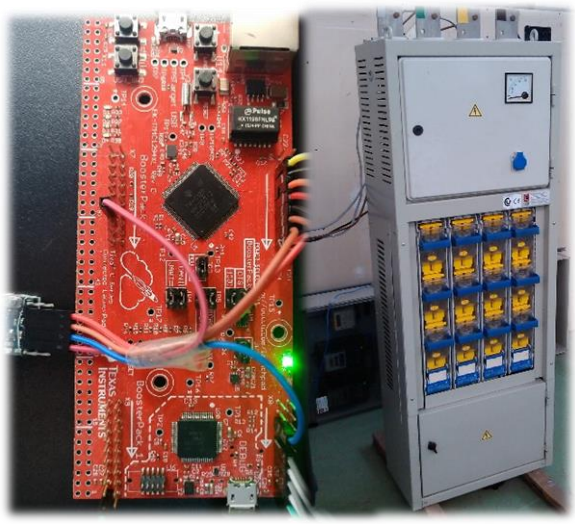
Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica



Concentrador de Medidas de Intensidad y Tensión con MCU

Autor: Carlos Pimentel Martorell

Tutores: Pedro L. Cruz Romero

Juan Carlos del Pino López

Dpto. Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Concentrador de Medidas de Intensidad y Tensión con MCU

Autor:

Carlos Pimentel Martorell

Tutores:

Pedro L. Cruz Romero

Profesor titular

Juan Carlos del Pino López

Profesor contratado doctor

Dpto. de Ingeniería Eléctrica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Carrera: Concentrador de Medidas de Intensidad y Tensión con MCU

Autor: Carlos Pimentel Martorell
Tutores: Pedro L. Cruz Romero
Juan Carlos del Pino López

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia y seres queridos

Agradecimientos

Ante todo agradecer a mi familia por todo el apoyo incondicional prestado, a mis amigos y a mi novia por su comprensión y sus ánimos.

Por todo el tiempo sacrificado para dar el mejor resultado e intentar disfrutar de ello.

Por los entrañables amigos que he conocido en mi estancia durante mis estudios.

A mis profesores y tutores por la atención recibida, y su inestimable ayuda, sus sugerencias y consejos. Y también mencionar su paciencia.

Muchas gracias a todos por vuestra ayuda sea en el grado que sea.

Carlos Pimentel Martorell

Sevilla, 2018

Resumen

En el presente proyecto vamos a abordar como combinar una caja distribuidora de baja tensión (CBT) con un micro-controlador con el fin de tomar medidas sobre lo que atañe a magnitudes eléctricas como vienen siendo las intensidades, los voltajes, las potencias, etc. y luego almacenarlas a disposición de quién sea.

Haremos un estudio de la implementación de componentes, los componentes a usar, la programación y los algoritmos empleados, y las distintas formas de comunicación y almacenaje de datos.

El microcontrolador usado es una placa de Texas Instruments (EK-TM4C1294XL) y la herramienta de programación es Code Composer Studio, descrito en lenguaje C.

Índice

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

La red eléctrica es una red interconectada que tiene como propósito suministrar electricidad desde los proveedores hasta los consumidores. Se compone de 3 ejes principales, las plantas generadoras que producen electricidad ya sea de combustibles fósiles o no fósiles (renovables); las líneas de transmisión que llevan la electricidad de las plantas generadoras a los centros de demanda; y los transformadores reducen el voltaje para que las líneas de distribución puedan entregarle energía al consumidor final. Destacar que es el mayor sistema industrial creado por el hombre.

Tradicionalmente, el sector de generación de electricidad se caracteriza por una tecnología madura y un marco legal estable que garantiza la rentabilidad del negocio: las compañías son administradas de acuerdo con la excelencia de sus criterios técnicos. Sin embargo, debido al impacto económico y social del sector, las compañías están bajo una fuerte presión por la necesidad de mejorar su eficiencia para lograr, no solo una mayor competencia en costos, sino también un nivel destacado de desempeño en relación con el medio ambiente. Por ello un buen control de la energía es imprescindible para el desarrollo de la industria y el buen uso del consumidor.

1.1. Red de distribución en media tensión

Esta componente de la red eléctrica representa el penúltimo eslabón al consumidor particular estándar, su voltaje se suele situar entre los 10.000 y 30.000 Voltios, y forma parte de la red de distribución (que no de transporte) la cual puede ser propiedad de una compañía privada.

Es habitual encontrar industrias y distintas instalaciones como aeropuertos u hospitales alimentados directamente por la red de distribución de media tensión para tener un buen acceso de potencia eléctrica. Luego el voltaje de las líneas es reducido a baja tensión y distribuido por la infraestructura con un centro de transformación de su propiedad y un cuadro distribuidor de baja tensión.

Ilustración 1.1. Cuadro distribuidor de baja tensión (CBT)

Es en este punto donde vamos a trabajar.

1.2. Objetivos. ¿Qué se pretende?

Dicho lo anterior nos vamos a enfocar en monitorizar los datos sobre las intensidades y tensiones que provee el cuadro distribuidor de baja tensión para permitir un mayor conocimiento de la carga en las líneas, como interés técnico, y dada la necesidad cada vez más imperiosa debido a la integración de renovables en baja tensión.

Esto se pretende conseguir mediante el uso de sensores magnéticos (acondicionando de señal para obtener la intensidad) y divisores de tensión, un microcontrolador TI (Texas Instruments) que administre las señales y las conexiones, las almacene en un disco duro para salvaguardar los datos, y la comunicación con la red para transferir los datos vía Ethernet y visualizarlos de forma cómoda. Todo ello implementado en la misma caja distribuidora.

Se pretende que el resultado de la obtención de los datos sea lo más fiable y barata posible.

1.3. Importancia del proyecto

Como se ha comentado anteriormente existe el interés (y casi la necesidad) de monitorizar constantemente el uso de la energía eléctrica de la que es proveída una instalación con su consecuente impacto comercial.

Entonces la idea consiste en desarrollar y estandarizar cajas distribuidoras de baja tensión con este sistema implementado para tener mayor conocimiento de la red en uso.

2 ESTUDIO Y ELECCIÓN DE COMPONENTES

2.1. Sensores TMR

Los sensores *magnetorresistivos de efecto túnel* (Tunneling Magnetoresistance o TMR) en su función más elemental consisten en separar dos materiales ferromagnéticos (normalmente hierro) por medio de un delgadísimo aislante (p.e. germanio) con aprox. 1 nm de grosor, esto hace que la corriente de túnel que pasa entre los materiales ferromagnéticos cambie con la orientación relativa de las 2 capas magnéticas provocando un suceso resistivo. Este efecto fue descubierto por Michel Julliere en 1975.

En nuestro caso particular haremos uso de los sensores ***LH45F*** *TMR Linear Sensor* los cuales usan un puente de Wheatstone compuesto por 4 de estos elementos sensibles a los campos magnéticos que dan una salida linealmente proporcional al campo magnético aplicado paralelamente a la superficie del conjunto de sensores, además de proveer de una alta compensación de temperatura a la salida.

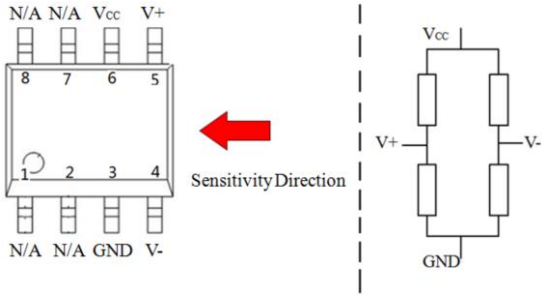


Ilustración 2. 1. Esquema del sensor del chip

Éste se puede alimentar de 1 a 7 V de tensión, con un rango de temperatura de funcionamiento de -40 a 125 ºC, cuestión importante pues las temperaturas que podría alcanzar en su implementación pueden superar fácilmente los 50 ºC.

También se puede destacar su muy bajo consumo.

Curva de transferencia para 1V de alimentación medido en Oersted*:

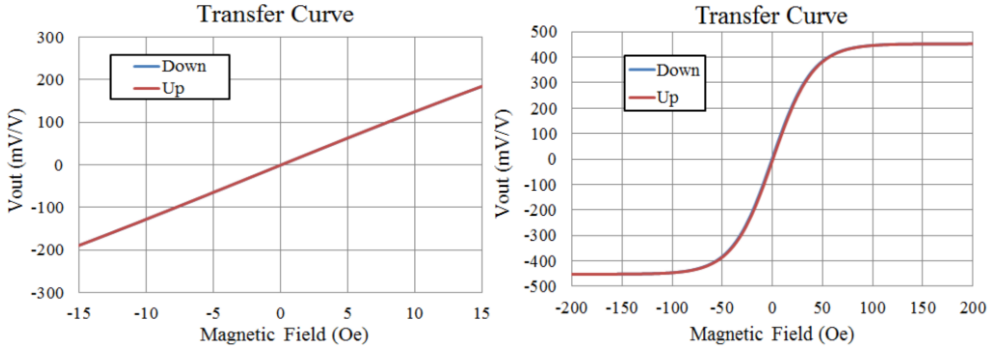


Ilustración 2. 2. Curva sensor con +- 15 Oe (izquierda) y +-200 Oe (derecha)

Importante resaltar la salida diferencial, muy a tener en cuenta en nuestro cálculo de las intensidades pues nos permite tomar cambios de voltaje de positivo a negativo y anular posibles interferencias electromagnéticas de la señal que afecten al modo común.

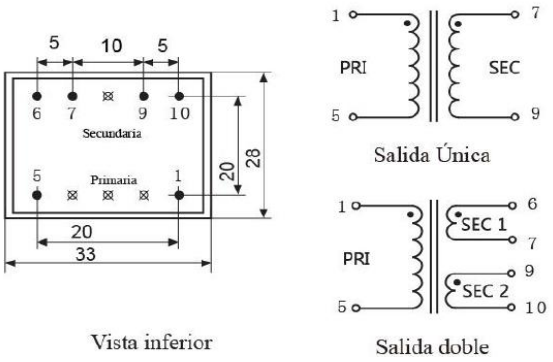
* 1 Oe = 0,1 mT

2.2. Circuito transformador y divisor de tensión

Dicho circuito no tiene mucha complicación, el objetivo es que mediante un pequeño transformador y una relación de resistencias se obtenga para un voltaje de entrada un voltaje inferior de salida para ser analizado por el MCU en su rango de bits abarcable. Se debe tener en cuenta que un aislamiento galvánico es más que recomendable por ello además de las ventajas que conlleva un transformador "aislamos" la red del microcontrolador de la red de baja tensión.

El transformador a usar será un PE3010 – M de la marca YHDC el cual presenta los siguientes datos técnicos:

Tensión de trabajo primaria	110V/220V/230V	Regulación de Voltaje	≤30%
Rango de tensión primaria	±10%	Frecuencia de trabajo	50/60 Hz
Potencia	1VA	Aumento de temperatura	≤15℃
Pérdida de carga	≤0.4W	Rigidez dieléctrica	3.75kV



Está constituido por acero de silicio de alta calidad. Rígido, sellado e impermeable, anti humedad. Se instala en PCB con un aislamiento térmico de clase B. Es de bajo coste y solo pesa 67 g. Su relación de transformación es de 220/12 V. Por ende su intensidad será de 83 mA.

Se puede apreciar el uso de dos devanados secundarios con los que **se parte a la mitad la potencia**, lo cual es interesante para minimizar la corriente de fuga hacia los pines del MCU.

Ilustración 2. 3. Esquema y especificaciones transformador

También damos especial importancia a la pérdida de potencia que se pueda efectuar por dicho método, usando un minitransformador de baja potencia (1VA) mitigamos considerablemente dicha pérdida y podemos optar a impedancias más pequeñas.

Hay que tener en cuenta que aunque el voltaje eficaz es de 230 V para nuestra instalación es necesario muestrear la onda con un pico de $230 \cdot \sqrt{2} = 325{,}27$ V (onda sinusoidal) sin que ésta haga saturar nuestro convertidor analógico-digital que, como se verá en la sección siguiente sobre las especificaciones del microcontrolador usado, es de 12 bits.

($2^{12\,\text{bits}} - 1 = 4095$ unidades para 3.3 V), puesto que:

```
mV per ADC code = (VREFP - VREFN) / 4096
```

Siendo VREFP=3.3 V y VREFN=0 V

Hay que destacar que hablamos de voltajes oscilando de positivo a negativo por tanto se nos hace obligatorio el uso del modo diferencial para adquirir los datos como es debido pues los valores negativos no quedarían registrados. Esto con lleva a que la precisión de nuestro ADC (o CAD) se reduzca a la mitad pasando de 4096 unidades para 3.3 V a **2048 para 3.3V.** Pues el convertidor toma como tensión 0 -> 2048, tensión 3.3 V -> 4096, y tensión -3.3 V -> 0.

Por tanto quedaría:

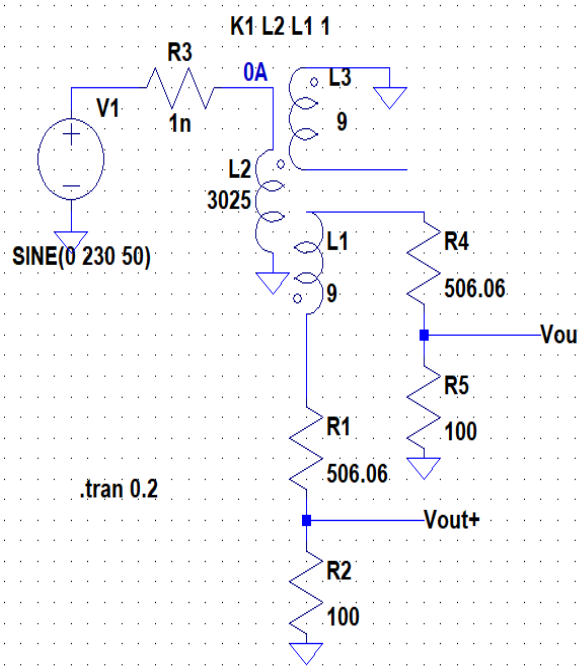


Ilustración 2. 4. Circuito electrónico de tensión

Para analizarlo basta con tomar una de las ramas y descartar la otra.

Siendo: $R1 = 506\ \Omega, R2 = 100\ \Omega$ y $Rt = 220/12\ V$

Considerando V1=325,27V nos queda que:

$$Vs \cdot \frac{1}{Rt} \cdot \left(\frac{R2}{R2 + R1}\right) = Vo$$

Con Vo pico máximo de 2,928 V < 3,3 V
Y potencia media disipada es de:

$$I = \frac{Vef \cdot 1/Rt}{R1 + R2} = 20{,}7\ mA < 41{,}5\ mA$$

$$P = Vef \cdot 1/Rt \cdot I =$$

$$= 230\ V \cdot \frac{12}{220} \cdot 0{,}0207\ A = 0{,}26\ W$$

Es decir, por cada rama circulará una intensidad de $\frac{20{,}7}{2} = 10{,}35\ mA$ y la mitad del voltaje medido, uno en signo positivo y otro en negativo, dándonos la misma diferencia de voltaje.

2.3. MCU de Texas Instruments

Nos disponemos a usar el microcontrolador Tiva TM4C1294NCPDT mediante el uso de la placa EK-TM4C1294XL de Texas Instruments. El Motivo por el que se ha escogido este MCU se debe a su gran versatilidad y número de puertos (especialmente los que pueden ser dedicados a ADC), su extenso número de funciones, su bajo coste y su fiabilidad, además de los puertos USB y Ethernet que muy convenientemente lleva incorporado.

El IDE desarrollado para dichos micro controladores es el Code Composer Studio, aplicación a través de la cual se han realizado los algoritmos de control y procesamiento.

La placa de TI está compuesta por 80 puertos, todos ellos con más de un posible uso o configuración. Las funciones que destacamos y van a ser usadas son:

- Convertidores ADC de 12 bits: Vamos a dar uso a 10 pines ADC, 4 de ellos se dedicarán a analizar la intensidad en modo diferencial (pines PE3-PE2, y PE1-PE0), y 6 de ellos a muestrear los voltajes de la caja de baja tensión en modo diferencial en sus diferentes fases; VR, VS y VT (pines PB4-PB5, PK0-PK1, y PK2-PK3). Además, se usa un canal interno del ADC para tomar la temperatura del equipo.
- Uso de botones: Daremos uso a los 2 botones ajustables para el usuario que vienen incorporados en la placa (pines PJ0 y PJ1).
- Leds identificadores: Utilizamos 2 de los 4 leds que incorpora la placa para indicaciones de las gestiones que realiza el MCU (pines PF0 y PF1).
- Puerto UART: A través de uno de los módulos UART que incorpora el MCU trasmitiremos datos por los pines PA6 (RX) y PA7 (TX) para un dispositivo bluetooth. También está se ha

habilitado el posible uso del módulo UART conectado al puerto ICDI por medio de los pines PA0 y PA1.

- Puerto micro USB: Muy útil y necesario para almacenar de forma segura todos los datos que obtenga el MCU. Por medio del uso del uDMA podemos escribir y leer datos en un dispositivo pendrive conectado a dicho puerto.
- Puerto Ethernet: Forma elegante e interesante de presentar y almacenar la información. Hace uso de los módulos MAC y PHY para realizar una conexión efectiva.
- Temporizadores de gestión: Uso de 3 módulos temporizadores para establecer pautas en la toma y gestión de datos del MCU.
- Sensor de Temperatura: La placa cuenta con un sensor de temperatura incorporado, no es de gran precisión pero ayuda a tomar información del estado al que sometemos el equipo.
- uDMA: el módulo del que tratamos es un microcontrolador de Acceso Directo a Memoria, accede a la memoria del sistema para leer y escribir datos independientemente de la CPU. Esta característica resulta muy útil para leer y escribir datos por el puerto USB en nuestro caso.

Ilustración 2. 5. Placa EK-TM4C1294XL con micro-controlador

2.4. Almacenamiento

Conectamos a la placa un dispositivo de almacenamiento masivo como viene a ser un Pen drive, pues es ahí donde se guardará la información de los datos, no es necesario un pen drive de ninguna clase en especial pero debe soportar el formato FAT o FAT32. Es necesario un adaptador para tal uso, pues necesitamos adaptar un conector USB a un puerto micro USB.

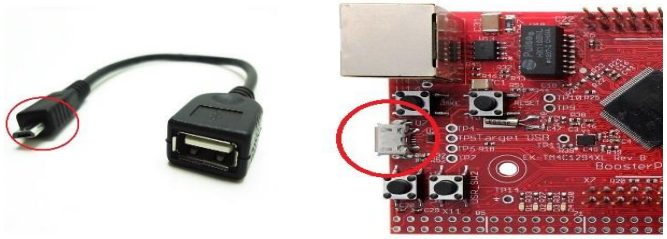


Ilustración 2. 6. Conexión USB

Se hace uso del uDMA y los pines de conexión del USB.

2.5. Comunicación

Es necesario que haya un sistema de comunicación para que se pueda saber el estado del equipo y los datos que está tratando, además de interactuar para enviarle datos como la fecha y la hora de inicio.

Se pretende comunicar la placa por medio de 2 métodos distintos:

- Por Bluetooth: Por medio de los módulos UART (concretamente el módulo 2) se establece una comunicación con el módulo/dispositivo bluetooth que hemos seleccionado. En nuestro caso se hace uso del módulo HC – 05 pues es simple y fácil de manejar.

Ilustración 2. 7. Módulo bluetooth HC-05

Está vía es la más importante pues es más difícil que presente errores y obliga al operario a estar cerca de la placa, lo que lo hace el método más fiable.

- Por Internet: Dando uso al puerto Ethernet incorporado en la placa podemos establecer una comunicación física con la red, éste se conecta y comunica con el rúter para obtener su propia dirección IP; de esta manera cualquier usuario conectado a la red podría acceder a la "página web del micro controlador", visualizar los datos y enviarle información cómo la fecha y hora de inicio, o marcarle el tiempo por medio de la propia red, es decir, que la propia red le proporcione la fecha y hora; de esta forma, cada cierto tiempo toma el tiempo de la red para ajustar cualquier desviación horaria que se haya producido.

Cabe destacar también el uso de los leds en la comunicación como refuerzo a dicha tarea. El led 1 es usado para plasmar cuando el MCU muestrea los datos que recibe del sistema eléctrico. El led 2 indica, post tratamiento de los datos, cuando empieza y termina la transferencia de información al dispositivo del almacenamiento masivo (pen drive), es decir, el almacenaje de información. Los leds 3 y 4 se usan para tener información del comportamiento del puerto Ethernet.

El uso de los botones está también pensado para navegar cómodamente por la información que recoge el MCU usándose conjuntamente con el puerto UART para "navegar" por el menú de datos. El botón 1 muestra por puerto UART la información pertinente al uso y almacenaje del USB; el botón 2 visualiza la información recogida por cada pin ADC y ya tratada, además de datos extra como la forma de muestreo, la temperatura del equipo, etc.

3 IMPLEMENTACIÓN

3.1. Dificultades del entorno

3.1.1. Interferencias Electromagnéticas

La IEM (o EMI de sus siglas en inglés) es la energía que causa una respuesta indeseable a cualquier equipo y puede causar alteraciones como sobretensión, caída de tensión, picos, etc., que pueden afectar a una red de comunicación. Es muy común en industrias y fábricas, donde la IEM es muy frecuente debido al mayor empleo de máquinas, motores (centros de control de motores), y las redes digitales y de computadoras cercanas a esas áreas.

El mayor problema causado por IEM son las situaciones esporádicas que degradan lentamente los equipos y sus componentes, además de ser muy común la ocurrencia de ruidos en la alimentación debido a mala puesta a tierra y blindaje.

Existen dos tipos de interferencia, la interna y la externa, siendo continuada o intermitente. Cada tipo tiene su propia causa y las causas más sujetadas a interferencia continuada son:

- Alimentación de 50 / 60 Hz.
- Motor eléctrico (especialmente el conmutador).
- Señales de radio de alta energía.
- Fuentes conmutadas.
- Microondas.

La fuente más común de ruido es constante y es causada por una fuente de alimentación 50 / 60 Hz, y en este proyecto será la IEM principal a tener en cuenta, además de ser la componente más común por ser una tensión oscilante, de alta potencia y un sistema de antena enorme.

Las Interferencias Electromagnéticas pueden reducirse por:

1. Cable trenzado.
2. Aislamiento óptico.
3. Uso de canaletas y cajas metálicas puestas a tierra.

Obviamente lo más sencillo para evitar posibles alteraciones de la señal de nuestros sensores sería trenzar el cable hasta su conexión con el microcontrolador.

En cuanto al blindaje en acoplamientos inductivos blindar campos magnéticos de baja frecuencia no es tan fácil como blindar campos eléctricos. La eficacia del blindaje magnético depende del tipo de material y su permeabilidad, su espesura y las frecuencias involucradas.

Entonces para escoger el material que más nos interesa en cuestión de blindaje de campos magnéticos tenemos que fijarnos en su permeabilidad magnética y su conductividad, que son 2 de las principales componentes para obtener la profundidad de penetración (δ) de un material.

De acuerdo con la fórmula de la profundidad de penetración:

$$\delta = \frac{1}{\sqrt{\pi \cdot f \cdot \mu_o \cdot \mu_r \cdot \sigma}}$$

O también: $\delta = \frac{0{,}066}{\sqrt{f \cdot \mu_r \cdot \sigma_r}}$

Nos percatamos que cuanto mayor sea el cociente de $\mu_r \cdot \sigma$ mayor será su efectividad para blindar, entonces siguiendo la siguiente tabla podemos concluir que el hierro es de los materiales más efectivos pero también el más barato de todos.

Metal	conductividad relativa σ_r	permeabilidad relativa μ_r @≤ 10 kHz	producto $\sqrt{\sigma_r \mu_r}$	cociente $\sqrt{\sigma_r/\mu_r}$
1. Plata	1,064	1	1,032	1,032
2. Cobre (Sólido)	1,00	1	1	1
3. Cobre (*)	0,10	1	0,316	0,316
4. Oro	0,70	1	0,837	0,837
5. Cromo	0,664	1	0,815	0,815
6. Aluminio (blando)	0,63	1	0,794	0,794
7. Aluminio (revenido)	0,40	1	0,632	0,632
8. Aluminio (papel metalizado 15 μm)	0,53	1	0,728	0,728
9. Aluminio (papel metalizado 25 μm)	0,61	1	0,781	0,781
10. Aluminio (*)	0,036	1	0,190	0,190
11. Latón (91 % Cu, 9 % Zn)	0,47	1	0,686	0,686
12. Latón (66 % Cu, 34 % Zn)	0,35	1	0,592	0,592
13. Magnesio	0,38	1	0,616	0,616
14. Zinc	0,305	1	0,552	0,552
15. Tungsteno	0,314	1	0,560	0,560
16. Berilio	0,33	1	0,574	0,574
17. Cadmio	0,232	1	0,482	0,482
18. Platino	0,17	1	0,412	0,412
19. Estaño	0,151	1	0,389	0,389
20. Tantalio	0,12	1	0,346	0,346
21. Plomo	0,079	1	0,281	0,281
22. Monel (67 % Ni, 30 % Cu, 2 % Fe, 1 % Mn)	0,041	1	0,202	0,202
23. Manganeso	0,039	1	0,197	0,197
24. Titanio	0,036	1	0,190	0,190
25. Mercurio (Líquido)	0,018	1	0,134	0,134
26. Nichrome (65 Ni, 12 Cr, 23 Fe)	0,0012	1	0,035	0,035
27. Bronce (Cu + Sn)	0,18	1	0,424	0,424
28. Acero (SAE 1045)	0,10	1.000	10,00	0,0100
29. Acero inoxidable (430)	0,02	500	3,162	0,0063
30. Isoperm	0,015	90	1,162	0,0129
31. Detamax	0,013	500	2,549	0,0051
32. Alfer	0,006	700	2,049	0,0029
33. Alperm	0,004	3.000	3,464	0,0012
34. Supermalloy	0,023	100.000	47,96	0,0005
35. 78 Permalloy	0,108	8.000	29,39	0,0037
36. Hierro puro (Hierro dulce)	0,17	5.000	29,15	0,0058
37. Conetic AA	0,031	20.000	24,90	0,0012
38. 4-79 Permalloy	0,0314	20.000	25,06	0,0013
39. Mumetal	0,0289	20.000	24,04	0,0012

Ilustración 3. 1. Tabla de permeabilidades y conductancias

Como se puede apreciar el hierro puro es uno de los materiales con mayor producto de $\mu_r \cdot \sigma$ además de ser barato.

Otra opción interesante podría ser el Supermalloy pero debido a su coste nos decantaremos por el hierro.

Entonces sabiendo que:

$$f = 50\ Hz;\ \ \mu_o = 4\pi \cdot 10^{-7};$$

$$\mu_{r\ Hierro} = 5000;$$

$$\sigma_{Hierro} = 9{,}93 \cdot 10^6\ S/m$$

$$\sigma_{r\ Hierro} = 0{,}1666$$

Tenemos que:

$$\delta_{Hierro\ 50Hz} = 0{,}319\ mm$$

Por último aclarar que el apantallamiento consiste en la conjunción de 3 distintas pérdidas:

$$S = A_{Pérdidas\ por\ Absorción} + R_{Pérdidas\ por\ Reflexión} + B_{Pérdidas\ por\ Multireflexión}$$

Cabe destacar que **las pérdidas por absorción** constituyen el principal mecanismo de apantallado en el caso de campos magnéticos de baja frecuencia.

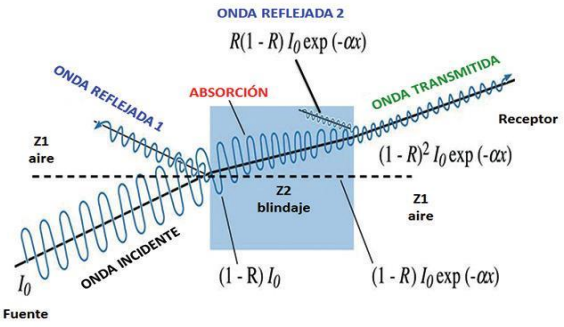


Ilustración 3. 2. Los blindajes y el CEM

Más adelante, en el punto 3.2., se ahondara en la cuestión del apantallamiento y se realizarán los cálculos pertinentes.

3.1.2. Temperatura y humedades

Las condiciones ambientales se deben de tener muy en cuenta pues de ello depende el buen funcionamiento del equipo, se debe comprobar su estado y los entornos a los que se va a someter. En esta sección se destacan 2 factores ambientales, factores principales a los que va a estar sometida nuestra placa pues se entiende que estará protegida de las inclemencias del exterior tales como la lluvia, el viento o la luz directa del Sol.

Temperatura: Un aumento de temperatura fuera de su condición "normal" de trabajo puede provocar un descenso del rendimiento del sistema, errores, pérdida de datos y deterioro del hardware del equipo. Según la documentación, en lo que respecta a características de operación nuestro micro controlador soporta un rango industrial de funcionamiento (-40 a 85 ^{o}C). Aun teniendo esto en cuenta no es recomendable que se a cerque a tan altas o bajas temperaturas si queremos disponer de una larga vida útil sin acontecer a errores. Por suerte el microcontrolador lleva incorporado un sensor de temperatura interno siempre activo usado para notificar al sistema de si se puede realizar operaciones sin peligro. Se le puede sacar provecho para obtener muestras de la temperatura del equipo y mostrársela al operario, tal y como se ha hecho.

En el cuadro distribuidor de baja tensión no sería normal que se alcanzaran temperaturas tan elevadas como los 85 ^{o}C, por la constante transmisión de potencia en los conductores se desprenderá más calor que en el entorno inmediato pero nada que deba ser tratado con especial importancia mientras haya cierto margen entre los conductores y el aparato. El que presentaría más peligro sería el sensor TMR pues éste estaría pegado al conductor para obtener fielmente la intensidad del campo magnético, pero tampoco presenta problema, con un rango de funcionamiento de -40 a 125 ^{o}C se puede estar más que tranquilo.

Humedad: Es más que sabido que la humedad y los componente electrónicos no son buenos compañeros, pueden provocar cortocircuitos y llevar al traste al equipo. Estos motivos son más que suficientes para poner en un entono libre de humedades al equipo con la CBT conjuntamente. Los sótanos y espacios similares pueden almacenar mucha humedad que pueda ser dañina por ello sería interesante incluir en nuestro equipo un sensor de humedad (cosa que no ha llegado a realizar) para tener constancia de si puede peligrar el microcontrolador.

Con mantener este tipo de entornos libres de humedad con deshumificadores sería suficiente para mantener a salvo el equipo.

3.2. Instalación y disposición de los componentes

Se procede a comentar parte por parte la instalación necesaria y los cálculos indicados para llevarla a cabo. Desarrollaremos en 3 apartados esta sección: Sensores TMR, circuitos de tensión y microcontrolador. Indicándose cómo se pretende realizar.

Sensores TMR:

Como se ha explicado anteriormente, estos sensores recogen el campo magnético que provoca la corriente al pasar por un conductor, dicho sensor debe ir pegado a la placa conductora. Hay un hueco especialmente útil que puede ser aprovechado por estos sensores como se mostrará a continuación:

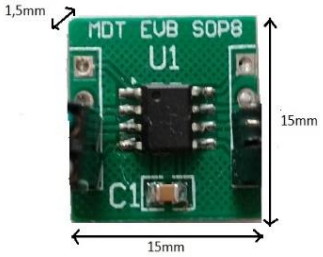


Ilustración 3. 3. Dimensiones de la placa del sensor

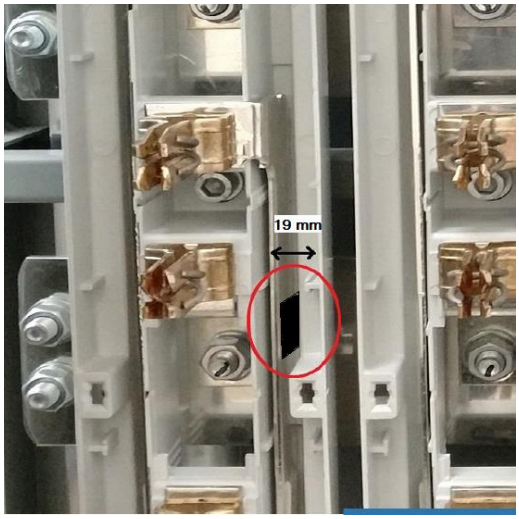


Ilustración 3. 4. Posición de la placa del sensor

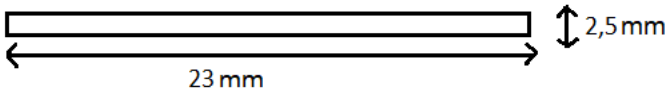
Entre el conductor y el sensor se propone dejar un espacio de 5 mm para evitar contacto directo entre metales y aplicar alguna lámina de hierro para atenuar si fuera necesario, entonces entre el conductor y la supuesta lamina dejaríamos 1 mm para incluir un aislante y lo mismo se haría desde la lámina atenuante hasta el sensor, aplicar un aislante, las juntas se haría mediante una pasta adherente que mantenga firmemente al sensor, aunque si queremos pensar en la posibilidad de reemplazarlo por cualquier circunstancia, es decir, que haya versatilidad, una masilla adherente como el Blu-Tack podría ser buena idea ya que hablamos de componentes ligeros.

Comentando el mismo punto, los cables conectados a la placa del sensor pueden ser conducidos por el hueco del conductor, es decir, por la canaleta de plástico, hasta la parte superior del módulo de tensión (o la inferior) sin que obstruyan el resto de elementos. Más adelante se pasará a especificar mejor este punto referente al cableado y se mostrará una figura para mejor comprensión.

Para saber la intensidad de dicho campo y si puede sufrir posibles interferencias se procederá a hacer los respectivos cálculos.

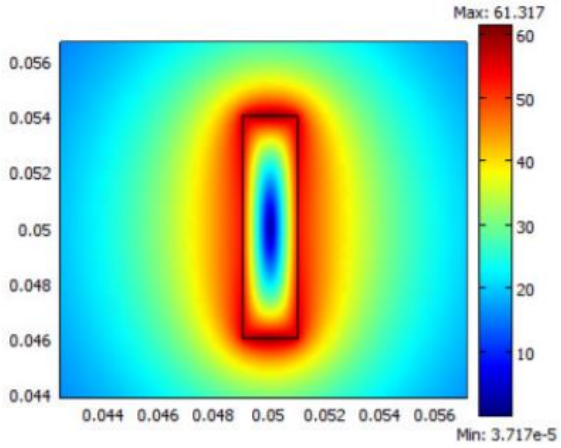
Debemos determinar la distribución de corrientes, la dirección y sentido del campo magnético, eligiendo una cuerva cerrada apropiada atravesada por las corrientes para calcular la intensidad de dicho campo.

Para ello debemos analizar la sección transversal del conductor:



Se aprecia que no es una sección transversal circular típica, por ello se deben hacer ciertas consideraciones: Supondremos una distribución de cargas uniforme en toda la sección. Dada la corta distancia a la que se sitúa el sensor sería un error considerar el campo cerrado con forma circular. Se supondrá también una longitud infinita de la barra siendo longitud >> anchura. Para que no haya confusiones con la anchura nos referimos a los 23 mm de la barra.

Para determinar la curva del campo magnético que se produce en dicha simetría atenderemos a la figura inferior donde se nos presenta una simulación del campo.



Campo magnético simulado por COMSOL Multiphysics en un conductor de sección rectangular con el plano X en dirección vertical.

Ilustración 3. 5. Intensidad del campo magnético en un conductor rectangular en 2D

Si nos fijamos en la simulación, la curva uniforme a la que más se asemeja el campo es a una elipse. Si nos alejáramos lo suficiente de ella decrecería su excentricidad hasta considerarse un campo circular como se puede ver a continuación:

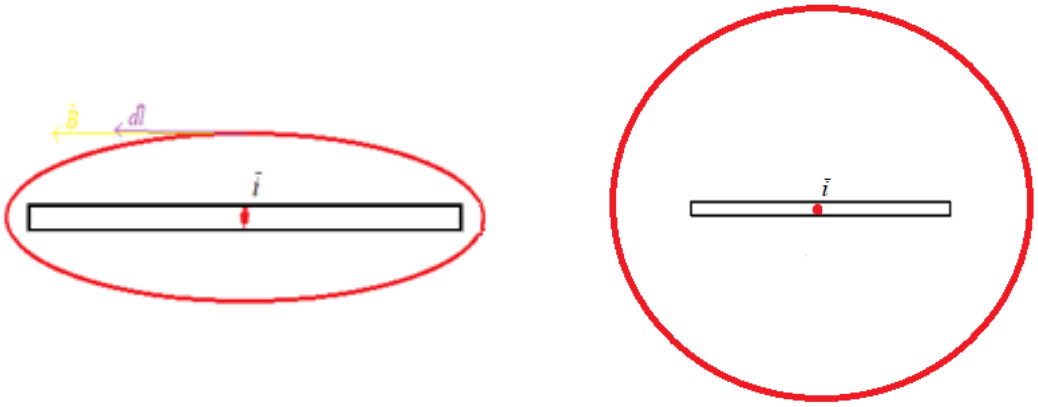


Ilustración 3. 6. Campos magnéticos de la sección

Pero obtener un cálculo más exacto a la geometría real del campo nos apoyaremos en las fórmulas propuestas en el **anexo 1.** La posición de la placa del sensor será paralela al eje X del conductor, y estará situada en el centro geométrico de la sección en lo que respecta al eje Z.

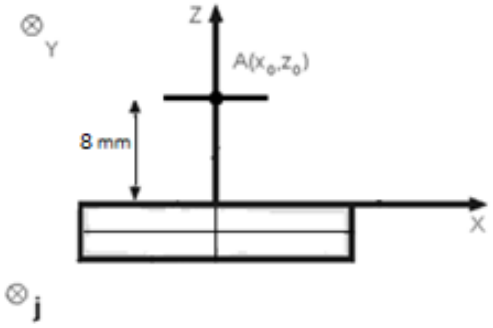


Ilustración 3. 7. Posición cartesiana del sensor respecto del conductor

Si prestamos atención al espacio dejado por el aislante que separa el sensor y el conductor se puede deducir la distancia total a la que se encuentra el núcleo del sensor TMR del centro de simetría de la sección del conductor. Entonces finalmente queda:

(la distancia del núcleo del sensor a la base de la placa es de 3mm)

$$distancia \rightarrow sensor + espacio + eje\ de\ la\ sección = 9{,}25\ mm = z$$

Entonces, según fórmulas del **anexo** 1, los parámetros nos quedan tal que así:

$$w = 23\ mm;\ \ p_o = 11{,}5\ mm;\ \ h = 2{,}5\ mm;\ \ z_o = 8\ mm;\ c = 3 \cdot 10^8 \frac{m}{s};\ \ \gamma = \frac{2}{c} \cdot \frac{i}{w \cdot h};$$

$$H_x = \bar{H}_x(p_o,\ z_o) - \bar{H}_x(-p_o,\ z_o);$$

$$H_z = 0;$$

$$|H_{total}| = \sqrt{H_x^2 + H_z^2} = |H_x|;$$

Note se que al estar el sensor posicionado paralelo al eje X la componente Z del campo magnético es nula, por lo que:

$$|H_x| = B = 1{,}4123 \cdot 10^{-5} \cdot i\ (T)$$

$$i \approx 70808{,}07 \cdot B\ (A)$$

Como se mostró en el capítulo anterior, sabiendo la diferencia de voltaje del sensor podemos determinar su campo magnético.

Es muy importante observar que el campo magnético del conductor hará saturar nuestro sensor:

Si 1000KVA es la potencia que llega a nuestra CBT entonces tomando el modelo monofásico la potencia de cada fase será de $\frac{1000}{3} KVA$, si el voltaje eficaz de cada línea es de 230V obtenemos que:

$$\frac{1000}{3} \cdot 230^{-1} = 1449{,}275\ A$$

Si son 4 módulos de corriente por cada fase de tensión queda $\frac{1449{,}275}{4} = 362{,}32\ A$ por cada fase en cada módulo de media asumiendo una demanda equitativa de potencia en todas las bornas.

Entonces:

$$362{,}32 \cdot \sqrt{2} = I_{pico} = 512{,}398\ A\ \rightarrow\ \ B = \frac{I_{pico}}{52203{,}617} = 7{,}236\ mT$$

Nuestro sensor satura a los 5 mT y pierde su linealidad a los 2 mT. Para solucionar esto hay que crear un apantallamiento entre el conductor y el sensor que debilite el campo magnético para ser medible en el rango de +-2 mT.

Antes de continuar se debe destacar que se ha probado de alejar el sensor del conductor pero debido al espacio limitado (19 mm) no se consigue un campo magnético lo suficiente mente pequeño como para prescindir de apantallamiento, entonces se continuará los cálculos realizados hasta ahora.

Primero obtengamos la eficiencia o efectividad que debe tener el blindaje, definida como:

$$S = 20 \cdot \log\left(\frac{B_{incidente}}{B_{saliente}}\right) = 11{,}17\ dB$$

Como se vio al principio de este capítulo:

$$S = A_{Pérdidas\ por\ Absorción} + R_{Pérdidas\ por\ Reflexión} + B_{Pérdidas\ por\ Multireflexión}$$

Siendo:

$$A = 1314{,}3 \cdot g \cdot \sqrt{f_{MHz} \cdot \mu_r \cdot \sigma_r} = 1314{,}3 \cdot \frac{g}{\delta} \cdot 0{,}0066 = 8{,}6744 \cdot \frac{g}{\delta} \quad (dB);$$

$$Siendo\ \boldsymbol{g}\ el\ espesor\ y\ estando\ \boldsymbol{g}\ y\ \boldsymbol{\delta}\ en\ cm$$

$$R = 74{,}6 - 10\log\left(\frac{\mu_r}{f_{MHz} \cdot \sigma_r \cdot \left(d + \frac{y}{2}\right)^2}\right)\ (dB); \qquad \text{(En el caso de campos magnéticos)}$$

$$Siendo\ d = 0{,}1\ cm\ y\ y = 0{,}25\ cm$$

$$B = 20\log\left(1 - e^{-2g/\delta}\right)\ (dB);$$

Como también se comentó en el capítulo 3 las pérdidas por absorción constituyen el principal mecanismo de apantallado en el caso de campos magnéticos de baja frecuencia. Entonces, atendiendo a la citada oración y para facilitar cálculos depreciaremos a ***B*** que suele aportar pocos decibelios a la ecuación. También comentamos que aplicaríamos **hierro** para el apantallamiento, aunque en este caso sea para atenuación.

Nos queda que:

$$A = 271{,}925 \cdot g\ \ dB \quad y \quad R = -26{,}14\ \ dB$$

Y por tanto:

$$S - R = A \rightarrow 39{,}958 = 271{,}925 \cdot g$$

Entonces queda: $g = 1{,}37\ mm$ como espesor del material.

Por último quedaría comprobar si las pérdidas por reflexiones múltiples se podrían considerar despreciables.

Entonces: $B = -1{,}6 \cdot 10^{-3}\ \ dB$

Por tanto B se considera despreciable, la conclusión es que debemos entre poner una lámina de hierro puro de 1,38 mm de grosor a 1 mm de la superficie del conductor, ocupando el espacio sobrante con material aislante, como podría ser un plástico. Importante aplicarle al hierro una capa de algún producto impermeabilizante, sea esmalte industrial, esmalte alquídico o semejante, para conseguir una gran durabilidad del hierro sin que éste presente oxidación.

Para eliminar correctamente las corrientes inducidas en la lámina de hierro debe conectarse ésta al plano de masas.

Respecto al blindaje, deberíamos seguir el mismo procedimiento, y atendiendo al peor caso, en el que hay al menos 35 mm que separan al canal del conductor del conductor de enfrente, se expresará de la siguiente manera:

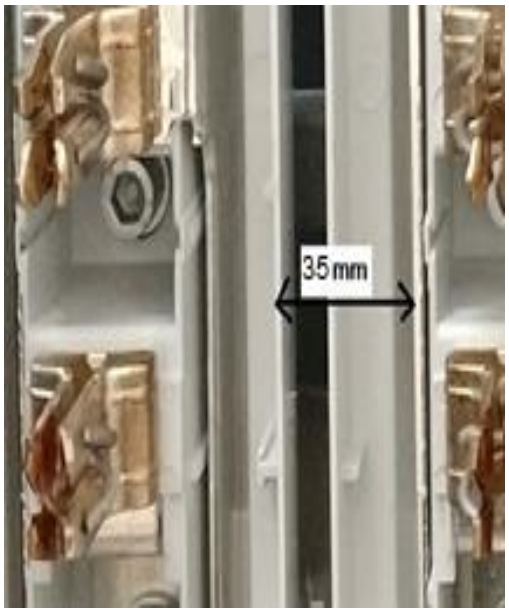


Ilustración 3. 8. Distancia de blindaje entre sensor y conductor

Para una distancia de 35 mm se tiene:

$$Curva\ campo\ magn. \approx P_{círculo} = 2\pi R;$$

Atendiendo a la ley de Ampère:

$$\oint \bar{B} \cdot \bar{dl} = \oint B \cdot dl \cdot \cos 0^{o} =$$

$$B \oint dl = B \cdot P_{círculo} = \mu_{o} \cdot i$$

$$B = \frac{I_{pico}}{174880} = 2{,}93\ mT$$

$$S = 20 \log\left(\frac{2{,}93}{0{,}01}\right) = 49{,}34\ dB$$

$$(con\ error \pm 0{,}5\%)$$

$$A = 271{,}925 \cdot g\ \ dB;\ \ R = -1{,}909\ dB;$$

$$g = 1{,}88\ mm$$

Por tanto con un encapsulado de dicho grosor será suficiente para blindar cualquier sensor y/o equipo electrónico que esté comprendido dentro de la CBT.

Y para terminar con los sensores TMR debemos prestar atención a las interferencias electromagnéticas que pueden atacar a la información (pequeña señal) que viaja por los cables de los sensores hasta los pines del micro-controlador. El mayor problema lo compondrán los acoplamientos inductivos que se pueden modelar como una fuente de tensión de ruido. Los acoplamientos capacitivos no se considerarían debido al carácter diferencial de la señal transmitida.

Entonces el peor caso sería cuando los cables de nuestro sensor salen del blindaje y se mantienen paralelos al conductor más cercano, en dicho caso el campo eléctrico recibido sería el del conductor donde obtiene las mediciones el sensor, antes presentaremos en la siguiente figura como se dispondrían los cables.

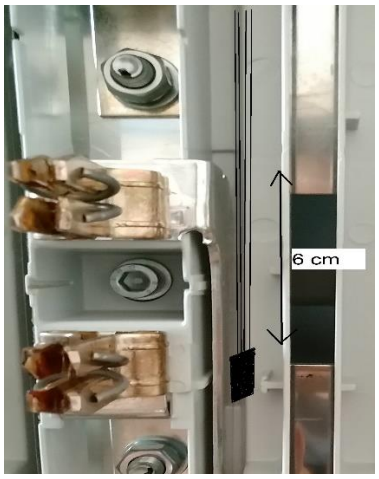


Ilustración 3. 9. Distribución de los cables del sensor

$$7{,}236\ mT\ /\ \mu_{o}\ = 5758{,}23 \frac{A}{m} = H$$

Entonces podemos relacionar los campos eléctricos y magnéticos a través de la impedancia intrínseca del vacío, la cuál es una constante física que de las radiaciones electromagnéticas.

$$Z_o = \frac{E}{H} \rightarrow \ E = Z_o \cdot H = 2170800\, V/m;$$

$$Siendo: \ \ Z_o = 120 \cdot \pi; \qquad E_{rms} = \frac{E}{\sqrt{2}} = 1534987{,}4\, V/m;$$

$$Si \ V_{inducida} = 2 \cdot E_{rms} \cdot l \cdot \sin\left(\pi \cdot S \cdot \frac{f}{c}\right);$$

$$(Tensión\ inducida\ en\ una\ espira\ con \cos\phi = 0^o)$$

$$c = 3 \cdot 10^8 m/s; \ \ f = 50\, Hz; \ \ S = 1\, mm\ (ditancia\ entre\ cables\ diferenciales); \ \ l = 6\, cm$$

$$V_{ind.} = 0{,}1\, mV_{rms}$$

$$Sensibilidad\ del\ sensor: 12\, mV/V/Oe \qquad Si\ lo\ alimentamos\ a\ 5V$$

$$Sens = 60\, mV/Oe \ \ \rightarrow \ \ \frac{0{,}1}{60} = 0{,}0017\, Oe = 1{,}7 \cdot 10^{-4}\, mT$$

Si los valores de nuestro sensor se mueven entre +- 2 mT obtenemos que:

$$\frac{1{,}7 \cdot 10^{-4}\, mT}{2\, mT} \cdot 100 = error_{indc.} = 0{,}009\%$$

Con esto podemos concluir que no se hace necesario pero sí **recomendable** trenzar los cables de nuestros sensores hasta los pines del MCU manteniendo los cables lo más alejado posible de los conductores.

Circuito de tensión:

Engloba la implementación de los mini transformadores y los divisores resistivos. Estos no suponen un aparatoso tamaño, siendo las resistencias del orden de 1cm y los mini transformadores de poco más de 3cm. Teniendo en cuenta que como se mostró en la figura 2.4. haremos uso de 4 resistencias y un minitransformador por toma de voltaje, necesitaremos 3 conjuntos iguales pues son 3 tomas de tensión proveniente del voltaje trifásico.

Teniendo esto en cuenta lo más óptimo sería unir todo en un solo PCB junto al lado de la placa del MCU en el lateral de la caja distribuidora de baja tensión:

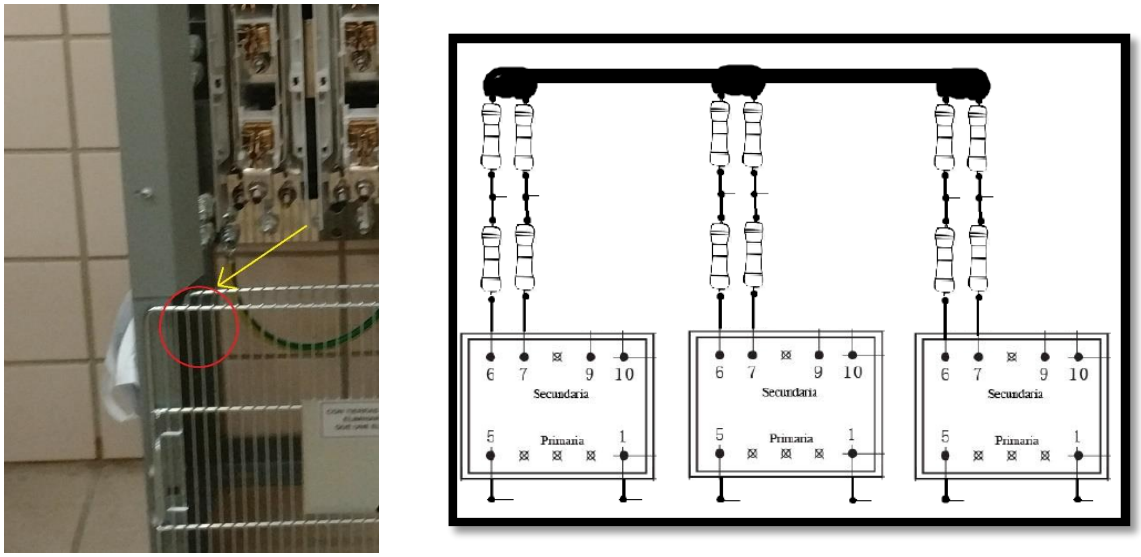


Ilustración 3. 10. Posición y estructura del circuito de tensión

Finalmente unos cables se conectarían a las bornas de salida como pequeño desvío a los pines del devanado primario de cada mini transformador en la PCB. Como la tensión de las 3 componentes es la misma en los 4 módulos bastará conectarse a las bornas del primero, la tierra del PCB se conectaría con el cuerpo de la Caja BT.

Es importante también pensar en blindar dichos componentes para que no sufran alteración. Con un encapsulamiento de hierro puro de 1,8 mm de grosor nos aseguramos que no ocurrirá interferencia inductiva alguna pues la distancia que separaría al circuito del conductor más próximo sería mucho mayor que en el peor caso antes mencionado.

Microcontrolador:

Se aplica el mismo caso para nuestra placa de Texas Instrumets. La idea consiste en pegar las dos placas (micro y circuito de tensión) juntas, una al lado de la otra. Por tanto la disposición de la placa será la misma.

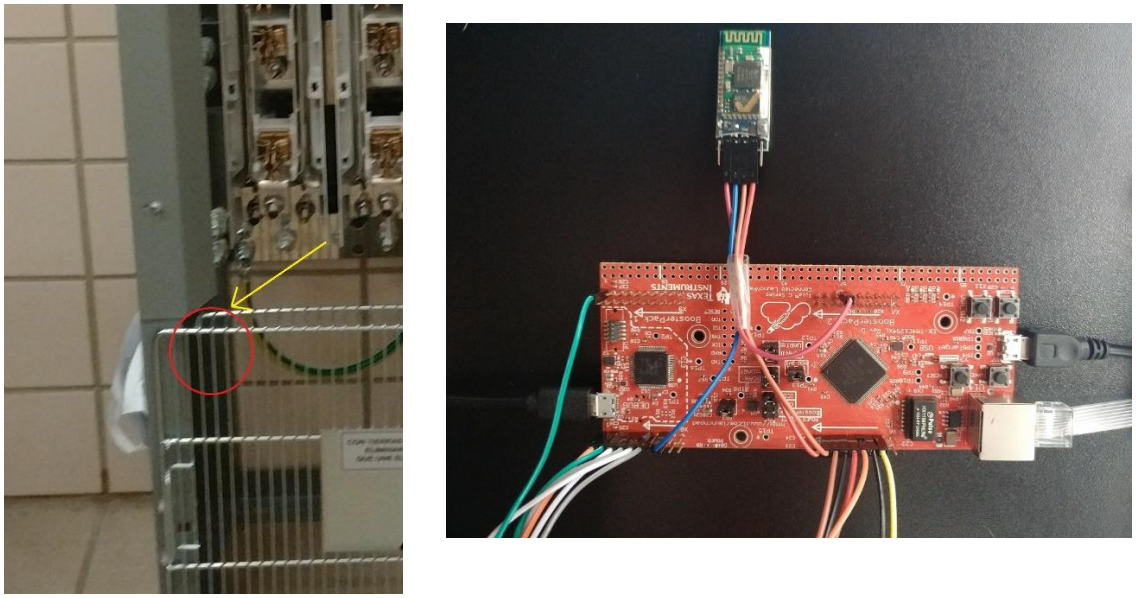


Ilustración 3. 11. Posición del microcontrolador

También debemos pensar en la alimentación y en el aislamiento.

En lo que atañe a lo primero debemos considerar que si conectamos la placa a un enchufe de la red esta estará a merced de los datos que analiza, es decir, si registra un pico de tensión la placa lo puede sufrir, si hay un apagón lo mismo. O podemos alimentarla por una fuente independiente como una batería pero a menos que la propia batería externa expida información de su nivel de carga no se podría saber cuándo reemplazar/recargar la batería, lo cual es un grave problema. Por tanto podríamos concluir que si no tenemos una buena batería externa y queremos dar prioridad a la comodidad de no estar pendiente lo mejor sería conectarla a la red pero teniendo presente esos problemas.

En cuanto al aislamiento podríamos decir lo mismo que en casos anteriores. La profundidad de penetración en del hierro es muy baja entonces tomando los resultados que hemos obtenido en el peor caso para el sensor, siendo éste mucho más sensible a los campos magnéticos que nuestro micro-controlador bastará con un encapsulamiento de 1,8 mm de hierro puro para poner nuestro equipo a salvo.

4 PROGRAMACIÓN Y TRATAMIENTO DE SEÑALES

4.1. Introducción

Antes de comenzar a hablar sobre el proceso de tratamiento es conveniente que nos familiaricemos con el entorno de desarrollo y comentar el diseño del código pues el proyecto se basa principalmente en la buena compilación del código del micro controlador para realizar todas las operaciones que deseamos.

Como ya se ha comentado, usamos Code Composer Studio como entorno de desarrollo, el lenguaje de programación es C pero con pequeñas modificaciones para adaptarse al entorno de un micro controlador como la activación de periféricos, definición de macros, configuración del reloj, etc.

El código está dividido en 2 principales secciones (en realidad son 3 pero luego se explicará por qué se soslaya una de ellas). Una para la recogida de muestras y posterior análisis, la otra se encarga de salvaguardar los datos obtenidos en un disco duro/memoria flash, ambas secciones presentan su propia información por pantalla según la elección del usuario sobre que quiere visualizar.

Antes de continuar mostraremos en la siguiente figura como queda el diagrama de funciones en el que se ha trabajado. Éstas son las llamadas que realizan entre sí las funciones y se intenta que con ello se vislumbre mejor la organización del código. Las flechas azules gruesas indican las llamdas directas a otras funciones mientras que las flechas negras indican llamadas indirectas por medio de la apertura de una variable o de la llamada a otra función no representada.

También destacar la enumeración. Los números indican el orden de llamada a las funciones, dichos números pueden ser rojos o negros; los números negros indican que solo se realiza una llamada desde el inicio del código mientras que los rojos indican a las funciones que después de llamarse se seguirán recurriendo, es decir, se realizan llamadas sucesivas cada cierto tiempo o suceso.

Para terminar con la explicación nos fijamos en 2 cosas destacables. Las funciones del margen derecho no parecen provenir de ninguna llamada en específica, eso se debe a que son interrupciones, su llamada no depende directamente de otras funciones ya que proceden del mismo proceso. Además se observa que se han etiquetado distintos colores para cada función dependiendo de la sección en la que estén descritas. Todas las funciones mencionadas se han desarrollado en especial para esta aplicación, es decir, no proceden de librerías, exceptuando 2 casos especiales, la función *USBHCDMain()* que resulta una función importante y por ello se ha mencionado pero está descrita en las librerías; e *itoa()*, la cual aún al ser una función desarrollada a mano se considera parte integra de una librería también realizada, por ello no se muestra en el diagrama.

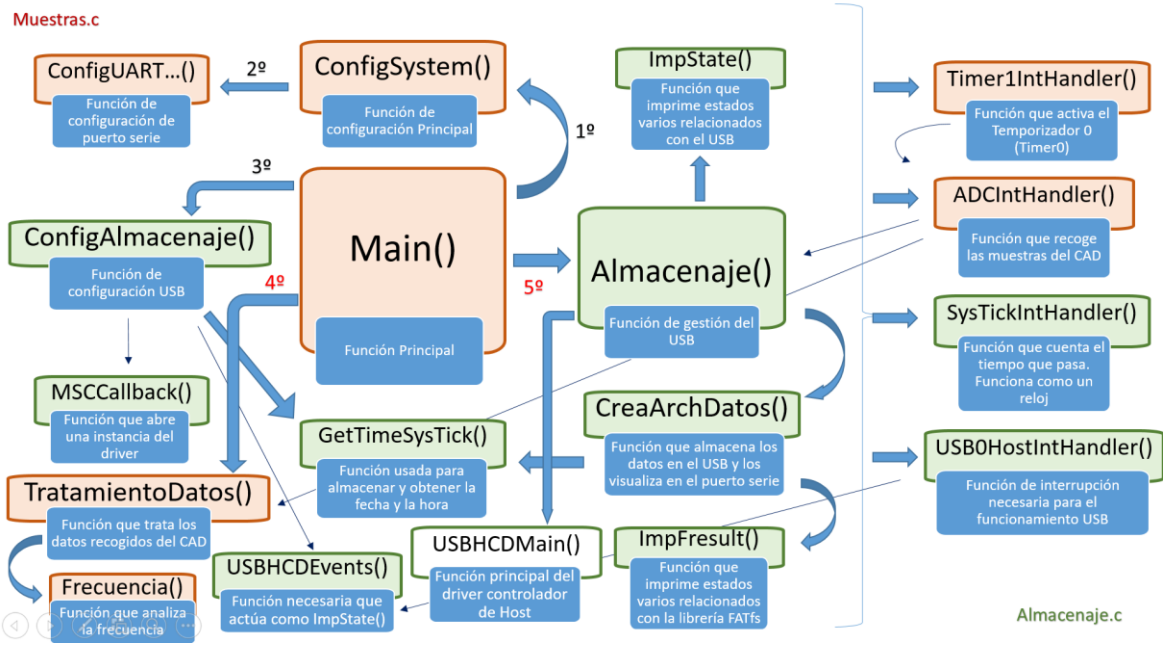


Ilustración 4. 1. Diagrama de funciones del código

Dicho lo anterior empezaremos, entonces, por la primera sección, llamada *Muestras.c*, el código ya incluye comentarios detallados sobre los sucesos o acciones que se pretenden en el código. No se pasará a comentar macros u otras cuestiones que particulares de los microcontroladores que no tengan que ver directamente con el proceso que se pide.

4.2. Algoritmos de tratamiento de la señal

Aquí presentamos la primera sección, *Muestras.c*. Es donde reside la función principal, *main()*, a partir de la cual emanan todas las funciones y llamadas.

Empecemos con las principales declaraciones:

```
43
44#define PERIODO_MUESTRAS    5   //  Periodo de aquisición de muestras (seg)
45#define Nmuestras   200 //  Muestras que se van a tomar
46#define FS      ( Nmuestras * 50 / 2 )  //  Frecuencia de muestreo = 5kHz (100 muestras por periodo de 50Hz)
47#define VREF        3.3
48#define Rt          1   //( 220 / 12    )   //  Relación de transformación
49#define R1          506.06      //Ohmnios
50#define R2          100
51#define DivTens     (R2 / (R1+R2))  //  Relación del voltaje de salida
52#define SenSensor   (5 * 0.012/0.0001)  //  Sensibilidad del sensor (V/T) <- 12mV/V /Oe  Alimentado a 5V
53#define pi          M_PI
54#define FactorMag   256199  //  Suponemos que se ha realizado el cálculo para atenuar
55                          //  el campo magnético entonces: Ipico/2mT = 256199
```

Ilustración 4. 2. Declaraciones principales

Como podemos apreciar en la figura aquí se especifican importantes cosas sobre cómo se tratarán los datos. Podemos resaltar que el buffer de datos es de 200 muestras, que cada 5 segundos se toman nuevas muestras, la frecuencia de muestreo, etc.

Continuamos con la configuración de los periféricos del micro-controlador, tales como los temporizadores, CAD, habilitación de interrupciones, botones para navegar por el menú, leds, etc. Como vemos a continuación:

```
136
137 void ConfigSystem(void)
138 {
139
140     MAP_SysCtlMOSCConfigSet(SYSCTL_MOSC_HIGHFREQ);
141
142     //  Frecuencia de procesamiento de 120MHz
143     RELOJ = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL
144         | SYSCTL_CFG_VCO_480), 120000000);
145
146     //  Habilitamos los puertos A, B, E, J, K y N
147     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
148     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
149     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
150     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
151     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
152     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
153
154     //  Habilitamos timers, ADCs y UARTs deseados
155     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
156     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
157     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
158     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
159     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
160
161     //  Led 0 y 1
162     MAP_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1);
163     MAP_GPIOPadConfigSet(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1,
164                     GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);    LED1_OFF; LED_OFF;
165
166     //  Botón 1 y 2
167     MAP_GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1);
168     MAP_GPIOPadConfigSet(GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1,
169             GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

170
171     MAP_ADCReferenceSet(ADC0_BASE, ADC_REF_INT);        //  Con una referencia de 3.3V para 4095 unidades de código ADC
172
173     //  Usamos modo diferencial para la intensidad: AIN1--AIN0 y AIN3--AIN2
174     MAP_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_0 | GPIO_PIN_1);
175
176      // Usamos modo diferencial para el voltaje: AIN10--AIN11, AIN16--AIN17 y AIN18--AIN19
177     MAP_GPIOPinTypeADC(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
178     MAP_GPIOPinTypeADC(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
179
180     // Indicamos que vamos a usar el ADC 0, secuenciador de muestra 0, activación de secuencia por timer y Prioridad máxima (0)
181     MAP_ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_TIMER, 0);
182
183     // Indicamos qué canales analógicos y de qué forma se van a leer y almacenar
184     MAP_ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH0 | ADC_CTL_D);     //  IR1 (canal diferencial 0)
185     MAP_ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH1 | ADC_CTL_D);     //  IS1 (canal diferencial 1)
186     MAP_ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH5 | ADC_CTL_D);     //  VR  (canal diferencial 5)
187     MAP_ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH8 | ADC_CTL_D);     //  VS  (canal diferencial 8)
188     MAP_ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH9 | ADC_CTL_D);     //  VT  (canal diferencial 9)
189     MAP_ADCSequenceStepConfigure(ADC0_BASE, 0, 5, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);   // Sensor Temperatura
190     ADCIntRegister(ADC0_BASE, 0, ADC0IntHandler);       //  Registramos nuestra función de interrupción del ADC
191     MAP_ADCIntEnable(ADC0_BASE, 0);                         //  Habilitación de la interrupción del ADC0 de secuencia 0
192     MAP_IntEnable(INT_ADC0SS0);                             //  Habilitación de la interrupción para el ADC0 de secuencia 0
193     MAP_ADCSequenceEnable(ADC0_BASE, 0);                    //  Activamos el uso del ADC 0 de secuenciador de muestra 0
194     MAP_ADCIntClear(ADC0_BASE, 0);
```

```
195
196     //  Los Timers0 usarán el oscilador interno de 16MHz
197     MAP_TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_PIOSC);
198     //  Elegimos el tipo de Timers0 deseados, en este caso de anchura completa (32 bits) y periódico.
199     MAP_TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
200     MAP_TimerLoadSet(TIMER0_BASE, TIMER_A, (uint32_t)(16e6 / FS));
201     //  Indicamos que la activación del ADC por timer se habilita para el timer0_A.
202     MAP_TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
203
204     //  Los Timers1 usarán el oscilador interno de 16MHz
205     MAP_TimerClockSourceSet(TIMER1_BASE, TIMER_CLOCK_PIOSC);
206     //  Elegimos el tipo de Timers1 deseados, en este caso de anchura completa (32 bits) y periódico.
207     MAP_TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
208     MAP_TimerLoadSet(TIMER1_BASE, TIMER_A, (uint32_t)(16e6 * (uint32_t)PERIODO_MUESTRAS));
209     TimerIntRegister(TIMER1_BASE, TIMER_A, TIMER1IntHandler);
210     MAP_TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
211     MAP_IntEnable(INT_TIMER1A);
212     MAP_TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
213
214     //  UART por puerto USB (ICDI)          (No funcionan los 2 a la vez si se usa la librería uartstdio.h)
215 //  ConfigUARTUSB();
216
217     //  UART por Bluetooth (PA6 y PA7)
218     ConfigUARTBluetooth();
219
220     MAP_IntMasterEnable();                                  //Habilitación global de interrupciones
221
222 }
```

Ilustración 4. 3. Función ConfigSystem()

A partir de aquí se mostrará parte del cuerpo del código main (código principal) y la sección de tratamiento para luego pasar a la segunda sección principal.

```
226
227 void main(void) {
228
229     ConfigSystem();
230
231     int res = 0, i;
232     int menu=1;
233     bool activado=true;
234
235     UARTprintf("\033[2J\033[H");     //Limpia el terminal
236
237     UARTprintf("Pulse el botón 1 para continuar");
238     while (!B1_ON);
239     UARTprintf("\033[2J\033[H");
240
241 //    UARTprintf("Configurando Ethernet\n");
242 //    res = ConfigComunicador();
243 //
244 //    UARTprintf("\n Acabada configuración\n");
245 //    while(!B2_ON);
246
247     UARTprintf("Configurando USB\n");
248     ConfigAlmacenaje();
249     UARTprintf("Configuración establecida como Host\n");
250
251     UARTprintf("\n Aprete al botón 1 para Información USB");
252     UARTprintf("\n O aprete al botón 2 para despliegue de Información ADC\n");
253     while (!B1_ON && !B2_ON);
254
255     //  Habilitamos el Timer encargado de la frec. de muestreo
256     TimerEnable(TIMER0_BASE, TIMER_A);
257     //  Habilitamos el timer que dicta cada cuando se inicia el muestreo
258     TimerEnable(TIMER1_BASE, TIMER_A);
```

Ilustración 4. 4. Principio de la función Main()

Arriba se muestra los pasos que sigue *main* para la configuración de distintos procesos, el Ethernet, el USB, la habilitación y configuración de periféricos (que se ha mostrado anteriormente); además de informar por puerto UART de las gestiones que está realizando el micro.

Lo que le sigue a continuación son meras presentaciones para mostrar de forma cómoda la información por puerto UART, se destaca la sección de transferencia de datos. Entonces pasaremos

a ver la sección de tratamiento de datos, a la cual se recurre cada vez que se termina de muestrear las ondas eléctricas, es decir, el periodo de muestreo:

```
680
681 void TratamientoDatos (int n)
682 {
683
684     int i, pmax[Vdatos][2], pmin[Vdatos][2];
685     float max[Vdatos], min[Vdatos], offset[Vdatos], frec[Vdatos], amp[Vdatos], rms[Vdatos];
686     float Sn[2], P[2], Q[2], teta[2];
687
688     //Inicializamos las variables de interés
689     for(i=0; i<Vdatos; i++)
690     {max[i]=-1000; min[i]=1000; rms[i]=0; amp[i]=0;
691     frec[i]=0; offset[i]=0; pmax[i][0]=0; pmin[i][0]=0;}
692
693     for(i=0; i<2; i++) {Sn[i]=0; P[i]=0; Q[i]=0; teta[i]=0;}
694
695     //  Max. error cuantización = 1/2048
696     //  Max. Error de amplitud = (2*pi/(Nmuestras/2) ) / 2*pi
697     //  Max. error total = 1 - (1 - max. error cuantización)*(1 - max. error de amplitud)
698     //  Para Nmuestras = 200 -> 0'3% de error en la medida
699
700     //2048 representa el voltaje 0 del modo diferencial
701
702     for(i=0; i<n; i++)
703     {
704         BR1 = ((Muestras.IR1[i]-2048)*VREF/2048) / SenSensor;   //  Campo Magnético
705         Muestras.IR1[i] = BR1 * FactorMag;  //  Intensidad
706
707         //  Buscamos los 2 puntos de máximos y mínimos
708         if (max[0]<Muestras.IR1[i]) {pmax[0][1]=pmax[0][0]; pmax[0][0]=i; max[0]=Muestras.IR1[i];}
709         if (min[0]>Muestras.IR1[i]) {pmin[0][1]=pmin[0][0]; pmin[0][0]=i; min[0]=Muestras.IR1[i];}
710
711         BS1 = ((Muestras.IS1[i]-2048)*VREF/2048) / SenSensor;   //  Campo Magnético
712         Muestras.IS1[i] = BS1 * FactorMag;  //  Intensidad
713
714         if (max[1]<Muestras.IS1[i]) {pmax[1][1]=pmax[1][0]; pmax[1][0]=i; max[1]=Muestras.IS1[i];}
715         if (min[1]>Muestras.IS1[i]) {pmin[1][1]=pmin[1][0]; pmin[1][0]=i; min[1]=Muestras.IS1[i];}
716
717         //  Voltaje
718         Muestras.VR[i] = (Rt / DivTens) * ((Muestras.VR[i]-2048)*VREF)/2048;
719
720         if (max[2]<Muestras.VR[i]) {pmax[2][1]=pmax[2][0]; pmax[2][0]=i; max[2]=Muestras.VR[i];}
721         if (min[2]>Muestras.VR[i]) {pmin[2][1]=pmin[2][0]; pmin[2][0]=i; min[2]=Muestras.VR[i];}
722
723         //  Voltaje
724         Muestras.VS[i] = (Rt / DivTens) * ((Muestras.VS[i]-2048)*VREF)/2048;
725
726         if (max[3]<Muestras.VS[i]) {pmax[3][1]=pmax[3][0]; pmax[3][0]=i; max[3]=Muestras.VS[i];}
727         if (min[3]>Muestras.VS[i]) {pmin[3][1]=pmin[3][0]; pmin[3][0]=i; min[3]=Muestras.VS[i];}
728
729         //  Voltaje
730         Muestras.VT[i] = (Rt / DivTens) * ((Muestras.VT[i]-2048)*VREF)/2048;
731
732         if (max[4]<Muestras.VT[i]) {pmax[4][1]=pmax[4][0]; pmax[4][0]=i; max[4]=Muestras.VT[i];}
733         if (min[4]>Muestras.VT[i]) {pmin[4][1]=pmin[4][0]; pmin[4][0]=i; min[4]=Muestras.VT[i];}
734     }
```

```
    //  Máximo y mínimo ponderados de la Intensidad R1
    max[0] = (Muestras.IR1[ pmax[0][0] ] + Muestras.IR1[ pmax[0][1] ]) / 2;
    min[0] = (Muestras.IR1[ pmin[0][0] ] + Muestras.IR1[ pmin[0][1] ]) / 2;
    offset[0] = (max[0] + min[0]) / 2;
    frec[0] = Frecuencia( Muestras.IR1 );
    amp[0] = (max[0] - min[0]) / 2;
    for(i=0; i<n; i++)      rms[0] = rms[0] + Muestras.IR1[i] * Muestras.IR1[i];
    rms[0] = rms[0] / n; rms[0] = sqrtf( rms[0] );

    //  Máximo y mínimo ponderados de la Intensidad S1
    max[1] = (Muestras.IS1[ pmax[1][0] ] + Muestras.IS1[ pmax[1][1] ]) / 2;
    min[1] = (Muestras.IS1[ pmin[1][0] ] + Muestras.IS1[ pmin[1][1] ]) / 2;
    offset[1] = (max[1] + min[1]) / 2;
    frec[1] = Frecuencia( Muestras.IS1 );
    amp[1] = (max[1] - min[1]) / 2;
    for(i=0; i<n; i++)      rms[1] = rms[1] + Muestras.IS1[i] * Muestras.IS1[i];
    rms[1] = rms[1] / n; rms[1] = sqrtf( rms[1] );

    //  Máximo y mínimo ponderados del Voltaje R
    max[2] = (Muestras.VR[ pmax[2][0] ] + Muestras.VR[ pmax[2][1] ]) / 2;
    min[2] = (Muestras.VR[ pmin[2][0] ] + Muestras.VR[ pmin[2][1] ]) / 2;
    offset[2] = (max[2] + min[2]) / 2;
    frec[2] = Frecuencia( Muestras.VR );
    amp[2] = (max[2] - min[2]) / 2;
    for(i=0; i<n; i++)      rms[2] = rms[2] + Muestras.VR[i] * Muestras.VR[i];
    rms[2] = rms[2] / n; rms[2] = sqrtf( rms[2] );

    //  Máximo y mínimo ponderados del Voltaje S
    max[3] = (Muestras.VS[ pmax[3][0] ] + Muestras.VS[ pmax[3][1] ]) / 2;
    min[3] = (Muestras.VS[ pmin[3][0] ] + Muestras.VS[ pmin[3][1] ]) / 2;
    offset[3] = (max[3] + min[3]) / 2;
    frec[3] = Frecuencia( Muestras.VS );
    amp[3] = (max[3] - min[3]) / 2;
    for(i=0; i<n; i++)      rms[3] = rms[3] + Muestras.VS[i] * Muestras.VS[i];
    rms[3] = rms[3] / n; rms[3] = sqrtf( rms[3] );
```

```
    //  Máximo y mínimo ponderados del Voltaje T
    max[4] = (Muestras.VT[ pmax[4][0] ] + Muestras.VT[ pmax[4][1] ]) / 2;
    min[4] = (Muestras.VT[ pmin[4][0] ] + Muestras.VT[ pmin[4][1] ]) / 2;
    offset[4] = (max[4] + min[4]) / 2;
    frec[4] = Frecuencia( Muestras.VT );
    amp[4] = (max[4] - min[4]) / 2;
    for(i=0; i<n; i++)      rms[4] = rms[4] + Muestras.VT[i] * Muestras.VT[i];
    rms[4] = rms[4] / n; rms[4] = sqrtf( rms[4] );

    //  Cálculo potencias y desfase de VR e IR1
    for(i=0; i<n; i++)
    {
        Sn[0] = Sn[0] + (Muestras.VR[i]*Muestras.IR1[i] * Muestras.VR[i]*Muestras.IR1[i]);
        P[0] = P[0] + ( Muestras.VR[i] * Muestras.IR1[i] );
    }
    Sn[0] = Sn[0] / n; Sn[0] = sqrtf( Sn[0] );
    P[0] = P[0] / n;
    Q[0] = sqrtf( Sn[0]*Sn[0] - P[0]*P[0]);
    teta[0] = atanf ( Q[0] / P[0]);

    //  Cálculo potencias y desfase de VS e IS1
    for(i=0; i<n; i++)
    {
        Sn[1] = Sn[1] + (Muestras.VS[i]*Muestras.IS1[i] * Muestras.VS[i]*Muestras.IS1[i]);
        P[1] = P[1] + ( Muestras.VS[i] * Muestras.IS1[i] );
    }
    Sn[1] = Sn[1] / n; Sn[1] = sqrtf( Sn[1] );
    P[1] = P[1] / n;
    Q[1] = sqrtf( Sn[1]*Sn[1] - P[1]*P[1]);
    teta[1] = atanf ( Q[1] / P[1]);
```

```
805
806     //  Amplitud, RMS, Offset y Frecuencia
807     for (i=0; i < Vdatos; i++)
808     {
809         Mdatos[i][0]=amp[i];    Mdatos[i][1]=rms[i];
810         Mdatos[i][2]=offset[i]; Mdatos[i][3]=frec[i];
811     }
812
813     //  Potencia Aparente, Potencia Activa, Potencia Reactiva, Fase
814     //  para P1R y P1S
815     for (i=0; i < 2; i++)
816     {
817         Mdatos[i+Vdatos][0]=Sn[i];  Mdatos[i+Vdatos][1]=P[i];
818         Mdatos[i+Vdatos][2]=Q[i];   Mdatos[i+Vdatos][3]=teta[i];
819     }
820
821 }
822
```

Ilustración 4. 5. Función TratamientoDatos()

Por tanto, como se puede comprobar, la matriz *Mdatos* contiene la resolución de todos los datos tratados. Además, antes de acabar con dicha sección se mostrará a continuación la función que trata la frecuencia de la onda muestreada. No es un requisito pero nunca está de más saber el estado en que llega la onda:

```
825 float Frecuencia (float *V)
826 {
827
828     float dato, dato_ant;
829     int a=0, conteo=0, conteo1=0, saltos=0,
830             b=(sizeof(V) / sizeof(float));
831     bool T=true;
832
833     while (a < b)
834     {
835         if (!T) dato_ant=dato;
836         dato = V[a];
837         if (T) {dato_ant=dato; T=false;} // Solo una vez

838
839         //  Si se produce un cambio de signo respecto del dato anterior
840         // (paso por 0)  se puede representar la frecuencia de la señal
841         if ( ((dato <0 && dato_ant >=0 ) || (dato >=0 && dato_ant <0)) && saltos<=2)
842         {
843             if(saltos == 0) conteo1 = a;
844             else conteo = a - conteo1;
845             saltos++;
846         }
847         a++;
848     }
849
850     if (saltos == 0)    return (0);
851     else if (saltos == 1)   return ( FS / (conteo*2) );
852     else return ( FS / conteo );
853
854 }
```

Ilustración 4. 6. Función Frecuencia()

4.3. Preparación y envío de datos

Por último y antes de finalizar este capítulo mostraremos la sección de transferencia de datos el cual consiste en preparar los datos que se van a mandar introduciéndolos en una cadena de caracteres. La sección se inicia de la siguiente forma:

```
265       if (MuestraCmpltd)
266       {
267           //  Función donde tratamos los datos del muestreo
268           TratamientoDatos (Nmuestras);
269           LED_ON;
270           if (res==-3){ if (VisualizarUSB) UARTprintf("\nProblema con el dispositivo USB");}
271           else if (res==-2){ if (VisualizarUSB) UARTprintf("\nDispositivo USB lleno, se necesita reemplazo");}
272           else
273           {
274               //  Fraccionamos la lectura de la cadena Cdatos en 3 partes
275               //  Esto es debido a que la librería fatfs no soporta manejar buffers de datos tan grandes
276               //  Para solucionarlo fraccionamos los datos dentro de una misma carpeta de "hora"
277               for (i=0; i<3; i++)
278               {
279                   goto CadenaDeDatos;
280                   VuelveCadena:
281                   res = Almacenaje( Cdatos );
282                   MismaCarpeta = true;
283                   if (VisualizarUSB)
284                   {
285                       if(res != 0) {UARTprintf("\nDATO NO GUARDADO\n\n"); break;}
286                       //HABRÍA QUE HACER BUFFER AUXILIAR EN LA FLASH
287                       else UARTprintf("\nDATO ALMACENADO\n\n");
288                   }
289               }
290               MismaCarpeta = false;
291           }
292           LED_OFF;
293
294   //      res = Comunicador(Muestras, Mdatos);
295
296           MuestraCmpltd = false;
297       }
```

Ilustración 4. 7. Sección de Transferencia de datos. Función Main()

En dicha sección se aprecia tanto la función *TratamientoDatos* como *Almacenaje* y *Comunicador*. A esta sección solo se recurre a ella cuando se ha acabado de muestrear las ondas como se comentó en el punto anterior. La cadena de datos está asignada a *Cdatos* y luego es tomada como puntero en la función de *Almacenaje*. Uno se puede percatar de que la función *Almacenaje* se encuentra en un bucle *for*, esto se debe a una fragmentación de los datos que se explicará en el punto *5.1.2.*

La forma en la que se ingresan los datos en la cadena Cdatos es de la siguiente manera:

```
604         strcat(Cdatos, "\n  Potencia VS * IS1\n");
605         strcat(Cdatos, " Sn = ");
606         itoa((int)Mdatos[3][0], &TempNum);      strcat(Cdatos, &TempNum);
607         strcat(Cdatos, ".");
608         itoa((int)fabsf(Mdatos[3][0]*100) % 100, &TempNum); strcat(Cdatos, &TempNum);
609         strcat(Cdatos, " VA   P = ");
610         itoa((int)Mdatos[3][1], &TempNum);      strcat(Cdatos, &TempNum);
611         strcat(Cdatos, ".");
612         itoa((int)fabsf(Mdatos[3][1]*100) % 100, &TempNum); strcat(Cdatos, &TempNum);
613         strcat(Cdatos, " W   Q = ");
614         itoa((int)Mdatos[3][2], &TempNum);      strcat(Cdatos, &TempNum);
615         strcat(Cdatos, ".");
616         itoa((int)fabsf(Mdatos[3][2]*100) % 100, &TempNum); strcat(Cdatos, &TempNum);
617         strcat(Cdatos, " VAr   Teta = ");
618         itoa((int)Mdatos[3][3], &TempNum);      strcat(Cdatos, &TempNum);
619         strcat(Cdatos, ".");
620         itoa((int)fabsf(Mdatos[3][3]*100) % 100, &TempNum); strcat(Cdatos, &TempNum);
621         strcat(Cdatos, " º\n");
622     }
623
624     goto VuelveCadena;
```

Ilustración 4. 8. Extracto cadena de datos Cdatos

Este fragmento de muestra solo comprende una parte de la asignación para no sobrecargar el documento. Se aprecia el uso de una función denominada *itoa* (integer to ascii), ésta es imprescindible para tratar los datos numéricos pues no sería posible escribir/presentar datos de tipo entero o similar mediante las funciones de las librerías *uartstudio* y *fatfs* imprescindibles para visualizar y almacenar información. Por ello merece una mención especial y se mostrará a continuación el código realizado a la pertinente función:

```
18 void strreverse(char* begin, char* end)
19 {
20     char aux;
21     while(end>begin)
22         aux=*end, *end--=*begin, *begin++=aux;
23 }
24
25
26 void itoa(int value, char *str)
27 {
28     char* wstr=str;
29     int sign;
30 //    div_t res;
31
32     if ((sign=value) < 0) value = -value;
33
34     do {
35       *wstr++ = (value%10)+'0';
36     }while(value=value/10);
37
38     if(sign<0) *wstr++='-';
39     *wstr='\0';
40
41     strreverse(str,wstr-1);
42 }
```

Ilustración 4. 9. Función itoa()

La idea se apoya principalmente en 2 mecanismos. Sumar las cifras aisladas (haciendo el resto de 10) al carácter '0'. Es un concepto simple pero tiene el inconveniente de voltear la cadena de números en una cadena de caracteres, entonces se desvoltea con un juego de punteros en *strreverse*.

5 ALMACENAJE Y COMUNICACIÓN

5.1. Algoritmos de almacenaje

5.1.1. Configuración y gestión del USB

Aquí se pretende mostrar el código seguido para almacenar la cadena de datos cedida por la función principal a través del puntero *Cdatos* en un dispositivo de almacenamiento masivo como viene siendo un pen drive. Primero debemos configurar al microcontrolador para que pueda gestionar tales operaciones.

La función que se muestra a continuación es la encargada de configurar el uso de un dispositivo USB de almacenamiento:

```
208
209 int ConfigAlmacenaje (void)
210 {
211
212     extern uint32_t RELOJ;
213     uint32_t PLLRate;
214     char buffer[7], buffer2[7];
215
216     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
217     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
218     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);
219     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
220
221     //  No es necesario introducirlo
222     HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
223     HWREG(GPIO_PORTD_BASE + GPIO_O_CR) = 0xff;
224
225     MAP_GPIOPinConfigure(GPIO_PD6_USB0EPEN);
226     MAP_GPIOPinTypeUSBDigital(GPIO_PORTD_BASE, GPIO_PIN_6);
227     // Enable PB0 and PB1 as VBUS and ID pins.
228     MAP_GPIOPinTypeUSBAnalog(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
229     // Enable the D+/D- as USB pins.
230     MAP_GPIOPinTypeUSBAnalog(GPIO_PORTL_BASE, GPIO_PIN_6 | GPIO_PIN_7);
231
232     MAP_GPIOPinTypeGPIOInput(GPIO_PORTQ_BASE, GPIO_PIN_4);
233
234     //  Habilitamos el uDMA
235     MAP_uDMAEnable();
236     MAP_uDMAControlBaseSet(g_sDMAControlTable);
237

238     //  Almacenamos la fecha y la hora
239     UARTprintf("Introduce la fecha en ddmmaa: \t");
240     while(!MAP_UARTCharsAvail(UART2_BASE) && !MAP_UARTCharsAvail(UART0_BASE));
241     UARTgets(buffer, sizeof(buffer));
242     UARTprintf("Introduce la hora en hhmmss: \t");
243     while(!MAP_UARTCharsAvail(UART2_BASE) && !MAP_UARTCharsAvail(UART0_BASE));
244     UARTgets(buffer2, sizeof(buffer2));
245     UARTprintf("\n");
246     GetTimeSystick(buffer, buffer2);
```

```
248     //Habilitamos el Systick
249     MAP_SysTickPeriodSet( RELOJ / TICKS_PER_SECOND);
250     SysTickIntRegister(SysTickIntHandler);
251     MAP_SysTickEnable();
252     MAP_SysTickIntEnable();
253     MAP_IntEnable(INT_SYSCTL);
254
255     MAP_IntEnable(INT_USB0);
256     USBIntRegister(USB0_BASE, USB0HostIntHandler);
257
258     USBStackModeSet(0, eUSBModeHost, 0);
259
260     USBHCDRegisterDrivers(0, g_ppHostClassDrivers, g_ui32NumHostClassDrivers);
261
262     // Abre una instancia del driver de clase de almacenamiento masivo.
263     g_psMSCInstance = USBHMSCDriveOpen(0, MSCCallback);
264
265     //  Configura la señal habilitada de encendido en activa
266     //  y no habilita el fallo de energía.
267     USBHCDPowerConfigInit(0, USBHCD_VBUS_AUTO_HIGH | USBHCD_VBUS_FILTER);
268
269     // Se le comunica a la librería USB el reloj de la CPU y la frecuencia del PLL.
270     //  ¡NO ES NECESARIO, ESTÁ POR DEFECTO!
271     SysCtlVCOGet(SYSCTL_XTAL_25MHZ, &PLLRate);
272     USBHCDFeatureSet(0, USBLIB_FEATURE_CPUCLK, &RELOJ);
273     USBHCDFeatureSet(0, USBLIB_FEATURE_USBPLL, &PLLRate);
274
275     // Inicializa el controlador USB la para la operación de host (servidor).
276     USBHCDInit(0, g_pHCDPool, HCD_MEMORY_SIZE);
277
278     return(0);
279
280 }
```

Ilustración 5. 1. Función ConfigAlmacenaje()

Como se puede comprobar, a la hora de configurar el dispositivo se nos pedirá la fecha y hora actuales pues nos servirán para registrar y almacenar ordenadamente los datos pasados por la cadena *Cdatos*.

Después de dicha configuración se crea otra función especializada en tratar con el USB y escribir/leer los datos en él. Es la denominada función *Almacenaje* como se muestra a continuación:

```
int Almacenaje(char *datos)
{

    uint32_t ui32DriveTimeout = USBMSC_DRIVE_RETRY;;
    int res;

    while (1)
    {

        // Mira si el dispositivio MSC ha sido enumerado.
        if(g_eState == STATE_DEVICE_ENUM)
        {
            // Comprobamos fácilmente si al dispositivo le cuesta
            // arrancar después de conectarse.
            if(USBHMSCDriveReady(g_psMSCInstance) != 0)
            {
                // Espera 500ms antes de comprobar de nuevo si
                // el dispositivo está listo de nuevo.
                SysCtlDelay(SysCtlClockGet()*10/2);

                // Decrementa la cuenta de reintento.
                ui32DriveTimeout--;

              // Si se alcanza el tiempo límite cambiamos
              // al estado STATE_TIMEOUT_DEVICE.
              if(ui32DriveTimeout == 0) g_eState = STATE_TIMEOUT_DEVICE;

              continue;   //Salta hasta el final del bucle
          }

          // Resetea el directorio de trabajo a la raíz.
          g_cCwdBuf[0] = '/';
          g_cCwdBuf[1] = '\0';

          // Inicializa el sistema de ficheros.
          // Procura abrir el directorio. A algunos driver les cuesta mas arrancar
          // que a otros, y esto puede producir un fallo aunque el dispositivo USB
          // se haya enumerado, si se está todavía inicializando.
          f_mount(0, &g_sFatFs);
          f_opendir(&g_sDirObject, g_cCwdBuf);
          strcat(g_cCwdBuf, "Mstr_CBT");

          directorio:
          if(f_opendir(&g_sDirObject, g_cCwdBuf) == FR_OK)
          {
              // El driver está completamente listo por ello cambiamos a ese estado.
              g_eState = STATE_DEVICE_READY;
          }

          else if (f_mkdir (g_cCwdBuf) == FR_OK)  goto directorio;

          else
          {
              if (VisualizarUSB)  UARTprintf("Fallo en la creación u opertura de directorio\n");
              return(-3);
          }
      }

      ImpState();

      if(g_eState == STATE_DEVICE_READY)
      {
          res = CreaArchDatos(datos);

          if(g_eState == STATE_DEVICE_FULL)
          {
              f_mount(0, NULL);
              ImpState();
          }

          USBHCDMain();

          return(res);
      }

      if (g_eState == STATE_TIMEOUT_DEVICE || STATE_NO_DEVICE) return(-3);

      //  Esta es la rutina principal del driver controlador de servidor.
      //  Se le tiene que llamar periodicamente por la aplicación principal
      //  fuera del contexto de una llamada de retorno.
      //  Esto permite mediante un sistema cooperativo simple acceder a la interfaz
      //  del driver controlador de servidor sin la necesidad de un RTOS (Real Time Operative System)
      USBHCDMain();
  }

}
```

Ilustración 5. 2. Función Almacenaje()

Básicamente lo que viene a hacer es esperar a que el driver USB esté inicializado para a continuación inicializar el sistema de ficheros (librería FATfs), luego crea la carpeta *Mstr_CBT* (Muestras de la Caja de Baja Tensión) en la raíz del dispositivo, no se ha podido extender a un nombre más largo pues pasados los 8 caracteres se produce un error de creación debido a limitaciones de la librería FATfs.

Debemos dar especial importancia a la función *CreaArchDatos* pues es la encargada de escribir los datos en el dispositivo como se verá a continuación.

5.1.2. Gestión del guardado y acceso a datos

En esta sección ondearemos en la función *CreaArchDatos* y la forma en la que se gestiona la creación de los nombres de archivos y ficheros con sus respectivos nombres. Lo primero que hay que tener en cuenta es que cuando llamamos a dicha función le estamos pasando la cadena de datos que deriva de *Cdatos*, a continuación se asigna un nombre de carpeta correspondiente con la fecha actual, ésta esta referenciada por la fecha que hemos introducido al principio, al mismo tiempo se crea de nuevo una carpeta que tiene como nombre la hora en la que se almacenan los datos tomados.

Aquí el correspondiente código:

```
530
531 int CreaArchDatos(char *Datos)
532 {
533
534     FRESULT fresult;
535     uint32_t ui32BytesRead;
536     uint32_t ui32BytesWritten;
537     char g_cTmpBuf[DATA_BUF_SIZE], num;
538     char g_cTempBufPath[PATH_BUF_SIZE];
539     int n = 1;
540
541     g_cTmpBuf[0]= '\0';
542     g_cTempBufPath[0]= '\0';
543
544     // Intenta no hacer nada si no hay un driver conectado.
545     if(g_eState != STATE_DEVICE_READY)      return(FR_NOT_READY);
546
547     if (!MismaCarpeta)
548     {
549         // Copia la ruta actual al buffer temporal así puede ser manipulada.
550         strcpy(g_cPathFileName, g_cCwdBuf);
551
552         // Si no está en el nivel de raíz entonces acopla un separador.
553         if(strcmp("/", g_cCwdBuf))      strcat(g_cPathFileName, "/");
554
555         //  Creamos Archivo de la fecha de muestra
556         strcat(g_cPathFileName, GetTimeSystick(0, "0"));
557         ArchivoFecha:
558         if(f_opendir(&g_sDirObject, g_cPathFileName) == FR_OK);
559         else { f_mkdir (g_cPathFileName);   goto ArchivoFecha; }
560
561         //  Ponemos como nombre al archivo de guardado la hora a la que se recoge
562         sprintf(g_cTempBufPath, GetTimeSystick(0, 0));
563
564         if (VisualizarUSB)  UARTprintf("%s> %s\n", g_cPathFileName, g_cTempBufPath);
565         strcat(g_cPathFileName, "/");
566         strcat(g_cPathFileName, g_cTempBufPath);
567         sprintf(g_cPathSave, g_cPathFileName);
568
569         ArchivoHora:
570         if(f_opendir(&g_sDirObject, g_cPathFileName) == FR_OK);
571         else { f_mkdir (g_cPathFileName);   goto ArchivoHora; }
572     }
573
```

```
574     while(1)
575     {
576         strcpy(g_cPathFileName, g_cPathSave);
577         sprintf(g_cTempBufPath, "Datos_");
578         itoa(n, &num);
579         strcat(g_cTempBufPath, &num);
580         strcat(g_cTempBufPath, ".rtf");
581         if (VisualizarUSB)  UARTprintf("%s> %s\n", g_cPathFileName, g_cTempBufPath);
582         strcat(g_cPathFileName, "/");
583         strcat(g_cPathFileName, g_cTempBufPath);
584 
585         fresult = f_open(&g_sFileObject, g_cPathFileName, FA_CREATE_NEW | FA_WRITE | FA_READ);
586 
587         if(fresult == FR_EXIST)
588         {
589             if (VisualizarUSB)  UARTprintf("Fichero existente\n");
590             n++;
591             continue;
592         }
593 
594         else if(fresult != FR_OK)
595         {
596             ImpFresult(fresult);
597             return(-1);
598         }
599 
600         else break;
601     }
602 
603     //  Cambia el Timestamp
604     g_sFileInfo.fdate = (WORD)((anno + 2000 - 1980) * 512U | mes * 32U | dia);
605     g_sFileInfo.ftime = (WORD)(h * 2048U | min * 32U | seg / 2U);
606     fresult = f_utime( g_cPathFileName, &g_sFileInfo);

608     sprintf(g_cTmpBuf, Datos);
609 
610     do
611     {
612         fresult = f_write(&g_sFileObject, (const void *)g_cTmpBuf,
613                 sizeof(g_cTmpBuf) -1, (UINT *)&ui32BytesWritten);
614 
615         if ( fresult == FR_TIMEOUT)
616         {
617             ImpFresult(fresult);
618             if (VisualizarUSB)  UARTprintf(" Disco lleno");
619             g_eState = STATE_DEVICE_FULL;
620             return(-2);
621         }
622 
623         if(fresult != FR_OK)
624         {
625             ImpFresult(fresult);
626             return(-1);
627         }
628 
629 
630         if (VisualizarUSB)
631         {
632             // Le un bloque de datos del archivo. Lee tantos como pueda introducir
633             // en el buffer temporal, incluyendo un espacio para el terminador.
634             fresult = f_read(&g_sFileObject, g_cTmpBuf, ui32BytesWritten,
635                              (UINT *)&ui32BytesRead);
636 
637             // Si ocurre un error imprime nueva linea y devuelve un error al usuario
638             if(fresult != FR_OK)
639             {
640                 ImpFresult(fresult);
641                 return(-1);
642             }
```

```
643
644             // Imprime el último fragmento del archivo que se recibió.
645             UARTprintf("%s\n", g_cTmpBuf);
646
647             // Continua a menos que el numero total de bytes se hayan leído.
648             // Eso significará que el final del buffer se ha encontrado.
649         }
650
651
652     }
653     while(ui32BytesRead == sizeof(g_cTmpBuf) - 1);
654
655
656     if(fresult != FR_OK)
657     {
658         ImpFresult(fresult);
659         return(-1);
660     }
661
662     fresult = f_close(&g_sFileObject);
663
664     if(fresult != FR_OK)
665     {
666         ImpFresult(fresult);
667         return(-1);
668     }
669
670     // Devuelve éxito.
671     return(0);
672
673 }
```

Ilustración 5. 3. Función CreaArchDatos()

Si nos fijamos detenidamente en la sección de la creación de archivos veremos que existe la posibilidad de que se cree un fichero con el mismo nombre, en cuyo caso se comenta *"Fichero existente"* pero si hablamos de fechas y horas es imposible que un dato pueda repetirse, se podría pensar que es por protección de los datos pero lo cierto es que se usa para crear una serie de subarchivos.

Se explicó en el capítulo anterior que los datos estaban fragmentados por un bucle *for* en 3 partes. Este se acomete debido a que un buffer excesivamente grande no puede ser soportado por la librería FATfs (se dan parones y problemas) por tanto se pensó en dicha solución. Para que los datos se mantengan en una misma carpeta horaria es necesario indicar que están fragmentados, no queremos que parte de los datos del segundo 1 se almacenen en la carpeta del segundo 2 o similar (luego se pondrá una figura vislumbre este apartado). Una vez indicado con la variable *MismaCarpeta* se crea el fichero correspondiente llamado *Datos_1*, si se ha almacenado alguno de estos ficheros se activa el mensaje *"Fichero existente"* y se crea el archivo *Datos_2*, en la misma carpeta horaria con la continuación de los datos, y así sucesivamente.

Para acabar con esta sección falta reseñar la función *GetTimeSystick* y el acceso a datos.

En el primer caso hablamos de una función que realiza las gestiones horarias de los días, meses y años. Como no hemos aplicado un RTOS es necesario conseguir una fuente de reloj que nos marque el tiempo. Esa es la función del Systick, un temporizador simplificado del sistema que nos acrecienta la cuenta de los segundos pasados en cada instante por medio de interrupciones. Entonces la función *GetTimeSystick* es la encargada de interpretar la cuenta del Systick y de procesar las fechas y horas que en un inicio hemos introducido al iniciar/resetear el micro-controlador.

En cuanto al acceso a datos es tan sencillo como conectar el USB al ordenador y obtener los resultados almacenados. En caso de extraer el USB con el microcontrolador funcionando se manda la respuesta a la función principal para que deje de transmitir datos al dispositivo de almacenamiento. Y no interrumpa el proceso.

A continuación se mostrará un vistazo de cómo queda la distribución de archivos:

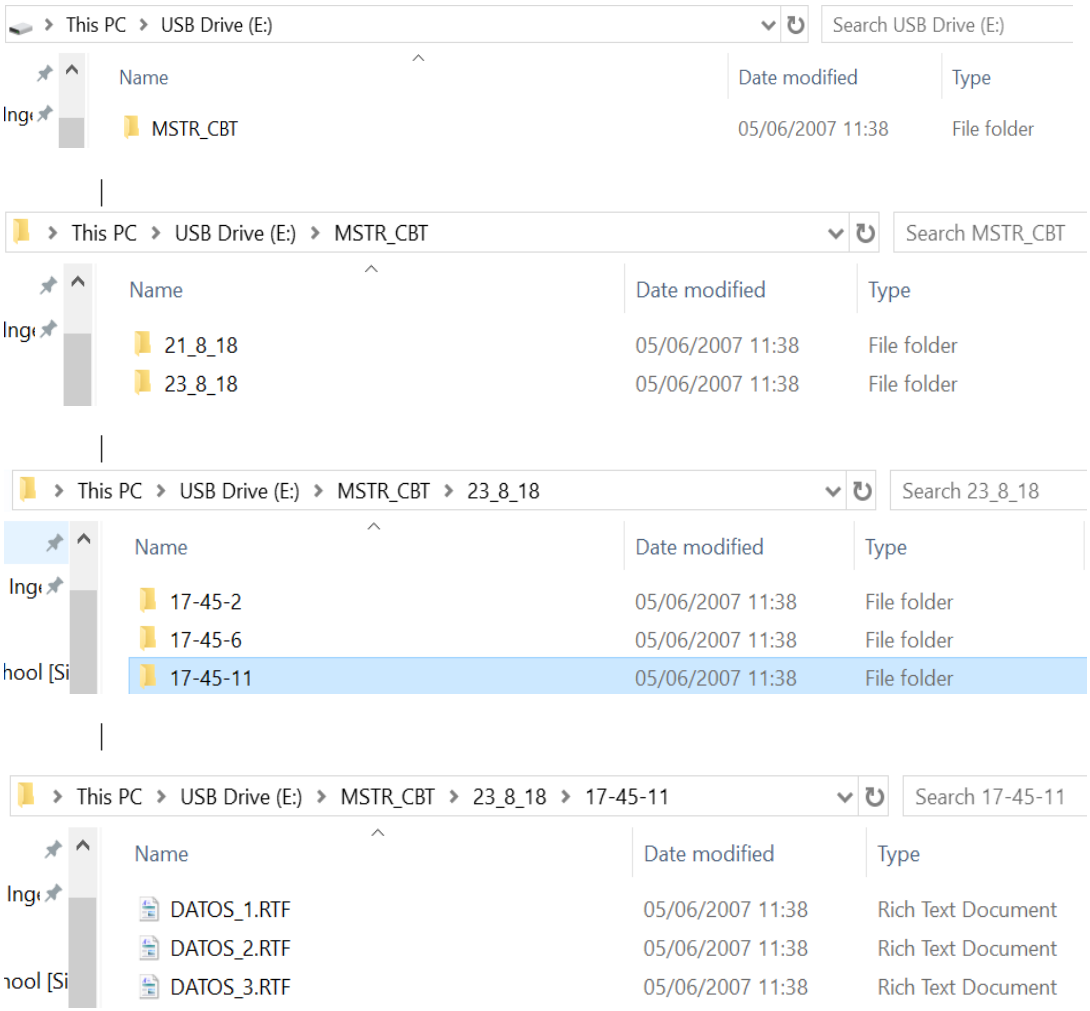


Ilustración 5. 4. Árbol de carpetas de almacenaje

Se puede vislumbrar con facilidad las carpetas de fecha y las carpetas de hora con los datos contenidos. Se ve también que dichos datos están almacenados en formato *.rtf* (Rich Text File), se ha escogido este formato por 2 motivos, el primero es que es un formato muy extendido y accesible por muchas aplicaciones, el segundo es que prevalece por encima del formato *.txt* porque presenta estructura y se pueden almacenar los datos de forma más ordenada y simple para la vista.

5.2. Comunicación

En esta parte veremos las formas que tiene para comunicarse nuestro micro-controlador por medio de los algoritmos.

5.2.1. Comunicación Bluetooth

La comunicación bluetooth se configura al inicio del programa en la función *ConfigSystem*, se mostró la línea en la que se seleccionaba el modo de operación del puerto UART al principio del capítulo 4. Hablamos de ésta línea:

```
214
215     //  UART por puerto USB (ICDI)          (No funcionan los 2 a la vez si se usa la librería uartstdio.h)
216 //  ConfigUARTUSB();
217
218     //  UART por Bluetooth (PA6 y PA7)
219     ConfigUARTBluetooth();
220
221     MAP_IntMasterEnable();                                  //Habilitación global de interrupciones
```

Ilustración 5. 5. Selección Bluetooth

Como se puede ver se dan 2 opciones. La primera configura el UART por el puerto de debug, la segunda por bluetooth. La primera no tiene más interés que en el caso de disponer, o no, de bluetooth se puede hacer una comprobación de comunicación conectado el micro al ordenador mientras procesa los datos. Pero dicho método obliga a que el operario mantenga la conexión por cable con el ordenador en todo momento pues se alimenta de la energía que éste procura. Para un funcionamiento óptimo es aconsejable manejar el bluetooth y conectar el micro a una fuente de alimentación independiente.

Las líneas de código que rigen esta configuración son las siguientes:

```
103 void ConfigUARTUSB(void)
104 {
105     //  Declaramos que el pin PA0 será de lectura para la entrada de datos de la UART 0
106     MAP_GPIOPinConfigure(GPIO_PA0_U0RX);
107     //  Declaramos que el pin PA1 será de escritura para la salida de datos de la UART 0
108     MAP_GPIOPinConfigure(GPIO_PA1_U0TX);
109     //  Pines y puerto usado por la UART para transferencia de datos
110     MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
111     //  Declaramos oscilador externo de 16MHz para el UART
112     MAP_UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);
113     //  Configuración de la UART 0 a 115200bps, sin paridad, 1 bit de stop y 8bits de datos.
114     MAP_UARTConfigSetExpClk(UART0_BASE, 16000000, 115200, UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE
115             | UART_CONFIG_PAR_NONE);
116     MAP_UARTEnable(UART0_BASE);
117     UARTStdioConfig(0, 115200, 16000000);
118 }

121 void ConfigUARTBluetooth(void)
122 {
123     //  Declaramos que el pin PA6 será de lectura para la entrada de datos de la UART 2
124     MAP_GPIOPinConfigure(GPIO_PA6_U2RX);
125     //  Declaramos que el pin PA7 será de escritura para la salida de datos de la UART 2
126     MAP_GPIOPinConfigure(GPIO_PA7_U2TX);
127     //  Pines y puerto usado por la UART para transferencia de datos
128     MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_6 | GPIO_PIN_7);
129     //  Declaramos oscilador externo de 16MHz para el UART
130     MAP_UARTClockSourceSet(UART2_BASE, UART_CLOCK_PIOSC);
131     MAP_UARTFIFOEnable(UART2_BASE);
132     //  Configuración de la UART 2 a 9600bps, sin paridad, 1 bit de stop y 8bits de datos.
133     MAP_UARTConfigSetExpClk(UART2_BASE, 16000000, 9600, UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE
134             | UART_CONFIG_PAR_NONE);
135     MAP_UARTEnable(UART2_BASE);
136     UARTStdioConfig(2, 9600, 16000000);
137 }
```

Ilustración 5. 6. Funciones de selección de Bluetooth

Además de lo comentado se explicó que hay dos modos independientes de visualización, *VisualizarUSB* y *VisualizarADC*. Ambas variables se determinan en el código principal según el botón que se aprete. Si se presiona el botón 1 se muestra la información que concreta sobre el USB; si se presiona el botón 2 se despliega la información que concierne al CAD.

Se puede visualizar en éste fragmento del código dicho proceso:

```
253     UARTprintf("\n Aprete al botón 1 para Información USB");
254     UARTprintf("\n O aprete al botón 2 para despliegue de Información ADC\n");
255     while (!B1_ON && !B2_ON);
256
257     //  Habilitamos el Timer encargado de la frec. de muestreo
258     TimerEnable(TIMER0_BASE, TIMER_A);
259     //  Habilitamos el timer que dicta cada cuando se inicia el muestreo
260     TimerEnable(TIMER1_BASE, TIMER_A);
261
262     while(1)
263     {
264         //  Esta condición decide si se visualiza la información del USB o del ADC
265         if(B1_ON) {VisualizarUSB = true; VisualizarADC = false; UARTprintf("\033[2J\033[H");}
266         else if (B2_ON) {VisualizarUSB = false; VisualizarADC = true; UARTprintf("\033[2J\033[H");}
```

Ilustración 5. 7. VisualizarUSB, VisualizarADC

5.2.2. Comunicación Ethernet

Nota: En este punto pese a haberse realizado funciones y códigos no se va a mostrar dicho código referente a la configuración y transmisión por Ethernet pues no se ha logrado que se establezca una conexión aunque en unos primero intentos se consiguió una conexión no efectiva con la obtención de una IP por parte del router. Por tanto es una sección fallida y sujeta a futuros cambios. En la versión final del proyecto dicho código se denomina *comunicador*, se trata como un mero apéndice sin estar sujeto a compilación para una posterior rectificación y mejora del mismo.

Éste comprende uno de los procesos más complejos del código. La idea en sí es simple. Hacemos uso del puerto Ethernet de la placa del micro-controlador para mandar los datos a una página de web. En este punto se tiene 2 opciones, o usar al propio micro-controlador como servidor web almacenando su propia página web, o usar una página de visualización para mandar la información y que está lo interprete, que es lo se ha pretendido. Es una opción más simple y menos engorrosa que la primera. Para éste caso se usa el GUI Composer v2, es una aplicación diseñada por Texas Instruments para crear tu propio panel visualizador y de control web.

Aquí queda el diseño de nuestra web de visualización:

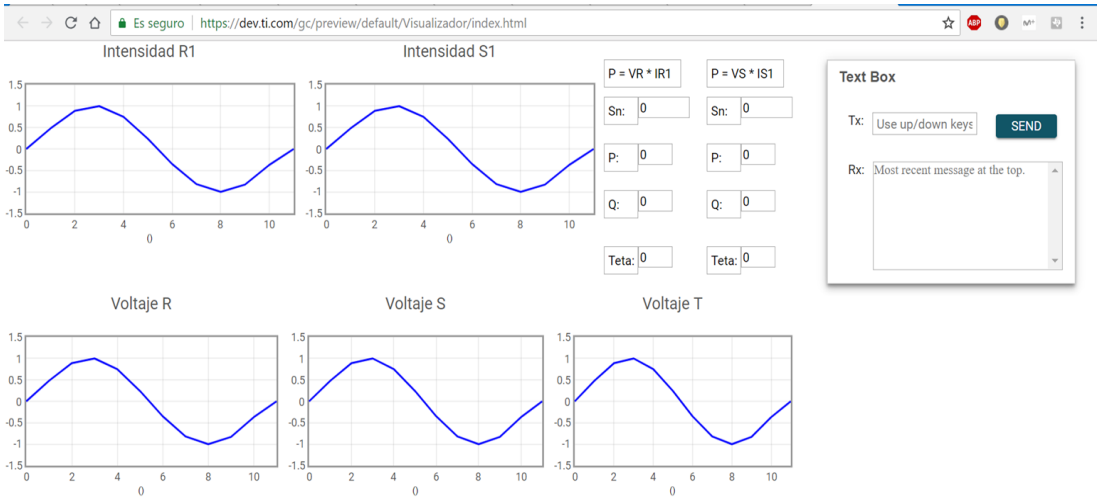


Ilustración 5. 8. Diseño web de visualización (Visualizador)

Los datos que se trasmiten en este caso son distintos a los de la cadena *Cdatos*, excepto los valores de potencia pasados por la matriz *Mdatos* el resto de valores que se mandarían serían los concernientes a la estructura de vectores *Muestras* donde contendría el valor de todos los puntos muestreados y luego representados en las gráficas de arriba. El nombre de la función para la transferencia de estos datos es *Comunicador* y la llamada a dicha función se encuentra en la misma sección de la transmisión y el envío de datos.

```
289                 }
290                 MismaCarpeta = false;
291             }
292             LED_OFF;
293
294     //      res = Comunicador(Muestras, Mdatos);
295
296             MuestraCmpltd = false;
297         }
298
```

Antes de mostrar dicha llamada se debe señalar la declaración de configuración. Situada unas líneas antes que la configuración del USB se encuentra *ConfigComunicador*, dicha función realiza la tarea de conectarse al router y obtener una IP válida para poder establecer la comunicación.

```
239     UARTprintf("Pulse el botón 1 para continuar");
240     while (!B1_ON);
241     UARTprintf("\033[2J\033[H");
242
243 //    UARTprintf("Configurando Ethernet\n");
244 //    res = ConfigComunicador();
245 //
246 //    UARTprintf("\n Acabada configuración\n");
247 //    while(!B2_ON);
```

Ilustración 5. 9. Llamada a ConfigComunicador(). Función Main

Luego la transmisión de datos se realiza mediante el protocolo MQTT de mensajes de red. Es en este punto donde se encarga la función *Comunicador* de tratar los datos en este formato para ser procesado por la página de visualización.

Todas las funciones comentadas se pueden encontrar en la hoja de código *Comunicador.c*.

6 RESULTADOS Y CONCLUSIONES

En este punto mostraremos resultados obtenidos en pruebas varias y entornos simulados. Desde el muestreo de las ondas que se obtienen hasta la resolución final de los datos tratados en el dispositivo de almacenamiento.

Primero debemos aclarar que todas las pruebas y experimentos se han realizado con bobinas. La idea parece obvia, con una corriente menos peligrosa que en una CBT se pueden obtener campos magnéticos considerables. Con esto se pretende confirmar que el sistema funciona frente a estos campos pero no se ha probado en un entorno real dentro de una caja distribuidora de baja tensión.

Lo mismo se puede decir del circuito de tensión, no se ha trabajado con altos voltaje, en este caso se han usado simulaciones de la onda característica como se verá más adelante.

Entonces, primero pasaremos a comentar sobre las muestras de laboratorio que se obtuvieron concernientes a experimentos con campos magnéticos y, por ende, con las intensidades.

La bobina que se muestra en el lateral fue la usada para dichos experimentos, esta bobina estaba alimentada a una fuente de 50 Hz de bajo voltaje con aproximadamente 6 amperios. No se está buscando una medida exacta de la intensidad, lo que se pretende conseguir con el experimento es ver si nuestro sistema reacciona y caracteriza, como es debido, el campo magnético ondulante.

Ilustración 6. 1. Bobina de laboratorio

A continuación se mostrará un extracto de las muestras que se obtuvieron en laboratorio para el caso de la intensidad.

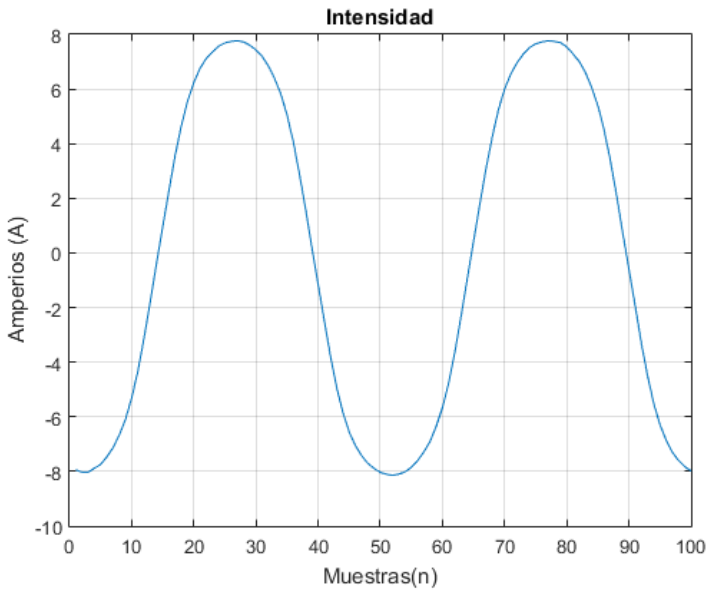


Ilustración 6. 2. Onda de intensidad resultante

Dicha onda se obtuvo a partir de las muestras obtenidas en el vector *Muestras.IS1[]*, en concreto se aprecia que fueron 100 muestras. Se puede ver una onda bien caracterizada a 50Hz.

Ahora pasamos al voltaje, en nuestro circuito de voltaje que vimos en el capítulo 2 se le ha realizado una simulación para ver la onda que muestrearía nuestro micro-controlador. Los resultados son:

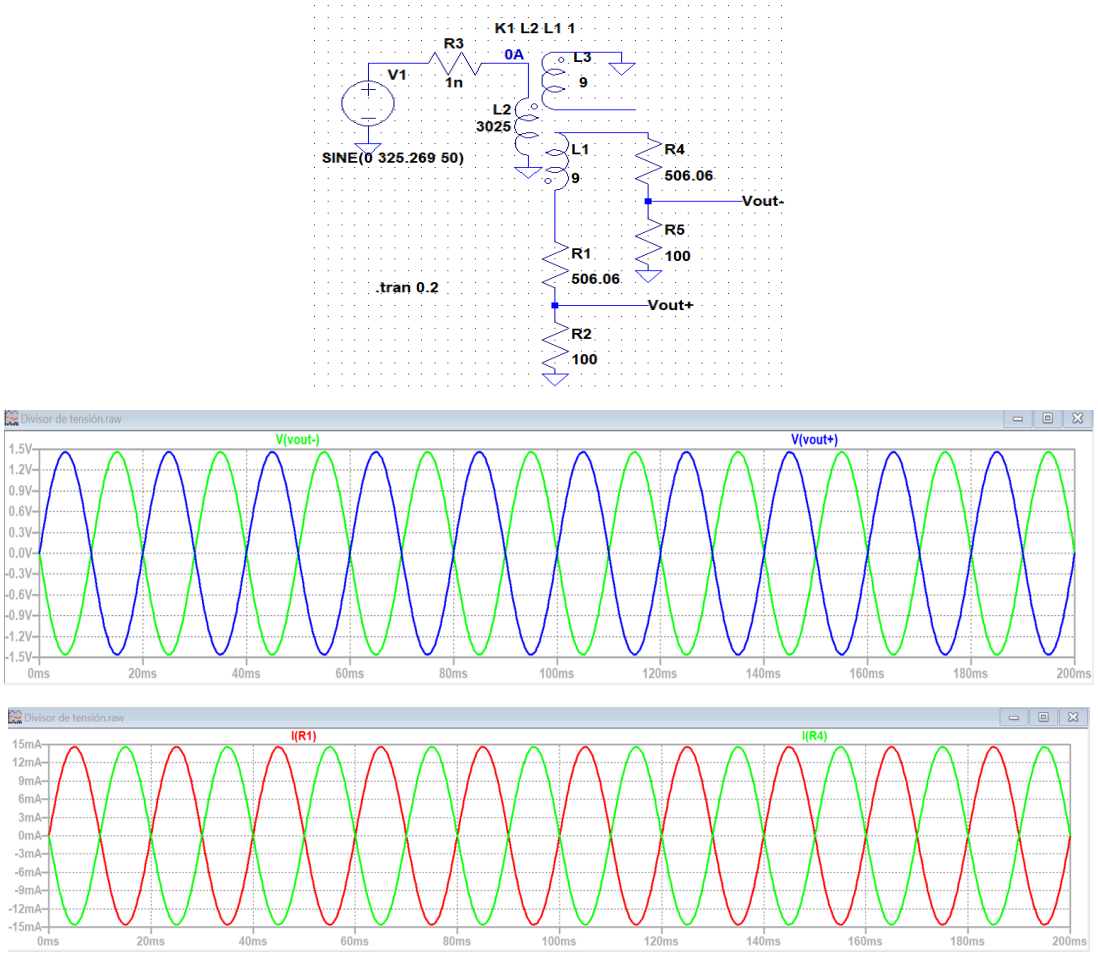


Ilustración 6. 3. Esquema y simulaciones de voltaje

Vemos que se ha simulado una fuente con una amplitud de 325.269, es decir, con una amplitud de $\sqrt{2} \cdot 230\ V$.

Vamos a continuar con una prueba realizada para ver cómo se comporta todo el sistema. El circuito a analizar es el siguiente, cabe destacar que todas estas pruebas se **han acometido en voltaje de continua**:

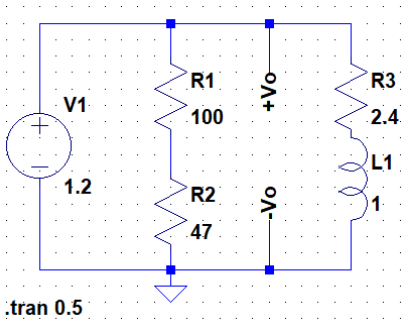


Ilustración 6. 4. Circuito de prueba en simulación

De éste circuito obtenemos los siguientes resultados simulados:

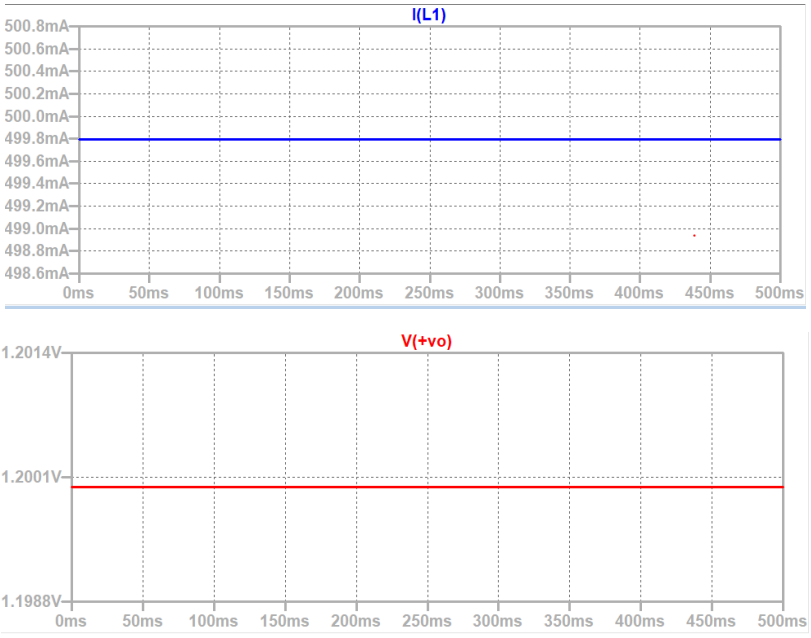


Ilustración 6. 5. Simulaciones del circuito de prueba

Ahora pasaremos a comprobar si se corresponden los resultados simulados con los obtenidos. Entonces pasaremos a realizar experimentalmente el circuito y mostraremos también un extracto de la información de la pantalla del terminal serie (comunicación UART) con los datos obtenidos y su posterior almacenaje en el USB.

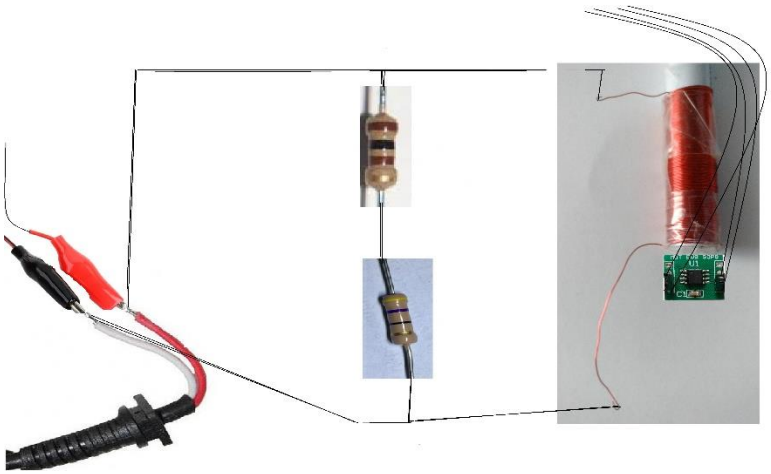


Ilustración 6. 6. Circuito de montaje simple

Entonces, empezaremos mostrando la intensidad IR1, el voltaje VR y la potencia derivada de ambos vista en el terminal serie:

```
COM6
        Análisis AC
Temperatura del aquipo: 39.52 ºC
ADC ->
  Muestras: 200 por entrada
  Frecuencia de muestreo: 5000 Hz
  Ratio de actualización: 5 seg
        Corriente IR1
  Tipo: diferencial
  Pines de entrada: AIN0/PE3 - AIN1/PE2

Ampl =   0. 3 A   I =   0.47 A   Offset =  0.48 A   Frec =  0. 0 Hz
```

```
COM6
        Análisis AC
Temperatura del aquipo: 39.88 ºC
ADC ->
  Muestras: 200 por entrada
  Frecuencia de muestreo: 5000 Hz
  Ratio de actualización: 5 seg
        Voltaje VR
  Tipo: diferencial
  Pines de entrada: AIN10/PB4 - AIN11/PB5

 Ampl =   0.54 V   V =   1.20 V  Offset =  0.74 V   Frec =  0. 0 Hz
```

```
COM6
        Análisis AC
Temperatura del aquipo: 43.75 ºC
ADC ->
  Muestras: 200 por entrada
  Frecuencia de muestreo: 5000 Hz
  Ratio de actualización: 5 seg
        Potencia VR * IR1
 Sn =     0.58 VA   P=     0.58 W   Q =     0. 0 VAr   Teta =   0. 0 º
```

Ilustración 6. 7. Resultados Terminal Serie

Vemos que los datos obtenidos se corresponden, se destaca cierto desvío en las muestras obtenidas por la falta de precisión sobre todo para muestrear el campo magnético y la forma de tratar los datos.

Seguido de esto accedemos accedemos al archivo para ver que se verifica.

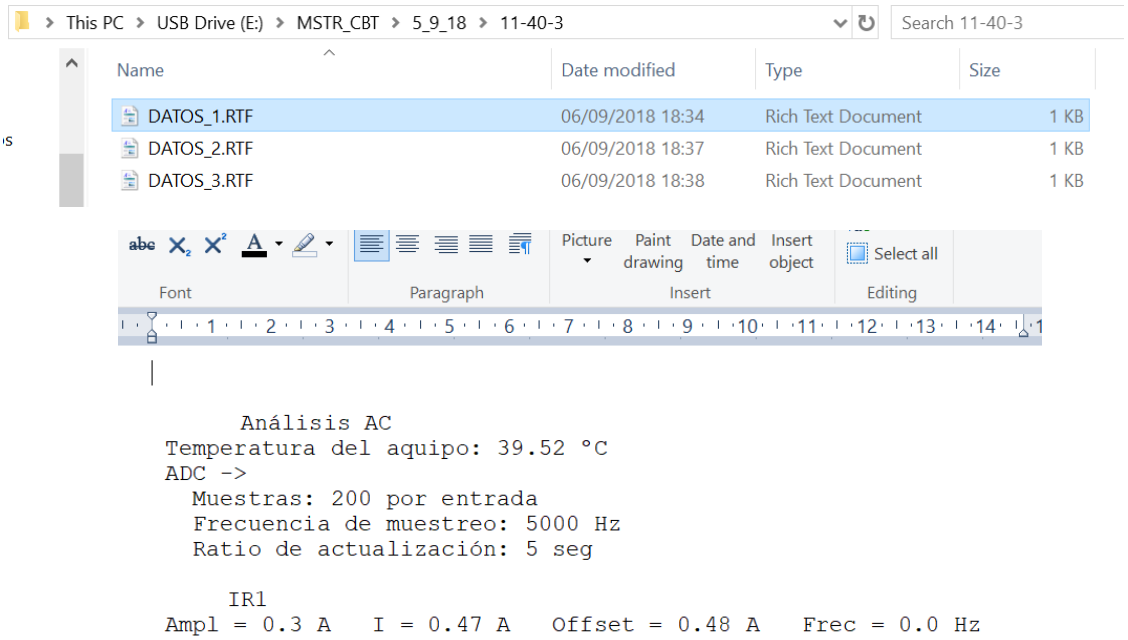


Ilustración 6. 8. Comprobación de archivos de intensidad

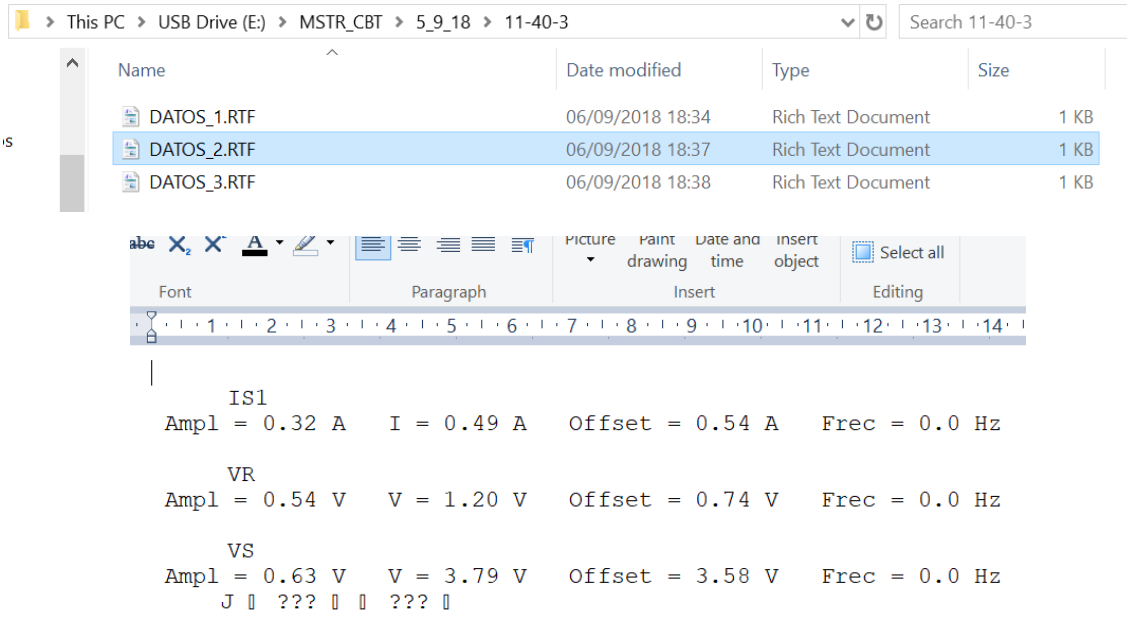
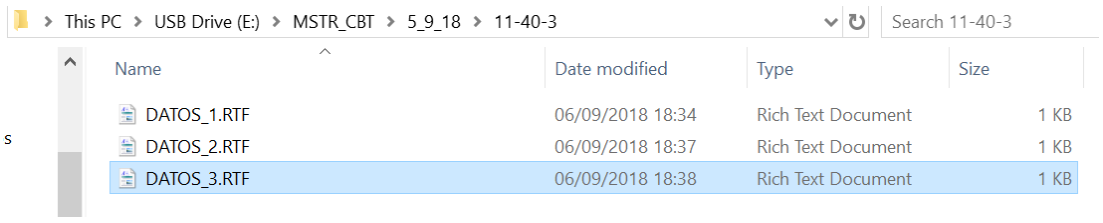


Ilustración 6. 9. Comprobación de archivos de voltaje



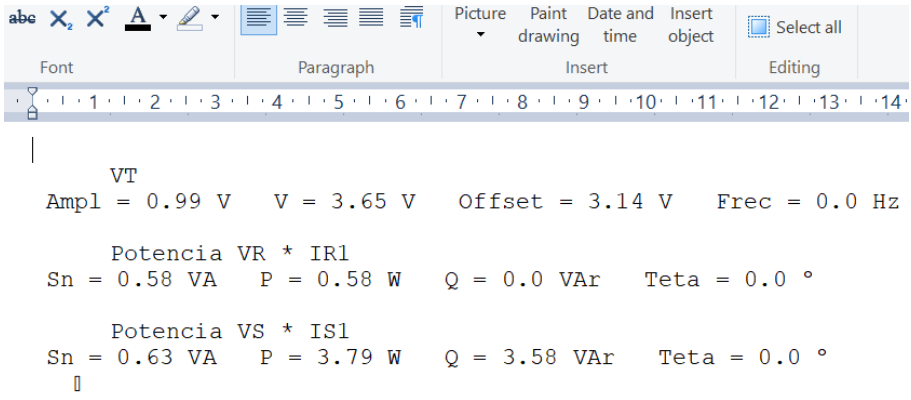


Ilustración 6. 10. Comprobación de archivos de potencia

Vemos que efectivamente las muestras se guardan debidamente en el pen drive.

7 REFERENCIAS

Texas Instrumets (Ed.). (2016a) *Tiva™ C Series TM4C1294 Connected LaunchPad Evaluation Kit: User's Guide.*

Texas Instrumets (Ed.). (2016b) *EK-TM4C1294XL Firmware Development Package: User's Guide.*

Texas Instrumets (Ed.). (2016c) *TivaWare™ Peripheral Driver Library: User's Guide.*

Texas Instrumets (Ed.). (2016d) *TivaWare™ USB Library: User's Guide.*

Texas Instrumets (Ed.). (2014) *Tiva™ TM4C1294NCPDT Microcontroller: Data Sheet.*

MultiDimension Technology Co., Ltd. () *TMRLINEAR SENSOR MMLH45F.* Ver.1.1

YHDC (Ed.). (2015) *PE3010-M: Product Specification.*

Texas Instrumets. () *IoT Embedded Web Server: Enet_IO Step by Step Example Slides.*

CIDEL Argentina (2014) *ATENUACION DE CAMPOS MAGNETICOS EN ESTACIONES TRANSFORMADORAS Y CENTROS DE TRANSFORMACION.* Departamento de Ingeniería Eléctrica, Universidad Tecnológica Nacional.

Chung-Kuan Cheng () *Interconnected and packaging. Lecture 3: Skin Effect.* UC San Diego.

Isend S.A. (2016) *PROFUNDIDAD DE PENETRACIÓN DE LAS CORRIENTES INDUCIDAS.*

Gaspar Fernández (2012) *Poesía Binaria. Algoritmos: Formas de transformar un entero a cadena en C y C++.*

Juan Manuel Oliveras Sevilla (2010) *Técnica Industrial. Blindaje electromagnético de estaciones transformadoras.*

cppreference.com (Ed.). (2018)

ChanN (Ed.). (2018) *Electronic Lives Mfg. FatFs – Generic FAT Filesystem Module.*

Texas Instruments () *CC3200 Thermometer Dashboard. Visualize your LaunchPad temperature data on a GUI Composer Dashboard.*

Ángel Franco García () *Procedimientos Numéricos. Longitud de una elipse.*

Josep Balcells, Francesc Daura, Rafael Esparza y Ramón Pallás (1992) *Interferencias electromagnéticas en sistemas electrónicos.*

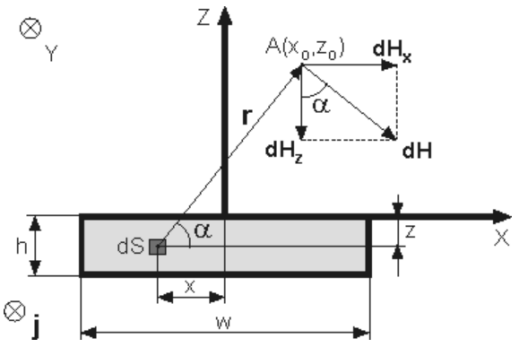
www.wikipedia.com () *Hierro.*

www.wikipedia.com () *Red eléctrica.*

Juan C. Olivares-Galván1, Iván Hernández, Pavlos S. Georgilakis, Eduardo CamperoLittlewood (2009) *Calculation of the Magnetic Field Intensity in a Rectangular Conductor Carrying Current in Electromagnetism Introductory Courses.* The COMSOL conference 2009, Boston.

NT-MDT Spectrum Instruments (2015) *MAGNETIC FIELD OF RECTANGULAR CONDUCTOR WITH CURRENT.*
https://www.ntmdt-si.com/resources/spm-theory/theoretical-background-of-spm/2-scanning-force-microscopy-(sfm)/27-magnetic-force-microscopy-quantitative-results-treatment/279-magnetic-field-of-rectangular-conductor-with-current

Anexo 1:
Campo magnético de una sección rectangular



De acuerdo con la ley de Biot-Savart-Laplace, el campo magnético de un alambre infinitamente largo y delgado atravesado por corriente a una distancia *r* en coordenadas Gaussianas es dada por:

$$H = \frac{\beta \cdot J}{r}$$

Donde $\beta = \frac{2}{c}$, siendo *c* la velocidad de la luz y *J* la corriente que atraviesa el cable. Coincidiendo el vector del campo magnético con el producto vectorial de $\bar{J}\ x\ \bar{r}$.

Teniendo una sección $dS = dx \cdot dz$ como la figura de encima podemos describir el campo magnético del alambre en un punto $A(x_o, y_o, z_o)$ como sigue:

$$dH_x(x_o, z_o) = \frac{\gamma \sin\alpha}{r} \cdot dx \cdot dz; \quad dH_y(x_o, z_o) = 0; \quad dH_z(x_o, z_o) = \frac{-\gamma \cos\alpha}{r} \cdot dx \cdot dz;$$

Donde $\gamma = \beta \cdot j$, siendo *j* la densidad de corriente y $r = \sqrt{(x_o - x)^2 + (z_o + z)^2}$ como la distancia más corta al punto A, α como el ángulo entre el vector $\bar{r}$ y el eje X, y

$\sin\alpha = \frac{z_o+z}{\sqrt{(x_o-x)^2+(z_o+z)^2}}$, $\cos\alpha = \frac{x_o-x}{\sqrt{(x_o-x)^2+(z_o+z)^2}}$.

Dicho esto el campo magnético en un punto $A(x_o, z_o)$ se puede calcular integrando las derivadas del campo arriba mencionadas.

$$H_x(x_o, z_o) = \gamma \int_0^h \int_{-\frac{w}{2}}^{\frac{w}{2}} dH_x(x_o, z_o) \cdot dx \cdot dz; \quad H_z(x_o, z_o) = \gamma \int_0^h \int_{-\frac{w}{2}}^{\frac{w}{2}} dH_z(x_o, z_o) \cdot dx \cdot dz;$$

$$\downarrow \qquad p = x_o - x;$$

$$\bar{H}_x(p_o, z_o) = \gamma \int_0^h \int_0^{p_o} \frac{z_o + z}{p^2 + (z_o + z)^2} \cdot dp \cdot dz;$$

$$\bar{H}_z(p_o, z_o) = \gamma \int_0^h \int_0^{p_o} \frac{-p}{p^2 + (z_o + z)^2} \cdot dp \cdot dz;$$

$$\downarrow$$ Expresado analíticamente queda:

$$\bar{H}_x(p_o, z_o) = \frac{\gamma}{0{,}2}\left(p_o \ln\left(1 + \frac{h^2 + 2z_o h}{{p_o}^2 + {z_o}^2}\right) + 2(z_o + h) \cdot \arctan\left(\frac{p_o}{z_o + h}\right) - z_o \cdot \arctan\left(\frac{p_o}{z_o}\right)\right);$$

$$\bar{H}_z(p_o, z_o) = \frac{\gamma}{0{,}2}\left(z_o \ln\left(1 + \frac{{p_o}^2}{{z_o}^2}\right) - (z_o + h)\right.$$
$$\left. \cdot \ln\left(1 + \frac{{p_o}^2}{(z_o + h)^2}\right) - 2p_o\left(\arctan\left(\frac{z_o + h}{p_o}\right) - \arctan\left(\frac{z_o}{p_o}\right)\right)\right);$$

Finalmente tenemos que el campo magnético es:

$$H_x = \bar{H}_x(w/2 + x_o,\ z_o) - \bar{H}_x(x_o - w/2,\ z_o);$$
$$H_z = \bar{H}_z(w/2 + x_o,\ z_o) - \bar{H}_z(x_o - w/2,\ z_o);$$
$$|H_{total}| = \sqrt{H_x^2 + H_z^2};$$