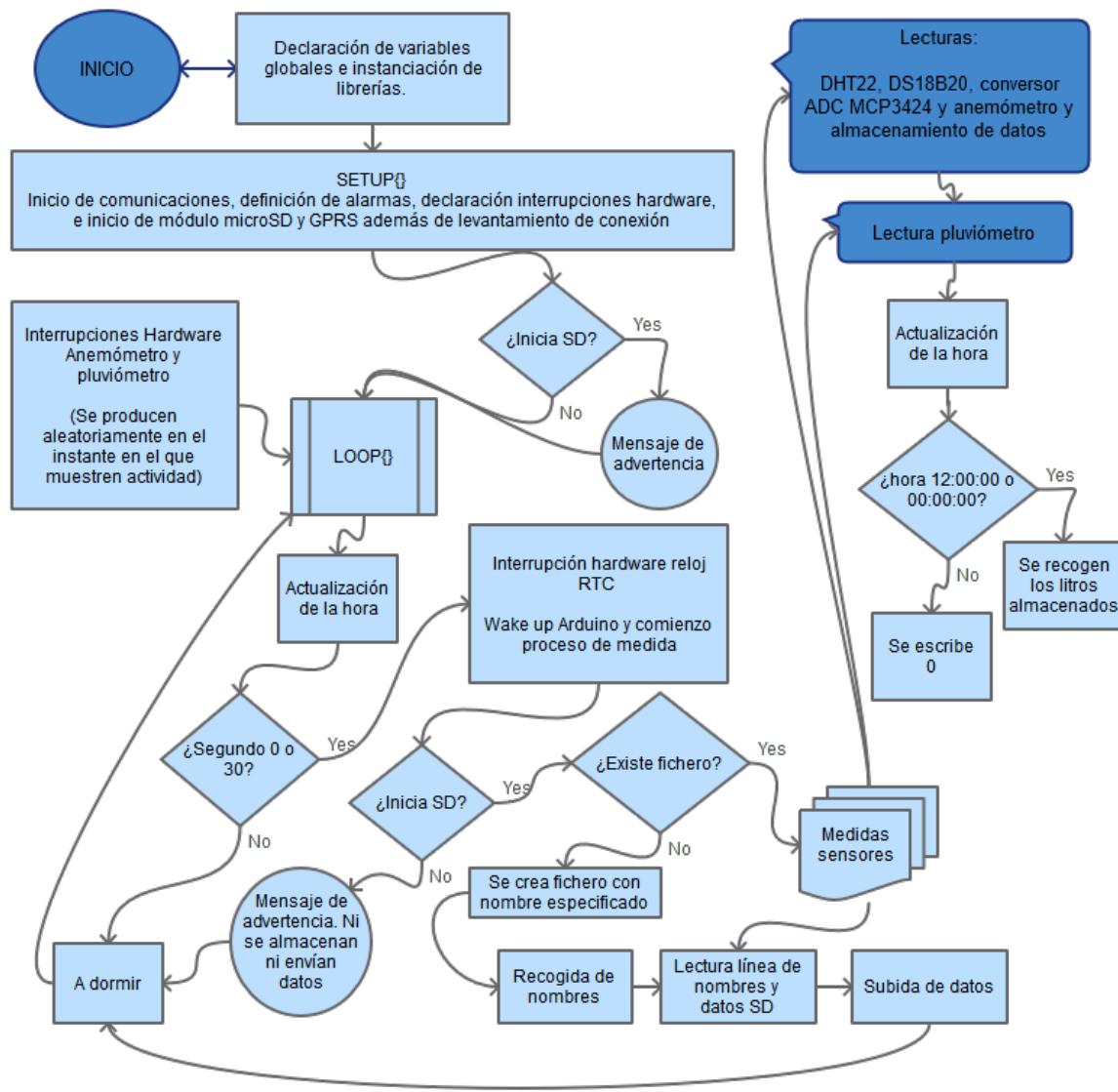


Figura 57: Diagrama de flujo del software de todo el proyecto, muy orientativo para una comprensión visual del mismo.

#### 14.4.2 Programa general

Este código compone el funcionamiento de todo el circuito armado, donde se realiza la lectura de sensores, puesta en hora del reloj RTC, almacenamiento y envío de los datos y el modo sleep de Atmel.

```

#include <SoftwareSerial.h>
SoftwareSerial sim800(0, 1); //RX,TX
#define rstpinSIM800L 3

//Libreria bajo consumo
#include <JeeLib.h> // sensor library from https://github.com/jcw/jeelib
/*
  
```

Como se puede apreciar, se iniciar el timer watchdog para generar un wakeup. Este solo es necesario en caso de necesitar el uso de delay, siempre y cuando en el delay se quiera verdaderamente parar el uso del microcontrolador, y recordando que en el estado powerdown, dejar de funcionar el resto de timer y varias partes de la MCU (mirar datasheet).

```

*/
ISR(WDT_vect) {
    Sleepy::watchdogEvent();
}

//Librerias para ADC MCP3424
#include "MCP3424.h"
// Configuration variables for MCP3424
byte address = 0x69; // address of this MCP3424. For setting address see datasheet -
//> Esta dirección hexadecimal es configurable
byte gain = 0; // gain = x1
byte resolution = 3; // resolution = 18bits, 3SPS
MCP3424 ADC1(address, gain, resolution); // create MCP3424 instance.

//Librerias SD
#include <SD.h>
#include <SPI.h>
File Archivo;
File litrosfile;
#define SDPIN 4
unsigned long pos;//Indica la posición del apuntador justo al terminar de escribir la
linea de datos.
struct datosHTTP
{
    String row = "";
    String nom = "";
};
datosHTTP valor;

//Librerías Reloj RTC y puerto I2C
#include <Wire.h>
#include <DS3231.h>
DS3231 clock;
RTCDateTime dt;
boolean isAlarm = false;
String str,anio,mes,dia,hora,minuto,segundo;

//Librerias DHT22
#define DHT22_PIN 12
#define DHT_VER DHT22
#include "DHT.h" //cargamos la librería DHT
DHT dht(DHT22_PIN, DHT_VER); //Se inicia una variable que será usada por
ATMega1284p para comunicarse con el sensor

//Librerias DS18B20
#include <OneWire.h> //Se importan las librerías
#include <DallasTemperature.h>
#define OneWirePin 13
OneWire ourWire(OneWirePin); //Se establece el pin declarado como bus para la
comunicación OneWire
DallasTemperature ds18b20(&ourWire); //Se instancia la librería DallasTemperature

//Anemómetro y Pluviómetro

```

```

/*
Declarar un variable como "volatile" ordenamos al compilador que mantenga
permanentemente presente el valor de la variable en la RAM y nos la suministre desde un
registro de almacenamiento, ROM.
*/
volatile int contaplumi = 0;
String litrosSD;
int litros = contaplumi;
long timeplumi = 0;
volatile int contaane = 0;
long timeane = 0 ;
byte flag=0;

//Pin para reiniciar el micro en caso de recibir errores en el envío de los datos del módulo
GPRS
int resetPin = 15;
byte numerrors=0;

void setup() {
  digitalWrite(resetPin, HIGH);
  delay(200);
  pinMode(resetPin, OUTPUT);
/*
Inicializamos este pin como salida y se coloca en nivel alto, ya que el proceso de
medida se inicia cuando este pin se coloca a nivel bajo
*/
  digitalWrite(18,HIGH);
  delay(200);
  pinMode(18,OUTPUT);
  Serial.begin(115200);
  pinMode(rstpinSIM800L, OUTPUT);
  rstSIM800L();

  clock.begin();
/*
Estas dos instrucciones desactivan el pin de reloj de 32Khz del reloj y la más importante
sería la segunda, ya que activa el pin SQW/INT, que es el pin capaz de generar una
onda cuadrada de varias frecuencias y la interrupción, pero es caso de no tener esta
línea, no se podría usar el pin como interrupción, ya que por defecto se activa la
generación de onda cuadrada a 1Hz
*/
  clock.enable32kHz(false);
  clock.enableOutput(false);

  // Set sketch compiling time
  //clock.setDateTime(__DATE__, __TIME__);
/*
Esta instrucción se comenta debido a que fija la hora en el reloj RTC en la cual
el compilador escribe el software en la flash del microcontrolador, y en caso de
un corte de energía o reinicio del mismo, volvería a escribir la misma hora anterior,
por lo que una vez fijada se comenta y se reescribe el software de nuevo en la flash.
*/
  /*
La función fijar hora, en primer lugar, realiza una conexión HTTP a una página web, esta
es leída en texto plano, tal cual lo haría un navegador web, una vez recibe la respuesta
del servidor para mostrar la página web. La página web es una página creada en este
proyecto, la cual sincroniza la hora de Madrid, una pequeña aplicación en PHP alojada en

```

<http://tfgvcr.esy.es/time.php> Una vez es leída, se recoge en un string el comando AT enviado, la respuesta del módulo GPRS y los caracteres leídos de la web. Gracias a que en caso de que la web haya sido leída de forma satisfactoria, el string siempre tendrá la misma longitud, esto nos da la posibilidad de poder dividirlo en substring que contendrán los datos de fecha y hora necesarios para poner en hora el reloj RTC. Una vez divididos en substring, se realiza una conversión string.toInt(), ya que el reloj necesita datos numéricos de tipo entero. Con esto se logra sincronizar nuestro reloj con la zona horaria elegida, desde Internet.

```
/*
fijarHora();

/*
Se fijan las alarmas que disparan la interrupción del reloj para comenzar el proceso de medida. La primera alarma se dispara en el segundo 30 de cada minuto y la segunda cada minuto.
*/
clock.setAlarm1(0, 0, 0, 30, DS3231_MATCH_S);
clock.setAlarm2(0, 0, 0, DS3231_EVERY_MINUTE);

dht.begin(); //Se inicia el sensor
ds18b20.begin();
pinMode(10, INPUT_PULLUP);
pinMode(11, INPUT_PULLUP);
pinMode(2, INPUT_PULLUP);
//Declaracion de las interrupciones por hardware
attachInterrupt( 0, ane, FALLING );//Esta interrupción hardware corresponde al pin 10 ATMega1284p
attachInterrupt( 1, pluvi, FALLING );//Esta interrupción hardware corresponde al pin 11 ATMega1284p
attachInterrupt( 2, alarm, FALLING );//Esta interrupción hardware corresponde al pin 2 ATMega1284p
iniciaSD();
/*
Aquí, además de leer los litros almacenados en la SD, se comprueba primero si existe el fichero de los datos en la SD, en caso de no existir, esta función acaba, en caso contrario, realiza una petición http al server para comprobar si existe el fichero en él, en caso de estarlo, la función termina, en caso contrario se lee la línea de nombres en la SD y es enviada mediante una petición GET. En caso de no recibir una respuesta similar a "SI" o "NO", significa un comportamiento extraño o producido por el módulo GPRS, por el microcontrolador ATMega1284p, o un fallo en la conexión al servidor, lo que provoca un reinicio del micro para volver a intentar realizar la comprobación.
*/
leerlitros();
leerlitros();
existe();
}

void rstsIM800L(){
digitalWrite(rstpinSIM800L, HIGH);
delay(10);
digitalWrite(rstpinSIM800L, LOW);
delay(100);
digitalWrite(rstpinSIM800L, HIGH);
delay(100);
Sleepy::loseSomeTime(14900);
```

```

sim800.begin(115200); //Esto se ha hecho porque al reiniciar el módulo, por defecto
inicia con un baudrate fijo a 115200, una marcado, en la siguiente instrucción indicamos
el baudrate de trabajo que queremos;
sim800.println("AT+IPR=57600"); //indicamos al módulo GSM el baudrate que
queremos
sim800.flush(); //Una vez cambiado el baudrate del módulo GSM, liberamos el buffer
serie
delay(2);
sim800.end(); //Con esto finalizamos el bus y lo reiniciamos a que esos pines
utilizados para serie tengan un uso normal
sim800.begin(57600); //Volvemos a inicializar la comunicación serie a la velocidad
fijada en el módulo GSM
sim800.println("AT+SAPBR=3,1,\"CONTYPE\", \"GPRS\"");
disponible();
sim800.println("AT+SAPBR=3,1,\"APN\", \"lowi.private.omv.es\"");
disponible();
sim800.println("AT+SAPBR=1,1"); //Levantamos la conexión GPRS(Imprescindible)
wait();
sim800.println("AT+HTTPINIT");
disponible();
sim800.flush();
delay(100);
Sleepy::loseSomeTime(14900);
}

void iniciaSD() {
    if (!SD.begin(SDPIN)) {
        pinMode(SDPIN, OUTPUT);
    }
}
/*
Las dos funciones siguientes, se han creado para leer y almacenar los litros del
pluviómetro en el fichero litros.txt. Estas funciones se han creado por dos razones.
La primera es evitar la pérdida de los litros, ya que como se expone en la memoria, esta
medida se realiza cada 12 horas, y en caso de un corte de alimentación por falta de
batería, no se perderían ya que se almacenan en la microSD y serán leídos al reiniciar.
El segundo motivo es que se ha programado la función que comprueba si los datos se
envían correctamente al servidor, y en caso de dos errores consecutivos, se produce un
reinicio por hardware, debido a esto podrían perderse los litros almacenados, pero con
estas funciones no sucedería, ya que serían leídos una vez se reinicia el dispositivo.
*/
void leerlitros(){
    //char array que almacena hasta 99999 por seguridad se ha sobredimensionado
    char c[5];
    byte i=0;
/*
Aquí, se comprueba si el fichero existe, en caso de no existir lo crea y escribe 0, y en
caso contrario se leen los litros y se igualan con la variable volatile que recoge los litros
mediante la interrupción hardware que produce el pluviómetro.
*/
    if (!SD.exists("litros.txt")){
        litrosfile = SD.open("litros.txt", FILE_WRITE);
        if (litrosfile){
            litrosfile.print('0');
            litrosfile.close();
        }else{
            Serial.println("El archivo litros.txt no se abrio correctamente");
        }
    }
}

```

```

}else{
    litrosfile = SD.open("litros.txt");
    if (litrosfile){
        while (litrosfile.available()){
            c[i]=litrosfile.read();
            i++;
        }
    }
    //La función atoi se utiliza para transformar el char array con los litros leídos de la
    //microSD en tipo de dato entero
    litros=atoi(c);
    litrosfile.close();
}else{
    Serial.println("El archivo litros.txt no se abrio correctamente");
}
contapluvi=litros;
}

void escribirlitros(){
    SD.remove("litros.txt");
    litrosfile = SD.open("litros.txt", FILE_WRITE);
    if (litrosfile){
        litrosfile.print(litros);
        litrosfile.close();
    }else{
        Serial.println("El archivo litros.txt no se abrio correctamente");
    }
}

void existe(){
if (SD.exists("datos.csv")){
    sim800.println("AT+HTTPPARA=\\"CID\\",1");
    disponible();
    sim800.println("AT+HTTPPARA=\\"URL\\",\\"http://tfgvcr.esy.es/existe.php\"");
    ;
    disponible();
    sim800.println("AT+HTTPACTION=0"); //Envía los datos al servidor
    wait();
    sim800.println("AT+HTTPREAD");
    delay(100);
    String response="";
    while(sim800.available()){
        response = sim800.readString();
    }
    String res=response.substring(28,response.length()-6);
    if (res=="SI" || res=="NO"){
        if(res=="NO"){
            char letra;
            //Se vuelve a abrir el fichero, esta vez para leer los datos escritos.
            Archivo = SD.open("datos.csv");
            //Si el archivo se ha abierto correctamente se muestran los datos.
            if (Archivo){
                Archivo.seek(0);
                while(Archivo.available()){
                    if( Archivo.peek() == '\n' ){goto end;}
                    letra=Archivo.read();
                    valor.nom+=String(letra);
                }
            end:
                Archivo.close();
            }else{

```

```

        Serial.println("El archivo datos.csv no se abrio correctamente");
    }
sim800.println("AT+HTTPPARA=\\"CID\\",1);
respuesta();

sim800.println("AT+HTTPPARA=\\"URL\\",\\"http://tfgvcr.esy.es/datos.php?dato
s="+valor.nom+"\\"");
disponible();
sim800.println("AT+HTTPACTION=0");//Envía los datos al servidor
wait();
sim800.println("AT+CSCLK=2");
disponible();
valor.nom="";
}
}else{
delay(10);
digitalWrite(resetPin, LOW);
}
response="";
res="";
}
}

void uploaddata() {
sim800.println("AT");//check AT
disponible();
sim800.println("AT+HTTPPARA=\\"CID\\",1);
//Leemos las respuestas del módulo GRPS para saber si se han enviado los datos o no
respuesta();

sim800.println("AT+HTTPPARA=\\"URL\\",\\"http://tfgvcr.esy.es/datos.php?dato
s="+valor.row+"\");//Peticion GET
disponible();
sim800.println("AT+HTTPACTION=0");//Envía los datos al servidor
wait();
sim800.println("AT+CSCLK=2");//Modo autosleep para el módulo GPRS SIM800L,
el cual una vez deja de recibir comandos AT, se coloca en sleep
disponible();
//Con esto limpiamos las variables que almacenan el nombre y los datos para volver a
tomar valores nuevos.
valor.row = "";
}

/*
Esta función identifica cualquier dispositivo conectado al bus 1-Wire no tiene por qué ser
solo para el sensor de temperatura DS18B20, pero en este caso está adaptada para dar
sus IDs
*/
void idDS18B20() {
byte numsensor = 0;
byte i;
byte addr[8];
String DatosHEX[8];
while (ourWire.search(addr)) {
    for (i = 0; i < 8; i++) {
        DatosHEX[i] = String(addr[i], HEX);
    }
    if (OneWire::crc8(addr, 7) != addr[7]) {

```

```

        numsensor = 0;
        Serial.print("CRC is not valid!\n");
        return;
    }
    numsensor++;
    String numerosensores = "S" + String(numsensor);
    Archivo.print(numerosensores + "_");
    if (DatosHEX[7].length() == 1) {
        DatosHEX[7] = "0" + DatosHEX[7];
    }
    DatosHEX[7] = "0x" + DatosHEX[7];
    Archivo.print(DatosHEX[7] + ",");
}
ourWire.reset_search();
return;
}

void names() {
//      ESCRIBIENDO DATOS EN LA MEMORIA SD
//Se abre el documento sobre el que se va a leer y escribir.
    Archivo = SD.open("datos.csv", FILE_WRITE);
    pos = Archivo.position();/*Se utiliza para almacenar la posición donde comienza
a escribir para después el apuntador comenzar a leer en la misma posición que comenzó
la escritura. Con esto, seremos capaces de leer solamente la última línea que contiene
los nuevos datos.*/
//Se comprueba que el archivo se ha abierto correctamente y se procede a escribir en él.
    if (Archivo) {
        //Se escribe información en el documento de texto datos.csv.

        Archivo.print("Fecha,(DHT22)Humedad,Temperatura,(DS18B20)_Temperatura_");
        idDS18B20();
        Archivo.print("MCP3424(CH1),(CH2),(CH3),(CH4),");
        Archivo.print("Anemometro(m/s),");
        Archivo.print("Pluviometro(L/m2_dia)");
        Archivo.println(",");
//Se cierra el archivo para almacenar los datos.
        Archivo.close();
    }
}

void leerultimalinea() {
// LEYENDO DATOS EN LA MEMORIA SD
    char letra;
//Se vuelve a abrir el fichero, esta vez para leer los datos escritos.
    Archivo = SD.open("datos.csv");

//Si el archivo se ha abierto correctamente se muestran los datos.
    if (Archivo) {
        Archivo.seek(pos);/*Posición donde comienza a leer el apuntador, que en este
caso es la posición donde se comienzan a escribir los datos, con esto se consigue leer
siempre los nuevos datos, que corresponde a la última línea.*/
        while (Archivo.available()) {
            //Se escribe la información que ha sido leída del archivo.
            letra = Archivo.read();
            valor.row += String(letra);
        }
        Archivo.close();
    }
}

```

```

        }
    }

    Serial.println("El archivo datos.csv no se abrio correctamente");
}

void leerDHT22() {
    //Leyendo datos sensor DHT22
    float h = dht.readHumidity(); //Se lee la humedad
    float t = dht.readTemperature(); //Se lee la temperatura

    //      ESCRIBIENDO DATOS EN LA MEMORIA SD

    //Se abre el documento sobre el que se va a leer y escribir.

    Archivo = SD.open("datos.csv", FILE_WRITE);
    if (Archivo) {
        //Se escribe información en el documento de texto del sensor DHT22
        Archivo.print(h);
        Archivo.print(",");
        Archivo.print(t);
        Archivo.print(",");
    }
    //Se cierra el archivo para almacenar los datos.
    Archivo.close();
}

//En caso de que haya habido problemas abriendo datos.csv, se muestra por pantalla.
else {
    Serial.println("El archivo de datos no se abrio correctamente");
}

//      FIN DE LA ESCRITURA DE DATOS EN LA MEMORIA SD
}

void leerDS18B20() {
    //Se abre el documento sobre el que se va a leer y escribir.
    Archivo = SD.open("datos.csv", FILE_WRITE);

    //Se comprueba que el archivo se ha abierto correctamente y se procede a escribir en él.
    if (Archivo) {
        byte numsensor = 0;
        byte addr[8];//Array que recibe la dirección de cada uno de los sensores del bus
OneWire
        while (ourWire.search(addr)) {
            ds18b20.setResolution(addr, 12);//Le indicamos la resolución que queremos
obtener
            ds18b20.requestTemperatures();//Preparamos al sensor para medir
            float temp = ds18b20.getTempCByIndex(numsensor);
            Archivo.print(temp);
            numsensor++;
            Archivo.print(",");
        }
        ourWire.reset_search();
        Archivo.close();
    }
    //Se muestra por el monitor que los datos se han almacenado correctamente.
}

```

```

    else {
        Serial.println("El archivo datos.csv no se abrio correctamente");
    }
    return;
}

void readMCP3424() {
    Archivo = SD.open("datos.csv", FILE_WRITE);
    if (Archivo) {
        for (byte i = 0;i<4;i++) {
            // get channel data with function
            double CH = ADC1.getChannelmV(i);
            Archivo.print(CH);
            Archivo.print(",");
        }
        Archivo.close();
    }
    else {
        Serial.println("El archivo datos.csv no se abrio correctamente");
    }
}

void readane() {
    detachInterrupt(digitalPinToInterrupt(10));/*Desactiva la interrupción del pin
    del micro(no el número de interrupción), que este caso equivale al pin 10 del
    ATMega1284p e interrupción 0*/
/*
Realizamos la conversión de nuestro número de clicks(cada click es como pulsar un
pulsador). Primero contamos el número de clicks y lo dividimos entre dos, ya que cada
dos clicks es una vuelta completa del anemómetro que también es igual a 1Hz. Lo
siguiente para realizar el cálculo en mi caso voy a realizar el conteo en un espacio
temporal de 30 segundos y pasado este tiempo resetearemos el contador a 0. Pues bien,
cuando sabemos el número de vueltas lo dividiremos entre el tiempo de conteo y con
esto tenemos el número de vueltas (o Hertzios) segundo. Ya por último nos quedaría
dividir entre 2.5Hz que equivale a 1m/s según nuestra hoja de características y con esto
ya obtendríamos la velocidad.
*/
    float freq = contaane / 2;// Frecuencia o número de vueltas
    float velm = freq / (30 * 2.5); //Maximo 40m/s(144Km/h).
    float velkm = velm*3.6;
    Archivo = SD.open("datos.csv", FILE_WRITE);
    if (Archivo) {
        Archivo.print(velkm);
        Archivo.print(",");
        Archivo.close();
    }
    else {
        Serial.println("El archivo datos.csv no se abrio correctamente");
    }
    contaane = 0;
    attachInterrupt(0, ane, FALLING); //Activa de nueva la interrupcion 0(Pin 10
    ATMega1284p)
}

void readpluvi() {

```

```

detachInterrupt(digitalPinToInterrupt(11));/*Desactiva la interrupción del pin
del micro (no el número de interrupción), que este caso equivale al pin 11 del
ATMega1284p e interrupción 1*/
if (flag != 0) {
    Archivo = SD.open("datos.csv", FILE_WRITE);
    if (Archivo) {
        Archivo.print(contapluvi);
        Archivo.println(",");
        Archivo.close();
        contapluvi = 0;
        litros=contapluvi;
    }
    Cuando son las 0 o 12 en punto, se envían los datos recogidos por el pluviómetro y se
    reinicia el contador además de reiniciarla y escribirlo en el fichero litros.txt, que también
    almacena los litros por seguridad
}
    escribirlitros();
    flag = 0;
}
else {
    Serial.println("El archivo datos.csv no se abrio correctamente");
}
else {
    Archivo = SD.open("datos.csv", FILE_WRITE);
    if (Archivo) {
        Archivo.print("0");
        Archivo.println(",");
        Archivo.close();
    }
    else {
        Serial.println("El archivo datos.csv no se abrio correctamente");
    }
}
attachInterrupt(1, pluvi, FALLING); //Activa de nueva la interrupción 1(Pin 11
ATMega1284p)
}

void writeDateTime() {
    Archivo = SD.open("datos.csv", FILE_WRITE);
    pos = Archivo.position(); /*Se utiliza para almacenar la posición donde comienza
    a escribir para después colocar el apuntador en la misma posición que comenzó la
    escritura. Con esto, seremos capaces de leer solamente la última línea que contiene los
    nuevos datos. Se comprueba que el archivo se ha abierto correctamente y se procede a
    escribir en él.*/
    if (Archivo) {
        //Se escribe información en el documento de texto del sensor DHT22
        dt = clock.getDateTime();
        Archivo.print(clock.dateFormat("d-m-Y_H:i:s,", dt));
        //Se cierra el archivo para almacenar los datos.
        Archivo.close();
    }
}

//En caso de que haya habido problemas abriendo datos.csv, se muestra por pantalla.
else {
    Serial.println("El archivo de datos no se abrio correctamente");
}

```

```

        }

    }

void tomarlecturas() {
    iniciaSD();
    litros=contapluvi;
/*
En cada lectura, por seguridad, los datos que contiene la variable volatile encargada de
recoger los litros del pluviómetro, son escritos en el fichero
*/
    escribirlitros();
/*
Condición que comprueba si el fichero existe y en caso de no encontrarse, se crea y se
escribe la línea de nombre, y en caso contrario, se escriben los datos
*/
    if (!SD.exists( "datos.csv" )) {
        names();
    }
    else {
        writeDateTime();
        dt = clock.getDateTime();
        if (((dt.hour == 12) || (dt.hour == 0)) && (dt.minute == 0) &&
(dt.second == 0)) {
            flag++;
        }
        leerDHT22();
        leerDS18B20();
        readMCP3424();
        readane();
        readpluvi();
    }
/*
Aquí, serán leídas la línea de nombre ya la nueva línea de datos generada al leer todos
los dispositivos, y está queda lista para ser enviada.
*/
    leerultimalinea();
}
//Interrupcion del reloj RTC DS3231 para disparar las medidas
void alarm()
{
    isAlarm = true;
}

/*
Función que lee la hora desde una página web en internet, a través del módulo GPRS
que realiza una conexión HTTP
*/
void fijarHora(){
sim800.println( "AT+HTTPPARA=\\"CID\\",1" );
disponible();
sim800.println( "AT+HTTPPARA=\\"URL\\",\"http://tfgvcr.esy.es/time.php\"" );
disponible();
sim800.println( "AT+HTTPACTION=0" );//Envía los datos al servidor
wait();
sim800.println( "AT+HTTPREAD" );
espera_hora();

/*

```

Esta condición, comprueba si la fecha ha sido leída correctamente del servidor ya que, en caso de no ser leído correctamente, el string estará vacío y si esto sucede procederemos al reinicio para intentar de nuevo la sincronización.

```
/*
String checkanio=anio.substring(0,1);
if(checkanio!="2"){
delay(10);
digitalWrite(resetPin, LOW);
}
clock.setDateTime(anio.toInt(), mes.toInt(), dia.toInt(), hora.toInt(),
minuto.toInt(), segundo.toInt());
}
/*
```

Función que lee el string con el comando AT, la respuesta del módulo GPRS y los datos leídos desde la web, y gracias a que siempre tienen la misma longitud, a continuación son subdivididos en substring que almacenaran los datos de fecha y hora para ser fijado al reloj RTC.

```
/*
void espera_hora() {
delay(100);
while (sim800.available()) {
str = sim800.readString();
}
anio = str.substring(29, 33);
mes = str.substring(34, 36);
dia = str.substring(37, 39);
hora = str.substring(40, 42);
minuto = str.substring(43, 45);
segundo = str.substring(46, 48);
}
```

```
/*
Estas dos funciones se utilizan para esperar a la escucha del puerto serie, para la recibir los datos del buffer, los cuales son los datos de respuesta del modulo GPRS
```

```
/*
void wait() {
delay(3000);
while (sim800.available()) {
Serial.write(sim800.read());
}
}
```

```
/*
La función respuesta, lee la respuesta del comando AT+HTTPPARA="CID",1, ya que, en caso de fallo, el módulo GPRS no envía los datos al servidor. En caso de que la respuesta sea errónea X veces (en este caso se ha optado por 2 errores consecutivos), el programa genera un reinicio por hardware del dispositivo.
```

```
/*
void respuesta(){
delay(100);
String response="";
while(sim800.available()){
response = sim800.readString();
}
Serial.println(response);
String errores=response.substring(22,response.length()-2);
if(errores!="OK"){
numerrors++;
if(numerrors==2){
```

```

    delay(10);
    digitalWrite(resetPin, LOW);
}
}else{
numerrors=0;
}
response="";
errorres="";
}

void disponible() {
delay(100);
while (sim800.available()) {
Serial.write(sim800.read());
}
}

void loop() {
dt = clock.getDateTime();
if (dt.second != 0 || dt.second != 30) {
Sleepy::powerDown();
}
if (isAlarm)
{
/*
Esto desactiva la alarma para evitar que al disparar la interrupción bloquee el
microcontrolador, ya que este pin pasa de nivel bajo a alto al dispararse la alarma, y con
esto, vuelve a nivel bajo.
*/
clock.clearAlarm1();
clock.clearAlarm2();
dt = clock.getDateTime();
if (dt.second == 0 || dt.second == 30) {
/*
Al colocar el pin 15 a nivel bajo, la placa Arduino UNO R3 encargada de realizar las
medidas de corriente, donde se encuentra el sensor de corriente Ina219, comenzará a
leer la corriente y colocará una marca cada 30s para dividir el proceso de lectura
completo y poder obtener varias medidas para sacar las posteriores conclusiones.
*/
digitalWrite(18,LOW);
tomarlecturas();
uploaddata();
}
}
}

/*
Las funciones pluvi y ane, son las funciones que se ejecutan en las interrupciones
hardware, y se encargan de contar los click del anemómetro y pluviómetro. Estas dos
funciones, poseen un sistema antirebote para evitar los posibles falsos contactos
generados en la unión de los metales.
*/
void pluvi()
{
if (millis() > timepluvi + 5)
{
contapluvi++;
timepluvi = millis();
}
}

```

```

}

void ane()
{
    if (millis() > timeane + 5)
    {
        contaane++;
        timeane = millis();
    }
}

```

#### 14.4.3 Toma de corrientes

Este software se ha desarrollado, para realizar la lectura de la corriente consumida por todo el circuito, el cual permite tomar lecturas de forma automatizada cada 100 ms, aumentando los datos de lectura para apreciar con mayor detalle el consumo, en caso de posibles fluctuaciones del mismo, y con estos datos poder observar la mejora y además desarrollar una gráfica. El método consiste en que la placa que toma las medidas, cambia el estado de uno de sus pines de nivel alto a bajo, disparando así el proceso de medida de corriente. Para poder separar el proceso de completo que tarda, se crea un salto de línea con la palabra “Marca”, separando así las medidas para realizar un estudio posterior del consumo total.

```

//El sensor ina219 trabaja con el protocolo I2C, por lo que se debe usar la librería Wire
para trabajar con 1-Wire
#include <Wire.h>
#include <Adafruit_INA219.h>
Adafruit_INA219 ina219;
long tiempo = 0;

void setup() {
    //Se inicializa la comunicación Serial y del sensor de corriente
    Serial.begin(115200);
    ina219.begin();
    pinMode(8, INPUT_PULLUP);
/*
Indicamos la resolución, teniendo previamente unos conocimientos aproximados del
voltaje y consumo del circuito que se quiere medir, ya que, con el comando posterior,
como se puede apreciar se ha aumentado la resolución reduciendo los rangos de medida
de voltaje y corriente que tiene el sensor que por hoja de características están entre 0-
32V y 3.2A
*/
    ina219.setCalibration_16V_400mA();
}

//Inicialización de variables
float current_mA = 0;

//Intervalo entre toma de medidas
void loop() {

```

```

        while (digitalRead(8) == LOW) {
            if (millis()<tiempo + 30000) {
                tiempo = millis();
                Serial.println("Marca");
            }
            //Toma de medida de corriente
            current_mA = ina219.getCurrent_mA();
            Serial.print(current_mA);    Serial.print(" ");
            //Intervalo de demora entre cada medida
            delay(100);
        }
    }
}

```

#### 14.4.4 Aplicación web PHP para procesado de datos

Este software se instala en nuestro servidor web y será el encargado de recibir los datos que envía ATMega1284p vía GPRS para ser procesados y almacenados siguiendo el mismo criterio que utiliza al almacenarlos en la tarjeta microSD, para así disponer de una copia exacta. Además, genera una url con los datos obtenidos para reenviarlos a la web de <https://emoncms.org>, para poder generar las gráficas y visualizarlos. Para realizar el envío hay que recordar indicar la APIKEY. Esta será facilitada una vez realizamos el registro en la web de nuestro usuario.

```

<?php
$nombre_fichero = 'datos.csv';

if (file_exists($nombre_fichero)) {
/*
La función explode separa la línea de datos y nombres en un array, guardando cada
carácter en cada posición de ese array hasta encontrar el carácter separador, que en este
caso se ha elegido una ","
*/
$arraydatos = explode(", ", $_GET['datos']);
$aux=$_GET['datos'];
$aux.=" ";
if($f = fopen('datos.csv', 'r')) {
    $lineanombres = fgets($f);
    fclose($f);
}
$arraynombres = explode(", ", $lineanombres);
/*
Aquí se comprueba si la línea de datos y nombres es igual, y en caso de serlo finaliza el
programa. Esto se comprueba para el caso en el que se sustituya la SD o se elimine el
fichero de la SD, ya que el criterio que sigue es que, si el fichero no existe en la tarjeta, se
crea y la primera línea es la de nombres, y para evitar volver a escribirla en el fichero del
servidor, se realiza esta comprobación
*/
if (!strcmp($aux, $lineanombres)) {
    goto end;
}

```

```

}

$file = fopen("datos.csv", "a") or die("Imposible abrir fichero");
if($_GET['datos'] == "")
{
echo "No ha introducido ningun valor.";
}
else
{
fwrite($file, $_GET['datos'].PHP_EOL);
fclose($file);
}

} else {


```

```

$file = fopen("datos.csv", "a") or die("Imposible abrir fichero");
if($_GET['nombres'] == "")
{
echo "No ha introducido ningun valor.";
}
else
{
/*

```

En este caso, el fichero al no existir, es creado y se escribe la línea de nombres. Como en cada petición GET, el micro manda tanto la primera línea de nombre como la de los datos, si en la SD el fichero ya existiera, al mandar tantos datos como nombres, se escriben las dos.

```

*/
fwrite($file, $_GET['nombres'].PHP_EOL);
fclose($file);
}

$arraydatos = explode(", ", $_GET['datos']);
$aux=$_GET['datos'];
$aux.=" ";

```

```

if($f = fopen('datos.csv', 'r')){
    $lineanombres = fgets($f);
    fclose($f);
}

```

```

$arraynombres = explode(", ", $lineanombres);

```

```

if (!strnatcmp($aux, $lineanombres)) {
    goto end;
}
/*

```

Aquí, es donde se escribiría la segunda fila con los datos, después de crear el fichero y escribir la primera línea de nombres.

```

*/
$file = fopen("datos.csv", "a") or die("Imposible abrir fichero");
if($_GET['datos'] == "")
{
echo "No ha introducido ningun valor.";
}
else
{
fwrite($file, $_GET['datos'].PHP_EOL);
fclose($file);
}
```

```

/*
Aquí, se genera la url que contiene los datos y el nombre que identifica a cada dato, para
así, almacenarlos correctamente en la base de datos que serán leídas por los gráficos. El
bucle comienza a contar desde 1, ya que el primer dato es la fecha, y no es necesario ya
que, al recibirlas, coge su propia marca temporal, por tanto, es omitida.
*/
}
$url="http://emoncms.org/input/post.json?apikey="aquí se incluye la apikey de
lectura/escritura, facilitada por la web de emoncms, una vez estamos
registrados"&node=1&json={ ";
for ($i = 1; $i < count($arraynombres)-1; $i++) {
    if($i<count($arraynombres)-2){
        $url.=$arraynombres[$i].":".$arraydatos[$i].",";
    }
    else{
        $url.=$arraynombres[$i].":".$arraydatos[$i];
    }
}
$url.="}";
/*
Con esto, logramos la ejecución de la url que envía los datos a emonCMS
*/
file_get_contents($url);
end:
?>

```

#### 14.4.5 Aplicación web PHP para sincronización hora reloj RTC

Esta pequeña aplicación, tan solo fija la zona horaria de interés, que este caso ha sido la de Madrid por encontrarnos en España y es mostrada por pantalla. En caso de utilizarse en otra región, deberá indicarse en el código:

```

<?php
date_default_timezone_set("Europe/Madrid");
//Este echo, muestra la fecha y hora ordenada por año,mes,día,hora,minuto,segundo
echo date("Y,m,d,H,i,s", time());
?>

```

#### 14.4.6 Aplicación web PHP para comprobar si el fichero de datos existe

Esta pequeña aplicación, tan solo responde mediante la función echo, lo que imprime en formato web, si el fichero existe en el servidor con un “SI” en caso afirmativo, o un “NO” en caso contrario. Esto ayuda a que, en caso de no existir, el módulo pueda mandar la línea de nombres, ya que por definición en ficheros con formato .CSV, según su construcción, la primera línea debe ser el nombre de los campos, para ir guardando bajo ellos el dato correspondiente a esa columna.

```
<?php  
if (file_exists('private/datos.csv')) {  
    echo "SI";  
} else {  
    echo "NO";  
}  
?>
```