UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño, construcción y puesta a punto de un Teslámetro mediante el sistema Arduino

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jose Galbis Soler

Tutor: Jorge Más Estellés

2016/2017

Resumen

Este proyecto pretende obtener un Teslametro de bajo coste, elaborado a partir de hardware y software libre, escogiendo como plataforma abierta Arduino junto con otros dispositivos y intentado reutilizar objetos desechados en casa. Todo ello para poder disponer de un dispositivo para el laboratorio de física Nikkola Tesla de la ETSINF; elaborado desde la misma universidad para el desarrollo de la universidad.

Desarrollaremos el análisis, diseño, implementación y testeo del dispositivo resultante, poniendo en practica distintas técnicas adquiridas durante el grado de ingeniería informática. Además de utilizar distintos buses de comunicación y corregiremos el efecto rebote de las señales eléctricas. En definitiva la resolución completa de proyecto de ingeniería.

Palabras clave: Arduino, Hardware libre, Teslametro, Instrumentación de laboratorio, Educativo.

Resum

Aquest proyecte preten obtindre un Teslametre de baix cost, elaborat a partir de hardware y software lliure, escollint com a plataforma oberta Arduino junt amb altres dispositius i intentant reutilitzar objectes en des ús de casa. Tot açó per poder disposar d'un dispositiu per al laboratori de fisiaca Nikkola Tesla de la ESTINF elaborat desde la mateixa universitat y per al desenvolupament de la universitat.

Desenvoluparem el analisí, disseny, implementacio e testeg del dispositiu resultant, posant en practica distintes tecniques adquirides al grau de ingenieria informatica. Ademes de utilitzar diversos busos de comunicació y corregirem el efecte rebot produït a les señals electriques. En definitiva la resolucio completa d'un proyecte de ingenieria.

Paraules Clau: Arduino, Hardware lliure, Teslametre, Instrumentació de laboratori, Educatiu.

Abstract

The objective of this project is to obtain a low cost Teslameter, made with free hardware and free software. Arduino has been chosen as an open platform along with other devices, and we have tried the recycling of discarded items at home. All this in order to have a device for the Nikkola Tesla Physics laboratory at ETSINF: created from university for the development of the University.

We have developed the analysis, design, implementation and testing of the resulting device, putting into practice different techniques learned during the degree on computer engineering. Moreover we have used different buses of communication and we have corrected the rebound effect of electrical signals. In conclusion, this work involves a complete resolution on an engineering project.

Keywords : Arduino, Open hardware, Teslameter, Laboratory instrumentation, educational.

Índice

Índice de figuras

Índice de tablas

Capitulo 1 : Introducción

En este trabajo vamos a aproximarnos a la fabricación de un Teslámetro. Intentado que sea fácil de montar, programar y reparar, destinado al uso educativo en los laboratorios de física de la ETSINF.

1.1 Motivación

Con la aparición del hardware libre o hardware abierto, se ha dado una puerta al desarrollo de instrumentación de laboratorio. Donde a partir de diseños libres, se puede ir creando distintos aparatos como Teslametros, Voltímetros, Amperímetros, Generador de señales, Osciloscopios, Brazos robots, Impresoras 3D... con los cuales promulgar la investigación científica de un coste mucho más asequible para estudiantes y emprendedores. Como al mismo tiempo aporta una capacidad creativa de adaptación de los instrumentos elaborados a las necesidades particulares de cada persona.

Es evidente que no alcanzan la precisión y estabilidad que los equipos diseñados industrialmente, ni tampoco se pretende la suplantación de la industrial productora de estos dispositivos. Solo aportar dispositivos para aquellos que no pueden permitirse uno privado con un elevado coste y promover la reutilización de desechos para el montaje final de los dispositivos.

1.2 Objetivos

1. Diseñar e implementar un Teslametro de bajo coste para su uso en un laboratorio de física.
2. Utilizar software libre basado en el sistema Arduino.
3. Utilizar hardware libre basado en el sistema Arduino.
4. Reciclar materiales en su construcción.
5. Verificar su correcto funcionamiento.

1.3 Estructura de la memoria

Con un total de siete capítulos, nos adentramos en el diseño e implementación de un Magnetómetro. En este apartado se comentara brevemente el contenido de cada capítulo.

- Capitulo 1, Introducción: Definimos la motivación y objetivos del trabajo, junto con la descripción del contenido del trabajo y el uso de la bibliografía más destacada.

- Capitulo 2, Análisis: Estudiamos los Teslametros típicos del mercado, para la extracción del funcionamiento y de los requerimientos mínimos de hardware, como además plantear los requerimientos personalizados para el laboratorio. Búsqueda de componentes y comparativas entre similares y por último una descripción de los componentes seleccionados.

- Capitulo 3, Diseño: En este capitulo nos centramos en el diseño lógico del Magnetómetro realizando diagramas de flujo, y del diseño hardware, distinguiendo las distintas conexiones como los buses utilizados, y su caja de montaje junto con el procedimiento de alimentación del dispositivo.

- Capitulo 4, Implementación: Durante este capitulo nos centramos en el desarrollo del dispositivo, incluyendo explicaciones de las implementaciones software realizadas y del montaje final en la caja.

- Capitulo 5, Comprobación: Comprobamos cada una de las conexiones, cada uno de los flujos de datos, y de correcta medición del campo magnético.

- Capitulo 6, Presupuesto: Contabilizamos el precio de los componentes y el tiempo empleado en el desarrollo de la aplicación mediante diagramas de Gannt. Además procederemos a comparar el dispositivo resultante con distintos Teslametros comerciales.

- Capitulo 7, Resultados y conclusiones: Este último capitulo describe los problemas y dificultades encontradas durante el desarrollo del dispositivo y sugerencias de ampliación del dispositivo y del trabajo.

Cabe destacar la existencia de varios anexos en el documento, como son la bibliografía utilizada y el código resultante del Teslametro.

1.4 Uso bibliográfico

En este apartado vamos a explicar del material bibliográfico utilizado para elaborar el proyecto.

- En primera instancia empleamos los recursos de la Efecto Hall **[13]**, RAE **[14]**, Magnetómetro **[16]** y Wikipedia **[17]** para introducirnos en los Teslámetro, y así poder analizarlos.

- Una vez clara la idea de los magnetómetros, procederemos a buscar dispositivos con los que confeccionar el proyecto. Para ello empleamos los recursos Arduino **[7]** y RsComponentes **[15]**.

- Habiendo escogido los componentes, necesitamos información acerca de ellos y su funcionamiento. Para ello recurrimos a los recursos de Adafruit RTCLib **[1]**, Arduino AnalogReadResolution **[2]**, Arduino Due **[3]**, Arduino Forum fat16lib **[4]**, Arduino HallEfect **[5]**, Arduino LiquidCrystal **[6]**, Arduino SD **[8]**, Arduino SPI **[9]** y Arduino Wire **[10]**. En estos recursos es consultada toda la información necesaria para entender el funcionamiento de los distintos componentes y de como ensamblarlos todos para componer el dispositivo.

- Por último, para dar precio a las horas dedicadas al trabajo y aproximarnos al mundo laboral, damos uso al recurso del Boletín oficial del estado **[11].** En el documento oficial del estado queda declarado el sueldo mínimo correspondiente con los distintos trabajos relacionados con la consultoría informática.

Capitulo 2 : Análisis

Durante el segundo capitulo se va a proceder a investigar y analizar el funcionamiento de un Teslámetro. Procederemos a buscar los Teslametros existentes en el mercado, extraeremos los requerimientos hardware, y procederemos a buscar y seleccionar los componentes necesarios para el desarrollo del dispositivo.

2.1 Introducción a los Teslámetros

A continuación, realizaremos una aproximación a los dispositivos de medición del campo magnético. Primero analizamos un poco más su significado, para ello recurrimos a la real academia de la lengua española, en ella descubrimos que Teslámetro no existe como tal, la palabra oficial en la lengua castellana es Magnetómetro y su significado lo podemos ver en la Figura 2.1.

Magnetómetro:
Aparato que mide la inducción de un campo magnético en una dirección determinada

Figura 2.1: Definición magnetómetro

Figura 2.2: Teslámetro comercial

Un magnetómetro básico como el de la Figura 2.2, dispone de una display LCD, una interfaz de usuario, y una sonda de medición o sensor.

Siguiendo con la introducción en la medición de los campos magnéticos, vamos a preguntarnos por que tipos de sensores existen para la medición de ese campo. Indagando un poco, encontramos que la gran mayoría de sensores de medición de campos magnéticos se basan en el efecto Hall[1].

1 El efecto Hall se produce cuando se ejerce un campo magnético transversal sobre un cable por el que circulan cargas. Como la fuerza magnética ejercida sobre ellas es perpendicular al campo magnético y a su velocidad según la ley de Lorentz, las cargas son impulsadas hacia un lado del conductor y se genera en él un voltaje transversal o voltaje Hall.

Entre ellos podemos destacar los escalares que miden la fuerza en una única dirección del vector B de campo magnético al que son sometidos; y los vectoriales que tienen la capacidad de medir todas las componentes del vector del campo, concretamente las componentes (X,Y, Z).

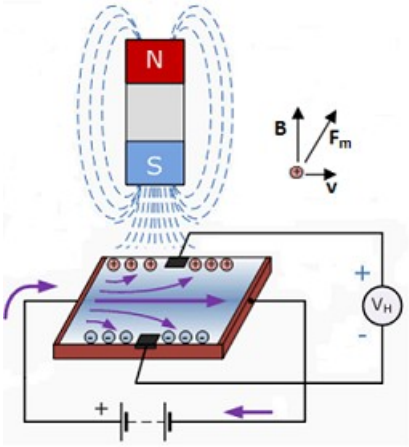

Figura 2.3: Efecto Hall

2.2 Requerimientos del dispositivo

En este apartado vamos a describir que necesidades presenta un Magnetómetro básico, como el de la Figura 2.2. Para ello nos valemos de uno convencional y le añadimos requisitos que creamos convenientes para nuestro magnetómetro, como es el almacenamiento de la información captada.

Leyendo un poco sobre el tema, podemos definir como común; una pantalla por la que imprimir la información, una botonera de interfaz de usuario y un sensor Hall.

Naturalmente nuestro dispositivo no es una copia simple de los Magnetómetros más económicos, al considerar de forma propia los requisitos del Magnetómetro que necesitamos, además de los requisitos observados en los Magnetómetro comerciales, vemos conveniente un almacenamiento de la información de forma detallada y ordenada; con fecha y hora de las medidas tomadas, para permitir una fácil forma de extracción de datos para gráficas.

2.3 Selección dispositivos

Sabiendo nuestras necesidades, es momento de tomar decisiones respecto a como plantearnos el dispositivo.

Primera decisión a tomar es la forma que utilizamos para imprimir la información. Para ello podemos disponer de la comunicación con un PC y aprovechar su pantalla; o integrar una pantalla ligera en el dispositivo. La ausencia de una pantalla integrada en el dispositivo impide que este dispositivo sea portátil, mientras que con una pantalla integrada nos permite desplazarnos donde queramos fácilmente y sin necesidad de un PC.

Definitivamente optamos por integrar una pantalla en el dispositivo. Ahora hay que decidir que tipo de pantalla integramos. En este caso, vamos a considerar cuanta información disponemos para imprimir. Donde recae otra decisión, que sensor Hall vamos a escoger, decidimos utilizar un sensor Hall escalar o lineal con una única medida, esta decisión se comenta más a fondo en el Capitulo 2.3.3.

En conclusión, como tenemos poca información respecto al muestreo, es un valor junto a su unidad y la polaridad de la muestra. Entonces con una pantalla simple es suficiente, y entre otras los display LCD, son económicos, y bastante útiles para representar valores y caracteres.

Otra decisión a tener en cuenta es como implementar la interfaz de usuario. Esta viene condicionada por la primera decisión de como imprimir la información, por el hecho de que en esa decisión escogimos integrar la pantalla para poder prescindir de un PC o otros.

Por ello, siguiendo con el mismo razonamiento, nos vemos obligados a escoger una botonera física integrada en el dispositivo, más adelante explicaremos con más detenimiento las distintas tomas de decisión que surgen a consecuencia de esta última, concretamente en el Capitulo 2.4.6

Respecto al almacenamiento de la información recogida, se cree conveniente el uso de tarjetas microSD; debido a que tienen un uso muy extendido y son económicas; y nos permiten un almacenamiento integrado en el dispositivo siguiendo los patrones anteriormente escogidos.

Para la placa controladora de los distintos dispositivos, escogemos las placas de Arduino, siendo un requerimiento del proyecto además de su vinculación con el mundo universitario, su licencia, y su bajo coste.

Arduino tiene licencia *open-source*, se programa con Arduino Programming Language, dispone de interfaz propia (Arduino Development Enviorement). Conformando una placa de prototipos electrónicos flexible y fácil de usar aunque puede llegar a usarse sin

la interfaz y su lenguaje de programación, utilizando por tanto C++ y aumentado la dificultad al uso pero ampliando la profundidad de comprensión en su funcionamiento.

Con una breve historia, dispone de una larga documentación y módulos compatibles, una gran parte contribuyen con la fundación Arduino, aunque hay muchos otros que son compatibles que no lo hacen, con esta historia hay mucha controversia en la cual no nos inmiscuiremos, aunque en este proyecto tendremos en cuenta a aquellos que si contribuyen con la fundación Arduino. Seguidamente analizaremos brevemente las placas existentes de Arduino y seleccionaremos aquella que más se adecue al proyecto.

2.3.1 Comparativa Arduinos

Existen varias placas con distintas prestaciones, intentando adaptarse a cierto tipo de aplicaciones. Las más comunes son Arduino UNO, Arduino Mega, Arduino Mini, Arduino Nano, Arduino Leonardo,Arduino Due. Existen otras menos comunes como Arduino Lili, Arduino Fio o Arduino Pro Mini.

De entre los más comunes, muchos de ellos tienen un número bajo de pines como Arduino Uno, Arduino Leonardo, Arduino Mini, Arduino Nano, los cuales nos sirven para este proyecto, pero están un poco limitadas para hacer ampliaciones y además nos quedamos un poco cortos con las capacidades de interrupciones hardware, que solo dispone de 2. Mientras que Arduino Mega y Arduino Due dispone de 6 con los pines fijos para la Mega, pero para la Due cualquier pin Digital puede ser configurado como interrupción. Todo esto podemos observarlo en la Tabla 2.3.1

Así que tenemos como posible candidatas Arduino Mega y Arduino Due, ambas sirven para nuestro propósito. Para este proyecto hemos decidido usar Arduino Due, porque dispone de más frecuencia de reloj, de una resolución de 12 bits en sus conversores analógico-digital, permitiéndonos una mejor precisión de medida del campo.

Aunque nos plateemos usar Arduino Due por posibles ampliaciones, debemos tener en cuenta que Arduino Mega también se puede usar y se programara el código de funcionamiento para que sea compatible con ella.

2.3.2 Comparativa Sensores Hall

Hemos de tener en cuenta, que este sensor va a ser el encargado de percibir el campo magnético. Por tanto cuanto mas sensibilidad posea el sensor más precisas serán las muestras obtenidas en las unidades estándares de medición (Tesla y Gauss), pero también nos interesa tener un rango de medida de al menos 200 mT [-100 mT, 100 mT].

Existen distintos tipos de sensores Hall (escalares, vectoriales). Para una primera aproximación a los Magnetómetros básicos, nos centraremos en escoger sensores escalares, son económicos y simples, miden en un eje pudiendo llegar a medir una de las componentes del vector B de campo magnético al que es sometidos dependiendo

del ángulo en el cual se realice la medida. Los sensores Hall vectoriales también son económicos, sencillos, y miden los tres ejes. A pesar de ello es más conveniente emplearlos en futuras ampliaciones del Magnetómetro y así desarrollar uno avanzado.

Teniendo en cuenta que vamos a utilizar una Arduino Due que funciona a 3,3V necesitamos un sensor que trabaje a esa tensión. Muchos trabajan por encima de los 3 V, así que estamos un poco limitados. Aunque en el caso de usar Arduino Mega tendríamos bastante más gama a elegir.

Sensor Hall	Vcc min	Vcc max	Sensitivity
A1318	3 V	3,63 V	1,2-1,4 mv/G
AH49E	3 V	6,5 V	1,1-2,1 mv/G
TSH481	3 V	6,5 V	1,8-2,2 mv/G

Tabla 2.3.2: Comparativa sensores Hall

Hay más opciones a escoger, pero estos tres modelos son los que ofrecían una sensibilidad adecuada para nuestro proyecto. Y como elección final tomamos el A1318 que dispone un voltaje máximo más próximo al máximo de los pines de Arduino Due, de este modo aseguramos el correcto funcionamiento del sensor, ja que los otros dos necesitan el doble de la proporcionada por los pines de Arduino Due.

Por lo tanto con el sensor Hall A1318, tenemos un rango entre [-128 mT, 128 mT], con una sensibilidad de 0,5 mT / b. Mientras que el AH49E tiene [-103,25 mT, 103,25 mT], con un sensibilidad de 0,4 mT / b. Y TSH481 tiene [9-1 mT, 91 mT] con 0,35 mT/b.

2.3.3 Comparativa SD/MicroSD

Para finalizar las comparativas, hablaremos un poco sobre las tarjetas SD y las microSD. Aunque a primera vista parezcan iguales pero distintas en tamaño, ocultan otras pequeñas diferencias. Existen distintos tipos de estas tarjetas, SD, SDHC y las SDXC. Además de distintas clases (2, 4, 6 y 10). Cada una tiene unas propiedades distintas de capacidad y velocidad de transmisión.

Arduino accede a las tarjetas utilizando el bus SPI a una velocidad máxima de 8 MHz sin acceso dinámico de memoria (DMA), por lo que nunca obtendrá una velocidad de transferencia superior a unos 500 KB/seg, aproximadamente la mitad de la velocidad

del bus estándar. En la práctica la velocidad de lectura máxima de un archivo es de aproximadamente de 300 KB/ seg.

La mayoría de PC's, Macs o otros dispositivos acceden a las tarjetas usando el bus a una frecuencia de reloj de hasta 50 MHz y el bus es de 4 bits de ancho de banda por lo que es posible una tasa de lectura de 25 MB/seg. Las tarjetas nuevas utilizan el bus UHS-1 que alcanza los 104MB/seg.

De esta manera las tarjetas que no entren dentro de lo anteriormente comentado no funcionaran en Arduino y por tanto estamos limitados a las tarjetas SD por sus características, las SDHC y SDXC no son soportadas.

2.4 Introducción a Arduino y los módulos seleccionados

Se va a proceder a presentar brevemente los dispositivos y módulos seleccionados y su funcionamiento típico.

2.4.1 Arduino Due

Arduino Due es una variante de Arduino con un procesador ARM CortexM3 de 32 bits, programables desde la interfaz de Arduino, con una frecuencia de funcionamiento de 84MHz, a 3,3V de tensión de funcionamiento.

Dispone de 12 entradas analógico-digital, 2 salidas digital-analógico, 54 entras/salidas digitales de los cuales todos se pueden configurar como interrupción hardware con un total de 6 interrupciones (INT0, INT1, INT2, INT3, INT4, INT5). Posee una conexión USB-OTG, conector SPI, 2 puertos para conexión I2C, 4 puertos Serie, además del puerto USB Nativo, ISCP, un encabezado Jtag y Debug. Dos botones, uno es el reset y otro de borrado EEPROM.

Este modelo es perfecto para dispositivos de laboratorio debido a su alta sensibilidad en sus pines analógicos y alta frecuencia de funcionamiento.

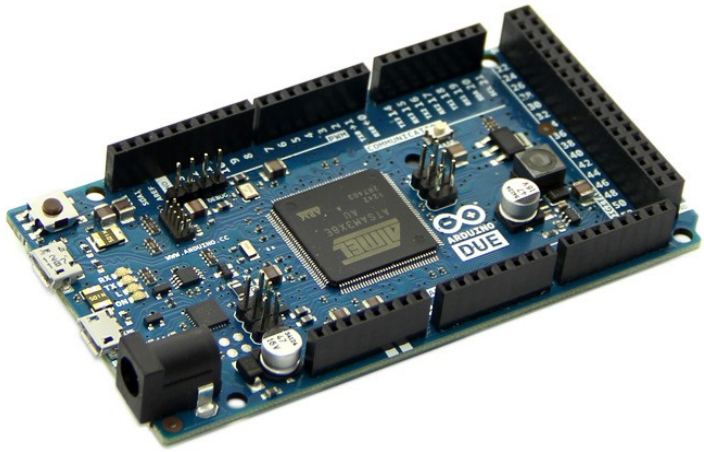

Figura 2.4: Arduino Due

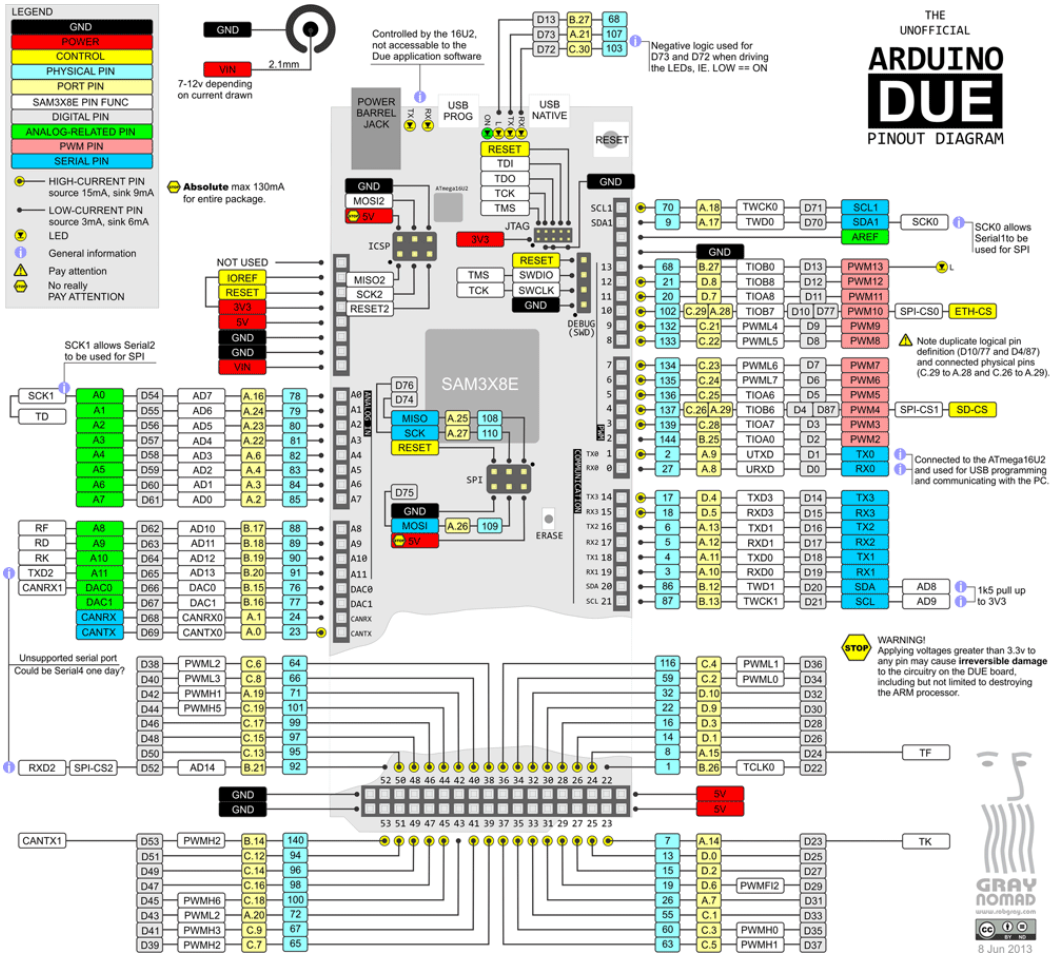

Figura 2.5: Mapa de los pines de la placa Arduino Due

2.4.2 Display LCD

El display monocromado retroiluminado con una matriz de 16 columnas y 2 filas, es económico y suficiente para mostrar la información de las muestras recogidas por el sensor.

Funciona a 5V y dispone de 16 pines para su funcionamiento.

Pin	Symbol	Level	Function
1	VSS	0V	Power GND
2	VDD	+5V	Power supply for logic
3	V0	—	Operation voltage for LCD
4	RS	H/L	H:Data L:Instruction code
5	R/W	H/L	H:Read L:Write
6	E	H/L	Enable signal
7	DB0	H/L	Data bus line
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	BLA	+5V	Power supply for LED backlight
16	BLK	0V	

Figura 2.6: Descripción de los pines del display LCD

Se va a conectar mediante el modo 4 pines, donde los pines 11, 12, 13 y 14 van a transmitir datos. Esta forma de funcionamiento es muy rápida y nos permite poder tener una alta tasa de refresco del display. Se podrían usar otros buses, como por ejemplo con un bus I2C con el que tendríamos una sensación de retardo para tan solo dos conexiones menos, mientras que con el modo 8 pines, duplicaríamos el cableado para la transmisión pero no mejoraríamos las prestaciones ya que no necesitamos tanto ni tampoco disponemos de un display muy grande.

Además hacemos uso de la librería LiquidCrystal para realizar una programación de alto nivel con el fin de controlar el display, sin necesidad de programar a bajo nivel.

Figura 2.7: Display LCD

2.4.3 Reloj RTC

Se necesita llevar un registro ordenado temporalmente del registro de muestras almacenado en la tarjeta microSD. Como el reloj interno de Arduino se reinicia cada vez que apagamos el dispositivo, necesitamos un reloj externo con batería propia. Para ello disponemos de los relojes RTC.

Echando una ojeada en las tiendas on-line de los productores oficiales de hardware compatible con la fundación Arduino, encontramos un módulo de Adaloger Feather RTC y microSD, integrados en la misma placa; nos proporciona una doble solución a nuestro proyecto.

El reloj RTC funciona a 3V y utiliza el bus I2C para comunicarse con la placa Arduino. Se utiliza la librería RTClib para su programación.

Figura 2.8: Adalloger Feathuring

2.4.4 Lector Targetas MicroSD

Como se ha comentado en el apartado anterior, el módulo lector de tarjetas microSD seleccionado es el Adaloger Feather , utiliza el bus SPI para conectarse a la placa Arduino, pines MISO, MOSI, SCK se conectan directamente al puerto SPI de la Arduino Due, mientras que el ChipSelect (CS) se conecta a los pines digitales.

Se usa la librería SD para su programación a alto nivel.

2.4.5 Sensor Hall

Definitivamente se escoge el sensor Hall A1318 lineal de 1.3 mV/Gauss con conexión One-Wire; de los cuales hay muchos fabricantes y se puede elegir cualquiera de ellos que cumpla esas condiciones y sea compatible con Arduino Due.

Figura 2.9: Sensor Hall A1318

2.4.6 PCD Botonera interfaz usuario

En este último apartado, desarrollaremos el proceso de elaboración de la botonera de interfaz de usuario. Debido a que en el mercado existe una variedad limitada de botoneras compatibles con Arduino Due, y ninguna de ellas se adecua a las necesidades del proyecto. Por eso la mejor opción es elaborar nosotros mismos la botonera adaptada a nuestras necesidades.

Para ello con unos botones para circuitos impresos, unas resistencias de 10k Ohmios, un condensador de 6 uF, una PCB perforadora en matriz, para que con un poco de estaño y un soldador poder disponer de nuestra interfaz de usuario para utilizar el Magnetómetro.

Con todo esto conseguimos solucionarlo de una forma muy económica y sencilla de reparar. No dispondrá de unos acabados perfectos para su venta, pero si suficientes para fomentar el aprendizaje y auto-construcción de prototipos.

Además configuraremos los pines correspondientes para que detecten los flancos de bajada de los botones y activen cada uno su interrupción correspondiente. Además de la incorporación de un condensador para anular el efecto rebote o *debounce* de los pulsadores, también se programara un sistema de control mediante la medición temporal entre flancos de activación de las interrupciones.

Capitulo 3 : Diseño

Durante este capitulo vamos a elaborar el diseño del dispositivo, para ello confeccionamos un flujo de los datos para orientarnos luego en la implementación del código. También seleccionaremos los pines a los que se conectará cada componente al Arduino Due, y por último diseñaremos dentro de la caja la colocación de los componentes y comentaremos la alimentación del dispositivo.

3.1 Diagramas de flujo

En este apartado se va a proceder a presentar y explicar los diagramas de flujo confeccionados para el diseño de un Magnetómetro.

3.1.1 Diagrama principal

Primero analizamos el funcionamiento de básico de la estructura del código de Arduino típica un método setup() de inicialización, donde configuramos los pines, interrupciones y los módulos LCD, microSD, sensor Hall y el reloj RTC.

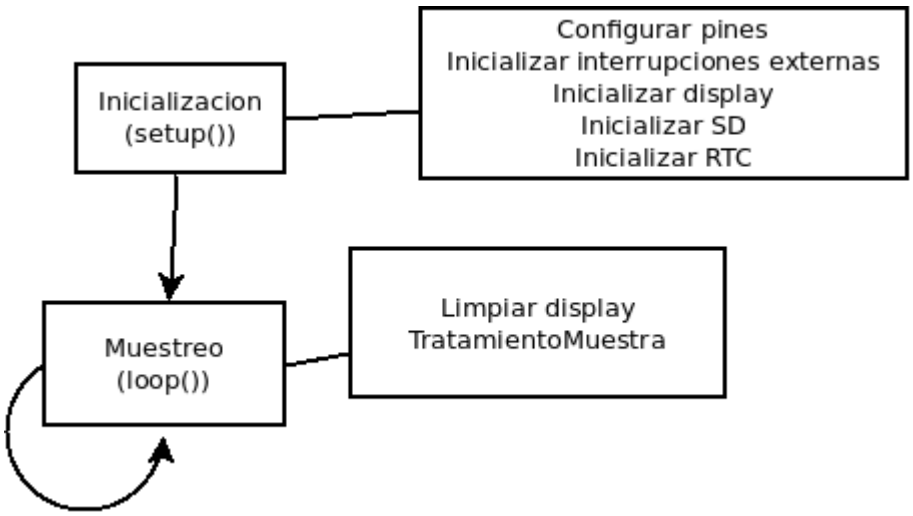

Figura 3.1: Diagrama de flujo principal

Seguido del método principal que siempre se repite, loop(). Donde se realiza una limpieza del display lcd y se invoca a la función tratamientoMuestra, que gestiona toda la toma de información, impresión y almacenado de cada muestra tomada.

3.1.2 Función tratamientoMuestra

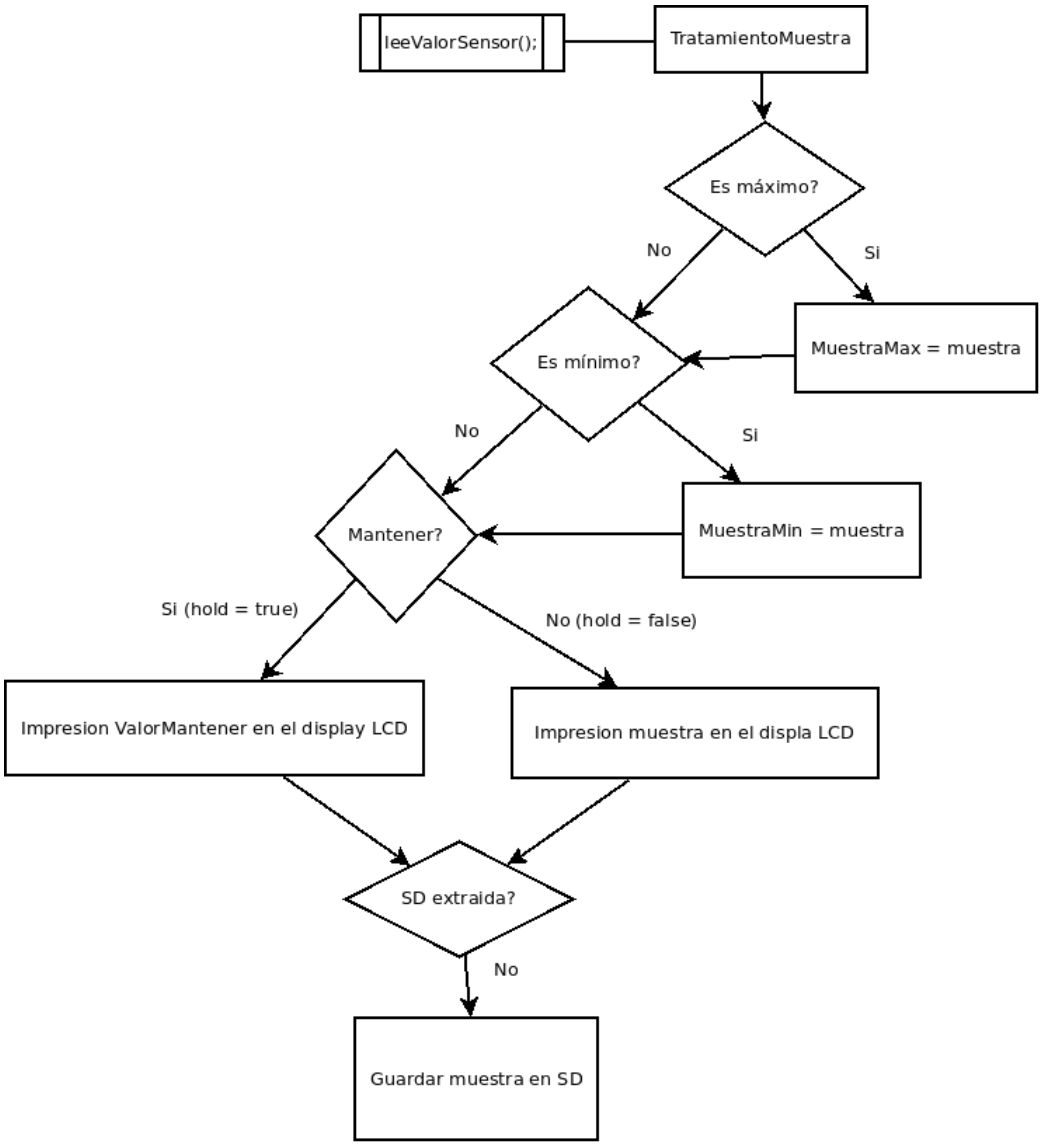

Figura 3.2: Diagrama de flujo del tratamiento de las muestras

Como se puede observar en la Figura 3.2 primero se lee una muestra, luego se examina por si es máximo o mínimo. Seguidamente dependiendo de la interacción del usuario, se realizara una gestión distinta, si el usuario deseaba mantener la imagen congelada en el display LCD se invocara la función impresionMuestra pasándole como parámetro el valor a mantener almacenado en la variable correspondiente; por otra parte se invocaría la misma función pero como parámetro la muestra tomada en esa misma iteración del programa.

Y por último si el usuario no ha extraído la tarjeta microSD se procederá a guardar en ella la ultima muestra junto a la hora y día y el máximo y mínimo en ese momento.

Hay que observar que se utiliza un tipo de datos llamado lecturaSensor, el cual es una estructura conformada por 7 elementos significativos de la representación de información perteneciente a una lectura del sensor Hall.

3.1.3 Función impresionMuestra

Como ya se ha comentado anteriormente, para la gestión de la impresión por el display LCD, se invoca la función impresionMuestra(lecturaSensor). Su flujo de datos se puede observar en la Figura 3.3

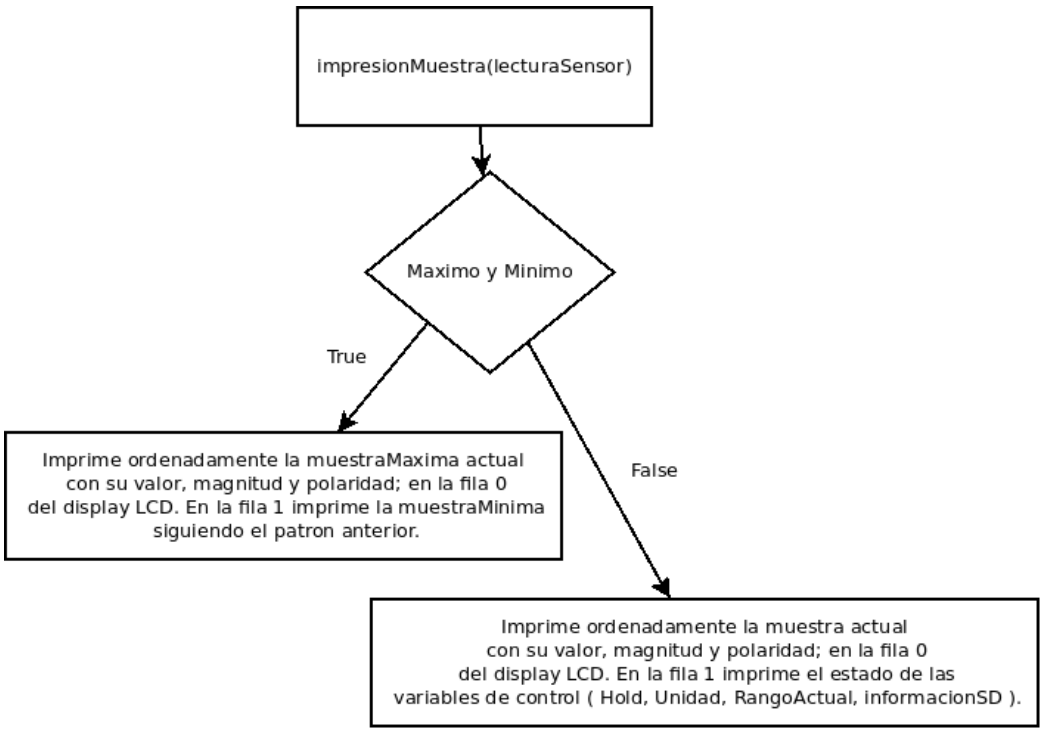

Figura 3.3: Diagrama de flujo de la impresión de las muestras por el display

Aquí, nuevamente, dependiendo de la interacción del usuario, se deberá actuar de manera distinta. Concretamente tenemos dos casos destacables que son, por un lado si el usuario desea ver el máximo y mínimo almacenados o si por el contrario desea observar el muestreo del sensor.

En cada uno de los casos, la gestión es similar, se organiza la información en la matriz de 16x2 del display LCD. Inicialmente, se imprimirá el muestreo del sensor, de manera que en la fila 0 se imprimirá la información relevante a la muestra. Y en la fila 1 se mostrara el estado de interacción del usuario con el dispositivo.

En caso de que el usuario solicite el máximo y el mínimo almacenados, se organizara de manera que cada fila muestre la información relevante de cada uno de ellos.

3.1.4 Función guardarMuestra

En la Figura 3.4 queda reflejada la gestión de almacenamiento de la información muestreada en cada iteración del loop principal. De esta manera podremos tener un registro ordenado del uso del Magnetómetro.

Para ello, se utiliza la función guardarMuestra, que es invocada pasándole una estructura lecturaSensor como parámetro. Esta función abre el archivo datos.txt, si no ha habido error, prepara en una variable la información ordenadamente para guardarla en la microSD.

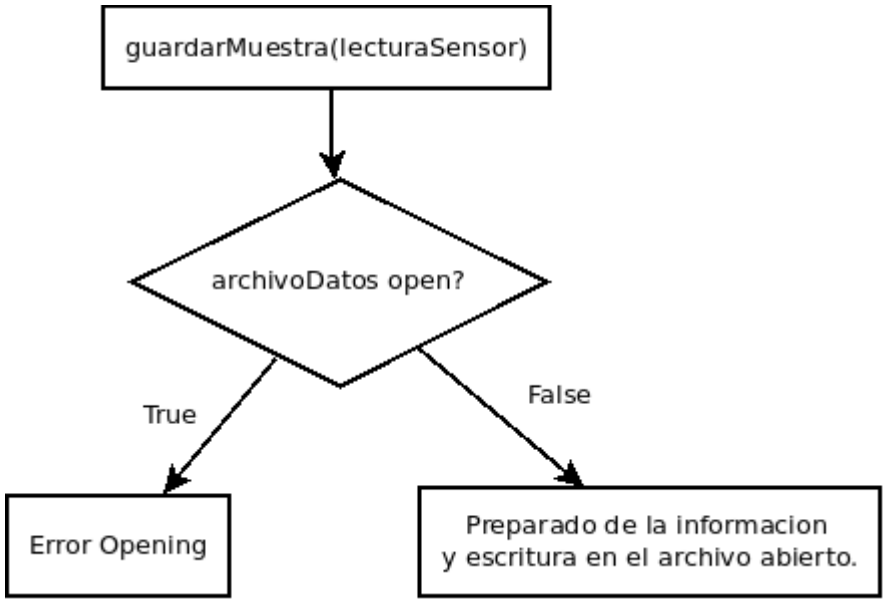

Figura 3.4: Diagrama de flujo de guardado de las muestras

Además se ha preparado un botón de la interfaz, para extraer con seguridad la tarjeta microSD, de manera que cuando se solicita su extracción deja de invocar a la función guardarMuestra.

De esta manera garantizamos que al extraer la tarjeta no lo haremos en medio de un almacenado de muestra, ja que eso podría causar daños a la información del archivo de registro.

3.1.5 Función leeValorSensor

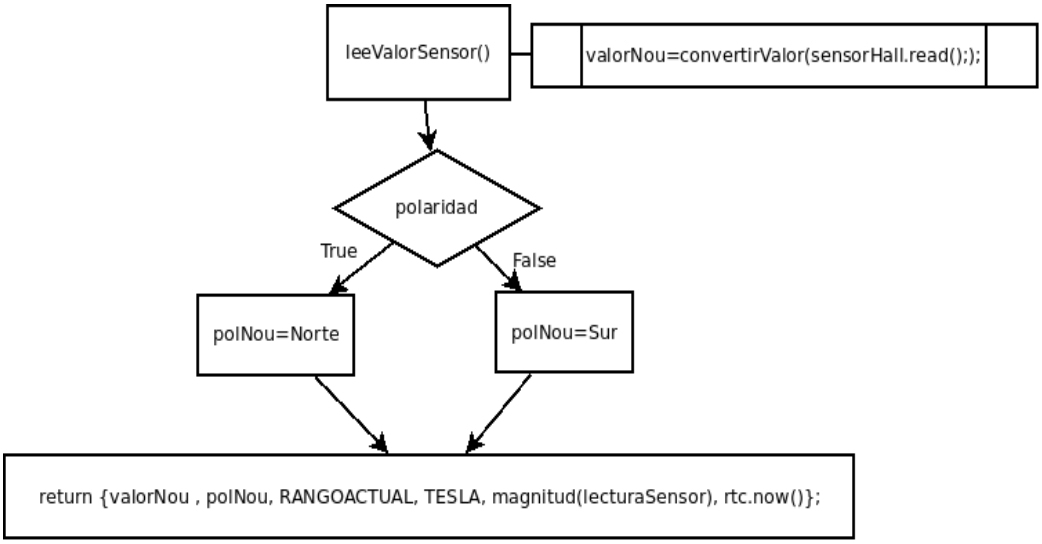

Figura 3.5: Diagrama de flujo de la lectura de una muestra

Hemos planteado la impresión y el almacenado de la información, ahora nos toca un punto por el cual no podrían funcionar los anteriores, el tratamiento de la señal obtenida del sensor Hall.

En este proceso están involucradas 3 funciones. La primera de ellas leeValorSensor, no necesita parámetros de entrada y devuelve una instancia del tipo lecturaSensor, recopila y organiza los 7 datos necesarios para crear la nueva instancia. Para ello necesita la ayuda de las otras 2 funciones, convertirValor y magnitud, que se explicaran a continuación.

3.1.6 Función convertirValor

Esta función necesita un entero como parámetro de entrada, que es la lectura del pin analógico al que esta conectado el sensor Hall. Además devuelve un double resultado de aplicar una ecuación de acondicionamiento de la señal proporcionada.

Como se observa en Figura 3.7, diferenciamos entre las muestras tomadas en Tesla o en Gauss, permitiendo poder tener mayor versatilidad de magnitudes, consiguiendo así T, mT, uT, kG, G y mG.

La ecuación de acondicionamiento consiste en acotar el rango de medida, sabiendo que tenemos 1.3 mV por cada 1G de variación en el sensor, que Arduino Due dispone de hasta 3300 mV, y que el sensor tiene una tensión de 1,62 V cuando el campo magnético captado es nulo. Operamos para calcular el rango máximo de medida del sensor A1318:

$$3300\,mV \times \frac{1\,G}{1{,}3\,mV} \times \frac{1\,mT}{10\,G} = \frac{3300\,mT}{1{,}3 \cdot 10} = 253{,}846\,mT$$

Figura 3.6.1: Máximo rango de medida

Sabiendo que el valor 0 del campo magnético corresponde a una tensión de 1,62 V, sabemos que los valores negativos del campo magnético oscilaran entre 0 V a 1,62V y los valores positivos de 1,62 V a 3,3 V. Habiendo calculado el valor máximo que podemos captar con el sensor, si este resultado le una ecuación de adaptación para el rango de medida de [-126,923 mT, 126,923 mT]. Observamos que el valor captado en la entrada analógica de Arduino Due varia entre [0, 1024], este valor hay que adaptarlo al rango de [-126,923 mT, 126,923 mT]. Para ello operamos y al generalizar obtenemos:

$$valor \times \left(\frac{\left(\frac{Vmax \times 1\,G \times 1\,mT}{Vsensor \times 10\,G} \right)}{resolucion} \right) - \frac{Vmax \times 1G \times 1\,mT}{Vsensor \times 10\,G \times 2}$$

Figura 3.6.2: Ecuación de adecuación de la señal

Donde **valor** es la lectura de la entrada analógica de la placa Arduino Due, **Vmax** es la tensión máxima de la placa Arduino Due (3300 mV), **Vsensor** es la tensión por cada Gauss dada de cada sensor hall; en nuestro caso el sensor hall A1318 tiene 1,3 mV/G. **Resolución** es el total de bits utilizados por el sensor y la placa Arduino Due para la representación de la señal analógica transmitida.

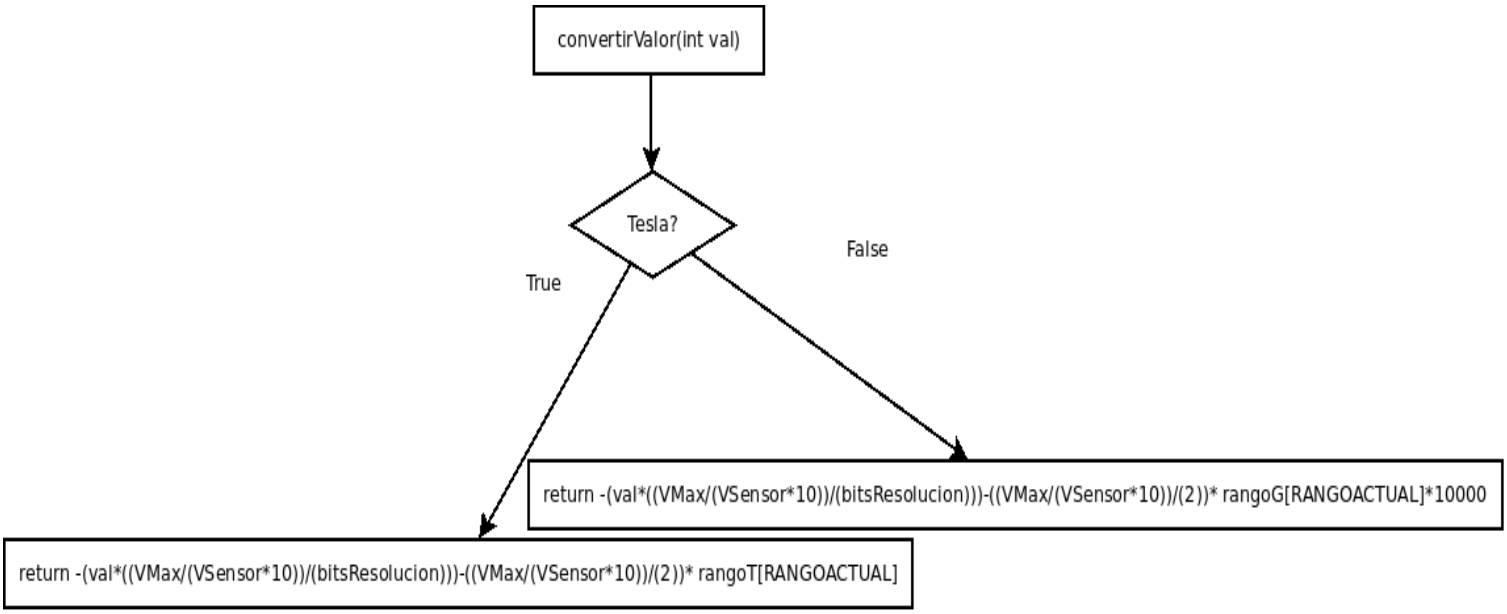

Figura 3.7: Diagrama de flujo de la aplicación de la ecuación de adecuación

En la figura 3.7, vemos el procedimiento seguido para convertir el valor leído al rango de medida adecuado. Además teniendo en cuenta en la unidad de medida en la cual se esta solicitando las muestras por parte del usuario.

3.1.7 Función magnitud

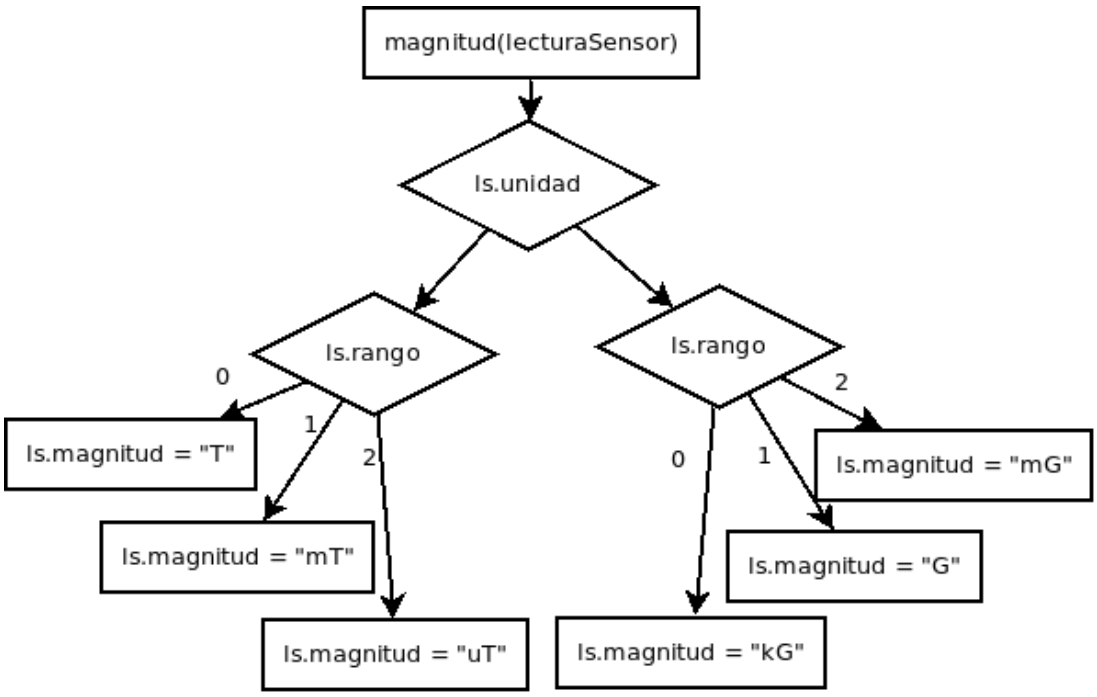

Figura 3.8: Diagrama de flujo de la representación de la magnitud medida.

Como podemos observar en la Figura 3.8, la función magnitud analiza el estado de la variable unidad de la lecturaSensor recibida por parámetro, esta dependiendo de su valor nos indica si esta medida en Tesla o en Gauss, luego comprueba el rango de la lecturaSensor para determinar el símbolo final que se asignara a la propia lecturaSensor en la variable magnitud.

3.2 Conexionado de los componentes

En este apartado plantaremos la interconexión de componentes y definiremos los pines de la placa de Arduino Due que tendrán los correspondientes buses de comunicación con los periféricos. Y al mismo tiempo definir en cada periférico en que pines se conecta a la placa.

Primero observemos la Figura 3.9, donde queda reflejado la interconexión de los componentes.

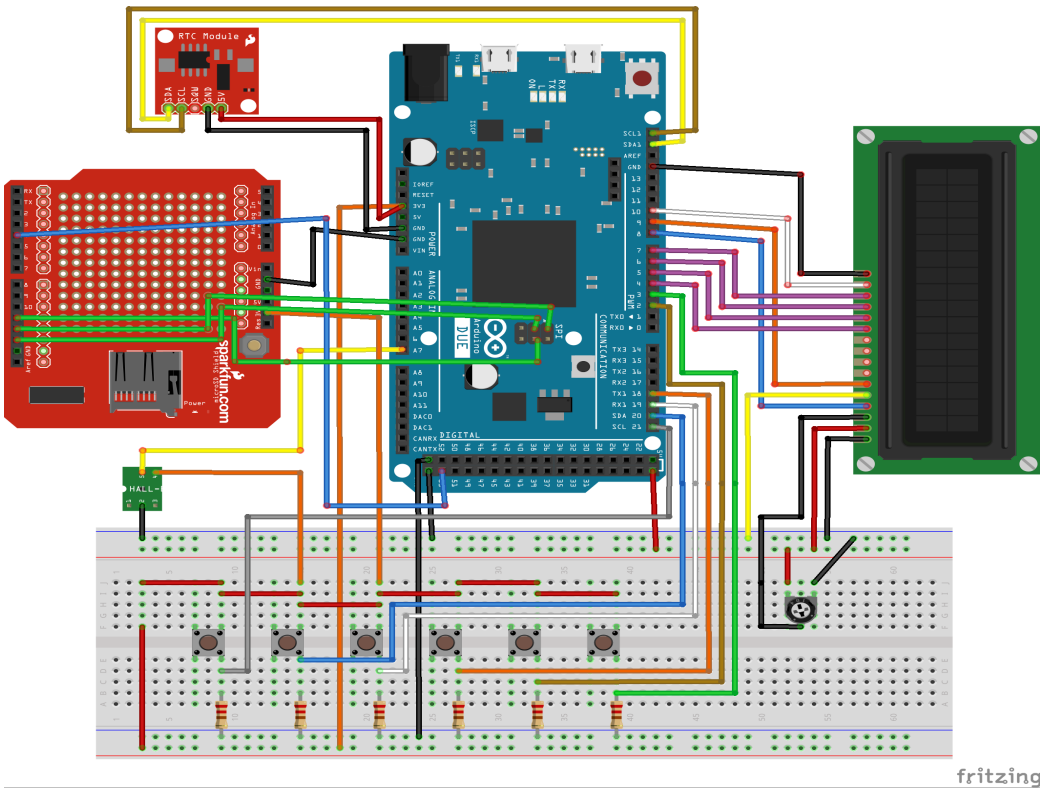

Figura 3.9: Diagrama de conexionado del dispositivo

Algunos de los módulos representados no son los mismos que el montaje final, pero cumplen sus misma características de funcionamiento e interconexión.

- **Display LCD 4 pin**

  Utiliza los pines 8(pin 4 del display),9(pin 6 del display), 4-7(pines 11-13 del display); de la placa de Arduino Due.

  Mientras que además tiene un potenciómetro de 50k Ohmios conectado al pin 3 del display, el resto de conexiones se realizan según la Figura 2.4.2.

- **Reloj RTC I2C**

  Al utilizar el bus I2C, utiliza un pin para reloj(SCL) y otro para datos(SDA). Se ha decidido utilizar los puertos de la placa Arduino dedicados al segundo bus I2C implementado, SCL1 y SDA1; el resto de conexiones son simplemente la alimentación del modulo como queda reflejado en la Figura 2.4.

- **Lector MicroSD SPI**

  El bus más eficiente para comunicarse con un modulo microSD que implementa Arduino, es el bus SPI, que utiliza 4 pines, CS (chipSelect), SCK (reloj), MISO (Master-IN, Slave-OUT), MOSI (Master-OUT, Master-IN), que dispone de pines propios en el centro de la placa. Por ello hace que Arduino solo sea capaz de leer tarjetas sd1 y sd2, ya que las sdh necesitan un puerto de mayor velocidad llamado USH.

- **Sensor Hall One-Wire**

  Se ha decidido utilizar un sensor con bus One-Wire, porque al ser un sensor lineal, su simplicidad permite que con el bus One-Wire tengamos junto con la alta frecuencia de procesamiento que nos ofrece Arduino Due, obtenemos una alta tasa de muestreo al que solo hay que aplicar una sencilla transformación que no ralentiza al sistema.

  Además de que reducimos el numero de cables y pines en uso de la placa Arduino Due pudiendo aceptar ampliaciones en un futuro.

  Al ser una señal analógica se conectara al puerto analógico-digital 0 de la placa Arduino Due.

- **Botonera interfaz usuario**

  Como ya se ha comentado anteriormente en el Capitulo 2.4.6, la botonera consistirá en un circuito eléctrico de una serie de pulsadores en pull-down, donde cada uno estará conectado a un pin de la placa Arduino donde detectara la interacción del usuario con la interfaz y disparar las interrupciones correspondientes.

  En un total de 6 pulsadores, accionando las interrupciones de HOLD(mantener) pin 20, LIGHT(iluminación) para la ArduinoMega o CALIBRA(calibrado) para la Arduino Due; pin 19, UNITS(units) pin 18, RANGE(rango) pin 3, MAX/MIN(máximo/mínimo) pin 21, EXTRACT(extracción segura microSD) pin 2. En este dispositivo vamos a encontrar un efecto físico en los pulsadores denominado *debounce o retardo*, consiste en la estabilización de la señal al cambiar de flanco, la cual produce unos instantes en que la señal oscila entre 0

y 3,3V, pudiendo activar nuevas interrupciones sin que sean realmente solicitadas por el usuario.

3.3 Revestimiento y alimentación

En este apartado vamos a centrarnos en la caja para el montaje definitivo, y como alimentaremos el dispositivo.

A lo que la caja respecta hemos decidido construir nosotros mismo la caja donde montar el dispositivo. Reutilizaremos algún tupper-ware o caja de plástico, que vallamos a desechar, suficientemente grande para abarcar nuestros dispositivos y cableado. De esta manera fomentamos la creación de prototipos caseros y una toma de contacto con la reutilización de los desechos plásticos en nuestros proyectos, consiguiendo darle uso a muchos desechos que se generan en el día a día en los hogares y que acaban en vertederos, así conseguimos alargar la vida útil de ciertos desechos y incentivando el reciclaje y la reutilización.

Además necesitamos, tornillos, porta-tornillos o fijaciones; para los tornillos podemos valernos de hardware en desuso, tal como lectores de cd, disqueteras, o discos duros rotos. Para la fijación de los tornillos en la caja, podemos utilizar tacos de plástico para fijar modificaciones con calor conseguimos fácilmente unos porta-tornillos robustos.

El proceso de montaje se describirá detalladamente más adelante en Capitulo 4., pero brevemente consistirá en seleccionar la ubicación de los componentes, realizar los cortes y perforaciones necesarios para dejar el sitio de los componentes, colocar unos porta-tornillos donde fijar los componentes a la caja. Todo esto realizado con un soldador de estaño, regla, lápiz y una pistola de termo-silicona.

Respecto a la alimentación del dispositivo, dispondremos de dos opciones a usar, dependiendo de las necesidades del momento. Se podrá conectar con un transformador de 220V a 5 a 7 V, o por contra usar un banco de energía y conectarse mediante el USB.

Figura 3.10: Componentes del Teslametro con Arduino

Capitulo 4 : Implementación

En este cuarto capitulo vamos a implementar los diseños realizados en el capitulo anterior. Para ello separamos la estructura de datos utilizada para datos del sensor, las librerías utilizadas, las funciones de flujo de datos y el montaje del dispositivo en la caja.

4.1 Estructura de datos lecturaSensor

Debido a la necesidad de transferir fácilmente la información de las lecturas del sensor entre las distintas funciones, se ha decidido agruparla toda ella en una estructura.

```
struct lecturaSensor {
        int valorOriginal;
        double valor;
        String polaridad;
        int rango;
        boolean unidad;
        String magnitud;
        DateTime fechaYHora;
};
```

Figura 4.1: Estructura de la lectura de muestras

Esta estructura consiste en el almacenamiento de información relativa a las lecturas tomadas del sensor Hall.

Tales como, el *valorOriginal*, un entero usado para comprobar si son máximos o mínimos; el *valor* convertido a Tesla o Gaus con un double, la *polaridad* del campo magnético en un String, la *unidad* utilizada mediante un boolean T *true* o G *false*, el *rango* indicado mediante un entero que varia entre T un 0, mT un 1, uT un 2 y igual para KG, G, mG; el String de la *magnitud* utilizada en la medición y por último la *fecha y hora* con un DateTime.

4.2 Librerías utilizadas

Durante el siguiente apartado vamos a presentar a las librerías empleadas en el desarrollo del software.

4.2.1 LiquidCrytal

Esta librería permite a la placa Arduino Due controlo el diplay LCD. La librería trabaja en el modo 4 o 8 bit.

Para trabajar con ella se crea una variable global del tipo LiquidCrystal e invocando al constructor de la clase que nos convenga. Existen 4 que son :

- LiquidCrystal(rs, enable, d4, d5, d6, d7)
- LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
- LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
- LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)

En ellos podemos observar las variables que guardan relación con los pines del display planteados en el capitulo 2.4.2. El primero de ellos activa el display en modo 4 bit y además solo lectura, el segundo es con lectura y escritura. Del mismo modo para el tercero y cuarto, pero estos para el modo 8 bit.

Además dispone de otras funciones, como begin(), clear(), setCursor(), print(), cursor(), leftToRight(), entre otros. Todo esto esta disponible en la pagina web de Arduino.

4.2.2 SPI y SD

La librería SPI, consiste en una clase de control del bus SPI (Serial Peripheral Interface) el cual sigue el protocolo de conexión serie síncrona. Este bus es utilizado por la tarjeta SD, para comunicarse con la placa Arduino Due. La librería SD habilita el bus utilizando la librería SPI, y invocando las funciones de la librería SD controlamos el modulo de la tarjeta micro SD. de esta manera lee y escribimos en los archivos almacenados en la micro SD.

La librería SD dispone de varias clases de las cuales obtiene distinta información y funcionamiento. Esas clases son File, SDClass, Sd2Card y SdVolumen, entre otras. La clase File es usada para lee y escribir archivos almacenados; con Sd2Card y SdVolumen obtenemos información de la tarjeta insertada en el modulo. Y con la SDClass obtenemos una instancia llamada SD, con la que invocar las funciones que nos permiten abrir los archivos que almacenaremos en la variables del tipo File.

Todo esto puede consultarse en la pagina de Arduino.

4.2.3 Wire y RtcLib

Estas librerías son utilizadas para el funcionamiento del dispositivo del reloj RTC, en concreto la librería Wire es utilizada por la librería RTClib para usar el bus I2C o TWI, al igual como pasa con la librería SPI y SD.

El bus en este caso esta formado por dos conexiones, una de datos(SDA) y otra de sincronización (SCL). Es un protocolo asíncrono, SCL indica cuando leer los datos. Cada dispositivo conectado al bus tiene una dirección de 7 bits con lo que se pueden conectar hasta 128 dispositivos en paralelo (2^7). Entre ellos debe haber uno o más maestros que controlan el reloj, pudiendo estar solo uno activo a la vez. Y el resto de dispositivos son esclavos.

La librería RTClib, es una librería de Adafruit, disponible en el GitHub de Adafruit. Esta incluye las clases DateTime, TimeSpan,RTC_DS1307, RTC_DS3231, RTC_PCF8523, RTC_Millis. Para su uso, creamos una variable global de la clase RTC_Millis; de la cual se tomara la fecha y la hora almacenadas en el modulo RTC. Se guardara en una variable de la clase DateTime. La librería se encarga de activar el bus I2C, establecer el maestro y los esclavos, en nuestro caso solo al modulo RTC; y realizar las comunicaciones a través de él.

4.3 Funciones

A continuación presentaremos la implementación de las funciones pertenecientes a los diagramas de flujo de datos realizados previamente. Están dividas dentro de donde son utilizadas, es decir, los programas en Arduino se componen del *setup()* y del *loop()*, y pueden haber más funciones pero solo van a poder ser invocadas dentro de los dos anteriores .

Por tanto están divididas entre las que son invocadas en el *setup()* y las que son invocadas en el *loop()*.

4.3.1 setup()

En el transcurso del dispositivo, lo primero que realizamos es la inicialización y el establecimiento de los pines y las interrupciones.

Primero inicializamos los pines mediante ***pinMode(nºPin, Mode)***, donde indicamos el numero del pin y si actuara como INPUT o OUTPUT. En total inicializamos 7 pines de esta manera; uno para cada botón y uno para la conexión de alimentación de la iluminación de la pantalla LCD.

Posteriormente, configuramos las interrupciones mediante:

attachInterrupt(digitalPinToInterrupt(nºPin), función de tratamiento, flanco de detección);

Donde indicamos el pin mediante una conversión a pines de interrupción con ***digitalPinToInterrupt()*** . Luego el nombre de la función de tratamiento donde ejecutara las acciones previstas para esa interrupción y el flanco de detección de la interrupción.

Luego se invocan 3 funciones escritas fuera del bloque del *setup()*, para juntar en cada una los procesos de inicialización de los distintos componentes.

4.3.1.1 inicializaPantalla()

El display LCD, es inicializado mediante la instrucción ***lcd.begin(16, 2)***;. donde lcd es una instancia de la clase LiquidCrystal declarada anteriormente a la invocación de la instrucción de inicialización; ***LiquidCrystal lcd(8, 9, 4, 5, 6, 7);*** indicando los pines utilizados para su conexionado, explicados anteriormente en el capitulo 3.2 Display LCD 4 pin. Por último se invoca ***lcd.clear();*** para limpiar alguna posible impresión anterior.

4.3.1.2 inicializaSD()

Mientras que para el lector de tarjetas microSD, se inicializa el modulo mediante *SD.begin(chipSelect)*, si esto falla se guarda el error en la variable ***iniSd***, dejándola en *false*, por el contrario si se inicializa correctamente se pone a *true* indicando que hay una tarjeta insertada en el modulo.

Además inicializamos otro modo de la tarjeta, ***card.init(SPI_HALF_SPEED, CS)*** activando una variable tarjeta del tipo ***Sd2Card***, de la cual extraemos información relevante como su tipo de datos o el tipo de tarjeta SD.

Por último, comprobamos de la existencia del archivo de registro, y si no esta lo crea mediante la función creaFichero().

4.3.1.3 creaFichero()

La función **creaFichero()**, es invocada por las funciones **inicializaSD()** y **borradoInformacion()**. Su cometido es de crear el archivo de registro de datos y escribir la fecha y hora de reinicio del archivo.

Para ello instancia una variable del tipo ***DateTime*** con valor **rtc.now()**, de manera que dispone de la fecha y hora de ese momento. Luego abre un archivo con **SD.open(nombreFichero, FILE_WRITE)** con nombre el String almacenado en la variables nombreFichero y en modo escritura; escribe *"Lectura del dd/mm/yyyy a las hh:mm:ss"* en el fichero abierto y por último cierra el fichero.

4.3.2 loop()

Llegados a este punto, nos encontramos delante de la actividad principal del dispositivo, el corazón y el cerebro que dan vida al cuerpo, metáfora para recalcar la importancia de las funciones siguientes.

Con un total de 14 funciones, se ha intentado separar ordenadamente las partes de código de cada componente que integran el dispositivo. Con ello conseguimos un código limpio y fácil de modificar en el caso de necesitar algún cambio de hardware o por posibles ampliaciones de él.

4.3.2.1 tratamientoMuestra()

En esta primera función invocada desde el loop principal, se encarga de invocar ordenadamente al resto.

Primero invoca *leeValorSensor()*, almacenando en la variable global ***muestra*** la estructura *lecturaSensor* devuelta como resultado de la invocación. Luego analiza la muestra comparándola con los valores originales del sensor a las muestras máxima y mínima. Si cumplen las condiciones se almacena la muestra actual en la muestra máxima o mínima según corresponda.

Seguidamente comprueba el estado de la variable ***HOLD***, por si el usuario ha solicitado mantener la imagen del display congelada, en ese caso ***HOLD*** valdrá *true* se invoca la función *impresionMuestra()*, pasándole como parámetro la variable global ***valorMantener.*** Por contra ***HOLD*** valdrá *false* y se seguirá invocando la función *impresionMuestra()* con parámetro la muestra tomada en esa iteración.

Por último comprueba si no se ha extraído la tarjeta microSD, mediante el estado de la variable ***extraida***, si su valor es false, la tarjeta permanece en funcionamiento, al activarse la interrupción *extraccionConSeguridad(), a petición del usuario;* modificará el valor de la variable a *true.* De esta manera se deja de invocar a la función *guardarMuestra().*

4.3.2.2 impresionMuestra()

La función ***impresionMuestra(lecturaSensor)*** esta declarada como *void,* recibe como parámetro una struc del tipo *lecturaSensor*.

Primero comprueba el estado del dispositivo, si el usuario ha solicitado mostrar el máximo y el mínimo, utilizando las funciones disponibles de la clase LiquidCrystal, librería la cual ha sido explicada en el capitulo 4.2.1; realiza una impresión por el display LCD, de las variables ***muestraMax*** y ***muestraMin***; mostrando sus *polaridades*, *valor* y la *unidad* de medición correspondiente con el valor mostrado.

Por contra, si no se solicitan el máximo y el mínimo; el dispositivo imprime las lecturas tomadas una tras otra. De manera que en la fila 0 del display se imprime la polaridad, el valor y la unidad del muestreo; y en la fila 1 del display se imprime el estado del dispositivo.

De manera que si ***HOLD*** es *true* se muestra una ***"H"***, se muestra la *unidad* utilizada, el *rango* aplicado a la unidad, y por ultimo información correspondiente con la tarjeta SD insertada, si hay tarjeta indica su *tipo* (SD1, SD2, SDH), y seguido el *formato* de archivos de la tarjeta, si existe un problema con el *formato* de archivos imprime ***"Fat E"*** o si no se puede escribir las muestras imprime ***"Error W"***; por contra en caso de no haber insertado tarjeta, imprime ***"Fallo SD"***.

4.3.2.3 guardarMuestra()

Para guardar el valor de las distintas variables con información relevante, se invoca la función ***guardarMuestra(lecturaSensor)*** declarada *void*, necesita como parámetro una estructura del tipo *lecturaSensor* almacenada en la variable *ls.*.

Primero mediante la invocación de ***SD.open(nombreFichero, FILE_WRITE);*** carga el archivo de registro en la variable *archivoDatos* y si se ha podido abrir, se procede a escribir una linea mediante la instrucción ***archivoDatos.println();*** pasándole como parámetro, una concatenación ordenada de la información recopilada; por otra parte si no se ha podido abrir se modifica el valor de la variable ***errorWrite*** a *true*.

En la concatenación primero se escribe la muestra actual, ***ls.polaridad***, ***ls.valor*** y ***ls.magnitud***. Luego realiza el mismo proceso para la ***muestaMax*** y la ***muestraMin***. Por último imprime la hora de la variable ***muestra*** mediante ***ls.fechaYHora***.

Luego de escribir la nueva linea en el archivo, cierre este mediante ***archivoDatos.close();*** y modifica el valor de la variable ***errorWrite*** a *false*.

4.3.2.4 leeValorSensor()

En esta función se realiza la lectura del sensor en un instante dado. Para ello no necesita parámetros de entrada, pero si devolver la estructura ***lecturaSensor*** creada en su invocación.

Para tomar la muestra, almacena en la variable ***valorOriginalNou*** del tipo *int* el valor de la entrada analógica mediante ***analogRead(SENSOR),*** además en la variable ***valorNou*** del tipo *double* almacena el valor devuelto por la invocación a la función ***convertirValor(valorOriginalNou)***.

También comprueba la polaridad de la señal captada mediante la comparación ***valorOriginalNou*** >= ***bitsResolucion***/2, donde ***bitsResolucion*** indica el numero total de bits medido por el convertidor analógico-digital de la entrada analógica de Arduino

Due. Si el valor es mayor o igual se asigna **"S"** a la variable ***polNou*** del tipo *String*, sino asigna **"N"** a la misma variable.

Crea una struc del tipo lecturaSensor llamada ***nova*** a la cual asigna todos los campos obtenidos. El ***valorOriginalNou***, el ***valorNou***, el ***polNou***, junto al estado de las variables globales ***RANGOACTUAL*** y ***TESLA***. Es asignado espacio nulo ("") en el *String* de la *magnitud*, y la *hora y fecha* del relog RTC mediante ***rtc.now()***.

Por último es invocada a la función ***magnitud(lecturaSensor)*** *pasándole* como parámetro la estructura creada anteriormente ***nova,*** completando la información correspondiente .

4.3.2.5 convertirValor(int)

En esta función ***convertirValor(int)*** es invocada desde la función ***leeValorSensor()*** pasándole como parámetro de entrada el entero resultante a la lectura de la entrada analógica A0.

La función ***convertirValor()*** se encarga de transformar el entero leído en la entrada analógica, a su valor correspondiente en T o G. Para ello aplica la ecuación de la figura 3.6.2. Esta ecuación esta explicada en el capitulo 3.2.1.

4.3.2.6 magnitud(lecturaSensor)

La función ***magnitud()***, recibe como parámetro de entrada una estructura del tipo ***lecturaSensor***, y retorna otra estructura del mismo tipo.

La tarea de esta función es asignar a la estructura de entrada el *String* correspondiente a su magnitud de medida. Es decir asigna la representación correspondiente de la unidad de medida en el rango de magnitud en el que la ***lecturaSensor*** ha sido creada. Varia entre T, mT, uT, kG, G o mG.

Una vez asignado el *String* correspondiente, devuelve la estructura ***lecturaSensor*** completa a la función ***leeValorSensor()***.

4.3.2.7 Tratamiento interrupciones

En este apartado vamos a presentar las distintas funciones de gestión de interrupción. Una por cada pulsador de la botonera, un total de 6 útiles. Aunque existen 7 creadas, en Arduino Due una de ellas, concretamente ***iluminacion()*** no es utilizada, por incompatibilidad de voltajes entre los pines digitales de la Arduino Due y el voltaje de alimentación de la iluminación del display LCD. Existe para poder usarla en la Arduino Mega con la incorporación de un pulsador más en la botonera.

Además hay que tener en cuenta que al tratarse de funciones de tratamiento de interrupciones, se ha pretendido hacerlas lo más simples posibles para conseguir la mayor eficiencia posible.

Además en todas ellas se ha insertado unas lineas de código para controlar temporalmente la activación de las interrupciones, mediante una estructura de control **if** donde compara los tiempos obtenidos mediante ***millis()***. De manera que si una interrupción es solicitada otra vez antes de un 1,5 ms se considera que es a causa del efecto rebote o *debounce*.

4.3.2.7.1 mantener()

La interrupción declarada sobre el pulsador conectado al pin instanciado con la variable global ***HOLD,*** activa a la función ***mantener(),*** encargada de gestionar a la variable ***HOLDS*** del tipo *boolean*.

En la variable ***HOLDS*** queda reflejada la interacción del usuario solicitando mantener congelada la muestra en ese instante dado. Todo el proceso de detección de la variable ***HOLDS*** se lleva a cabo en las función ***tratarMuestra()***. La gestión de la interrupción consiste solamente en modificar el valor de la variable ***HOLDS***.

Dicha variable cuando su valor es *false*, el usuario quiere ver por el display el seguimiento del sensor, además por tiene este valor al iniciarse el dispositivo. Por el contrario al solicitar congelar la imagen su valor es modificado a *true*.

4.3.2.7.2 unidades()

La función ***unidades()*** es invocada con la activación de la interrupción asociada al pin instanciado con la variable global ***UNITS.*** Esta función modifica el valor de la variable global ***TESLA,*** en la cual queda reflejado la solicitud del usuario a representar la medida del campo magnético en Tesla o en Gauss.

Cuando la variable ***TESLA*** esta con valor *true* significa que el usuario quiere representar la medición en Tesla, además al iniciarse el dispositivo se inicializa a *true* por defecto. Mientras que si la variable vale *false* el usuario quiere representar la información en Gauss.

La variable **TESLA** es usada en las funciones ***leeValorSensor()***, ***convertirValor()*** e ***impresionMuestra()***.

4.3.2.7.3 iluminacion()

Esta función tiene asociada la interrupción del pulsador conectado al pin instanciado con la variable ***BOTONLLUM***, al activarse esta interrupción la función ***iluminacion()*** modifica el valor de la variable global ***apagada***.

El valor por defecto al iniciarse el dispositivo es *true*, esto hace que la función ***iluminacion()*** ejecute la instrucción ***digitalWrite(LLUM, HIGH),*** haciendo que el pin 14 del display LCD reciba 5V y encienda la iluminación de la pantalla. Mientras que al modificar su valor a *false*, ejecuta ***digitalWrite(LLUM, LOW)***, apagando la iluminación.

4.3.2.7.4 rangos()

La interrupción activada por el pulsador conectado al pin instanciado por la variable ***RANGO***, invoca la función ***rangos()***, encargada de la modificación de la variable global ***RANGOACTUAL*** del tipo int, instanciada mediante constrain ***(RANGOACTUAL, 0, 2)*** de manera que solo puede tener tres valores el 0, 1 y 2.

En la función ***rangos()*** se incrementa su valor, teniendo en cuenta que al llegar a 2 modificamos el valor a 0.

La variable ***RANGOACTUAL*** modifica el comportamiento de las funciones ***impresionMuestra()***, ***leeValorSensor()*** y ***convertirValor()***.

4.3.2.7.5 maximoMinimo()

Otra función encargada de la gestión de interrupciones, es la función ***maximoMinimo()***, esta esta asociada a la interrupción activada por el pulsador conectado al pin instanciado por la variable global ***MAXMIN.***

La función ***maximoMinimo()***, al ser invocada modifica la variable global ***MAXIMOMINIMO*** del tipo *boolean*. Esta variable por defecto su valor es *false*, de esta manera en la función ***impresionMuestra()*** imprima el muestreo del sensor. Por el contrario, cuando su valor es *true* la función ***impresionMuestra()*** imprime el máximo y el mínimos almacenados hasta el momento.

4.3.2.7.6 extraerConSeguridad()

La interrupción para la extracción segura de la tarjeta SD, se gestiona mediante la rutina ***extraerConSeguridad()*** invocada por el pulsador conectado al pin instanciado con la variable global ***EXTRACT*** del tipo *boolean*.

Dicha función examina la variable global ***extraida*** del tipo *boolean*, si su valor es *true* modifica a la variable ***extraida*** a *false* y además a otra variable *boolean* llamada ***iniSd*** a *true*. Por otra parte al valer ***extraida*** *false*, esta función modifica estas dos mismas variables a su opuesto, es decir, ***extraida*** queda a *true* y ***iniSd*** queda a *false*.

La variable ***extraida*** afecta a la funciones ***impresionMuestra()*** y ***tratarMuestra()***. Mientras que la variable ***iniSd*** afecta a las funciones ***impresionMuestra()*** y ***inicializaSd()***.

4.3.2.7.7 borrado()

Por finalizar con las rutinas de gestión de interrupciones, tenemos la función **borrado()**, ella es invocada mediante la instanciación del pin ***BORRADO,*** con la intención de gestionar la renovación del archivo de registro de datos.

Para ello invoca la función **borradoInformacion()**. Además controla temporalmente las instancias a la gestión de interrupciones **borrado()**, eliminando llamadas involuntarias producidas por el efecto rebote del circuito eléctrico de la interfaz de usuario.

4.3.2.7.8 borradoInformacion()

La función **borradoInformacion()** es invocada por la rutina de interrupción **borrado()**. Se ocupa de eliminar el archivo de registro de datos de la tarjeta SD, con nombre el String almacenado en la variable ***nombreFichero***. Posteriormente invoca a la función **creaFichero()**.

4.4 Montaje del dispositivo en su caja

Durante este apartado, se va exponer paso a paso el montaje de los componentes en la caja para tener el dispositivo acabado.

4.4.1 Preparativos

Para el montaje necesitamos, deformar los cilindros de plástico de manera que dejemos el orificio al tamaño de nuestros tornillos reciclados.

Figura 4.2: Reciclado y deformación de cilindros de plástico para porta-tornillos

Luego atornillar los porta-tornillos que acabamos de modificar a los orificios correspondientes de los componentes y soldarlas con termo-silicona o con otro adhesivo a la caja en la ubicación decidida para ese componente.

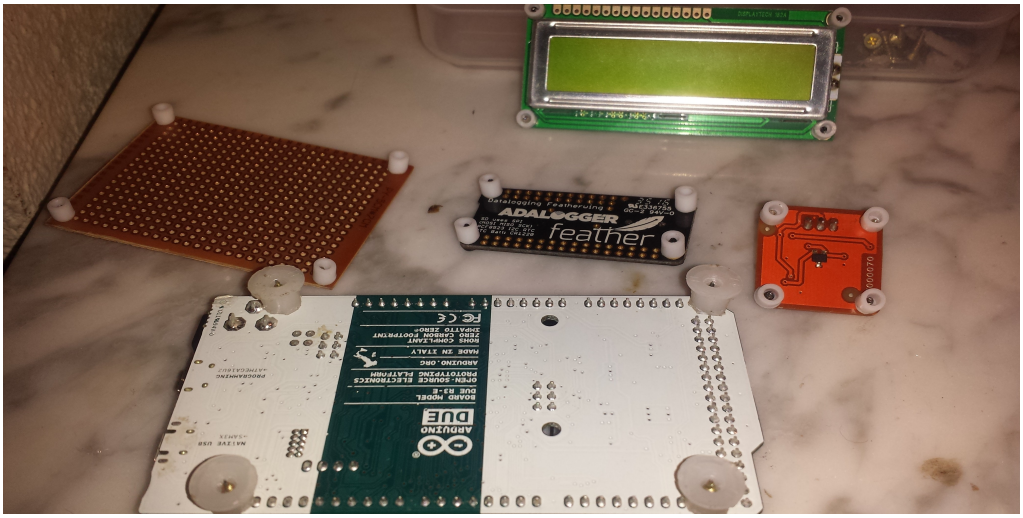

Figura 4.3: Componentes con tornillos y porta tornillos

Una vez fijadas las bases para el atornillado se puede montar el circuito de la figura 3.3, y verificar la buena colación de los componentes y el cableado de conexión. Una vez comprobado podemos desconectar el cableado y desatornillar los componentes de las fijaciones. Previamente marcaremos las zonas a perforar para dejar hueco para conectores, la pantalla y los botones.

4.4.1.1 Realizar las perforaciones

Para perforar las zonas de los conectores, botones y pantalla; utilizaremos un soldador de estaño o un pirograbador.

Quemaremos las marcas realizadas previamente y comprobaremos que acoplan correctamente. Seguidamente limamos y pulimos los contornos de las perforaciones.

4.4.1.2 Soldado de la botonera y de la ampliación de pines de alimentación

Siguiendo el diseño realizado de interconexión de la botonera, se realizara a la implementación sobre una placa de cobre perforada siguiendo una matriz, colocación de los botones, resistencias y el condensador para eliminar el efecto rebote.

Una vez soldado con estaño los componentes y los cables de conexionado con la Arduino Due se realiza un testeo de la tensión del circuito, comprobando su correcta implementación.

Además en una porción de una PCD se conectan conectores hembra para ampliar los pines de alimentación de 3.3V y GND. Y en el se coloca el potenciómetro que necesita el display LCD.

Y el sensor Hall A1318, con tres cables de conexionado, GND, VCC, DATA.

4.4.2 Colocación y conexionado de los componentes

Una vez preparados podemos colocar los componentes en su posición y procedemos a implementar la interconexión diseñada previamente. Para ello nos basamos en la figura 3.9.

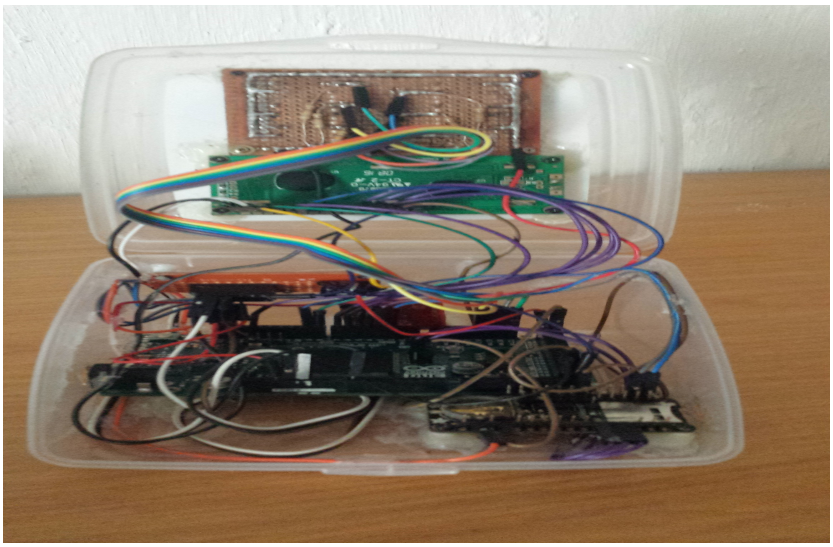

Figura 4.4: Caja acabada y circuito de conexionado montado

4.4.3 Cierre y comprobación del acabado

Teniendo ya todo bien conectado procedemos a cerrar la caja y comprobar si todo encaja bien. En caso de que no encaje bien, tendremos que volver a colocar los componentes como se comenta en el capitulo 4.4.1, en ocasiones es recomendable el uso de pines macho curvado para el uso en lugares con poco espacio.

Cuando este todo bien colocado y ninguna conexión se fuerce, el dispositivo esta preparado para probarlo.

Figura 4.5: Teslametro con Arduino y reciclaje de desechos plásticos

4.4.4 Cargar el código fuente

Por último, subimos el código fuente a través de la interfaz de Arduino. Para su ejecución y puesta a punto.

Capitulo 5 : Comprobación

Durante este capitulo vamos a exponer los pasos seguidos para comprobar el correcto funcionamiento del dispositivo.

5.1 Pruebas caja blanca y caja negra

Una vez el dispositivo esta montado y el código finalizado, procedemos a realizar una serie de pruebas para comprobar su correcto flujo de datos, y la cohesión entre las distintas funciones.

Para ello primero realizamos la prueba de caja blanca que consiste en un seguimiento en la ejecución del programa. Mediante la impresión de comentarios al comenzar una función, durante su procedimiento y al salir de la función. De esta manera podemos observar si el dispositivo ejecuta todo el código. Además se forzara a recorrer todas la funciones a través de la interacción con la interfaz de usuario. Como resultado obtendremos un registro detallado de las distintas rutas que sigue la ejecución del código fuente.

Posteriormente se analiza dicho registro y se comprueba que no queda bloqueado en ninguna función y que realiza correctamente su funcionamiento tal como queda reflejado en los diagramas de flujo diseñados.

Por último se realiza la prueba de caja negra, de manera que se analiza el comportamiento del dispositivo a partir de las entradas proporcionadas a la interfaz de usuario y el resultado del comportamiento del dispositivo. El objetivo es comprobar que el dispositivo hace el comportamiento esperado.

5.2 Comprobación con otro Teslámetro

Seguidamente a la comprobación del correcto funcionamiento, es necesario contrastar la función de adaptación de la señal obtenida por el sensor. Para ello compararemos las medidas obtenidas con nuestro dispositivo con las muestras obtenidas por un Teslámetro comercial.

Haciendo uso del Teslametro PHYWE del laboratorio Nikola Tesla de la ETSINF, hemos contrastado el magnetómetro construido con Arduino. Tras unos cuantos errores, conseguimos el modelo correcto. Para contrastar que las medidas son correctas, medimos el campo magnético de una bobina la cual generaba hasta 25mT en su interior, nosotros pudimos medir el exterior, debido a la ausencia de sonda externa. Y además medimos el campo magnético de un imán.

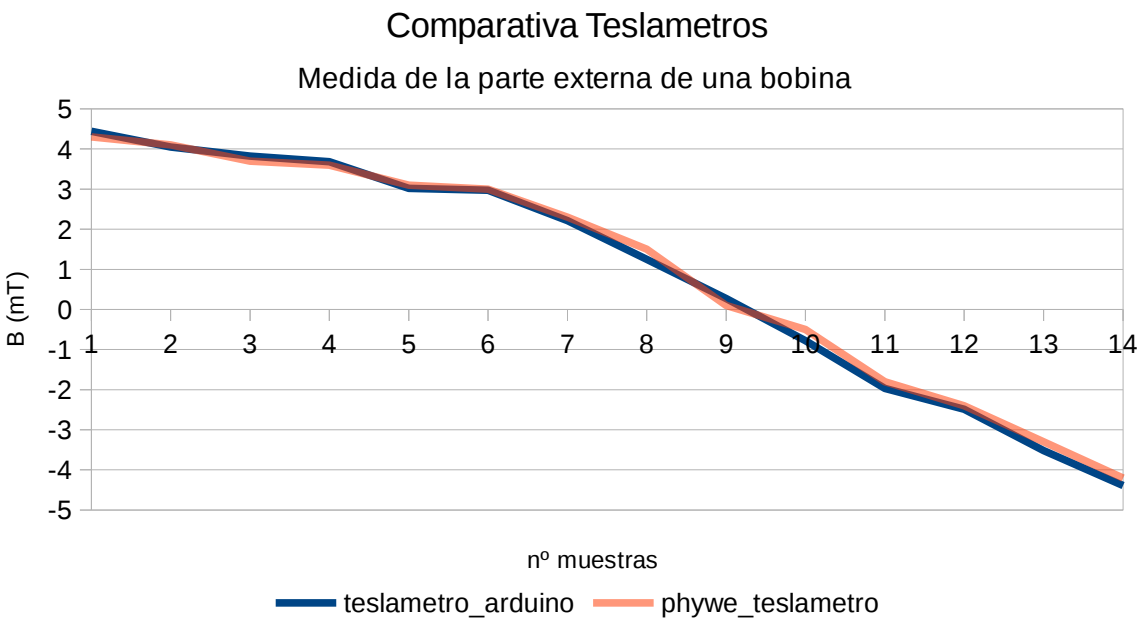

Figura 5.1: Gráfica comparativa teslametros midiendo una bobina

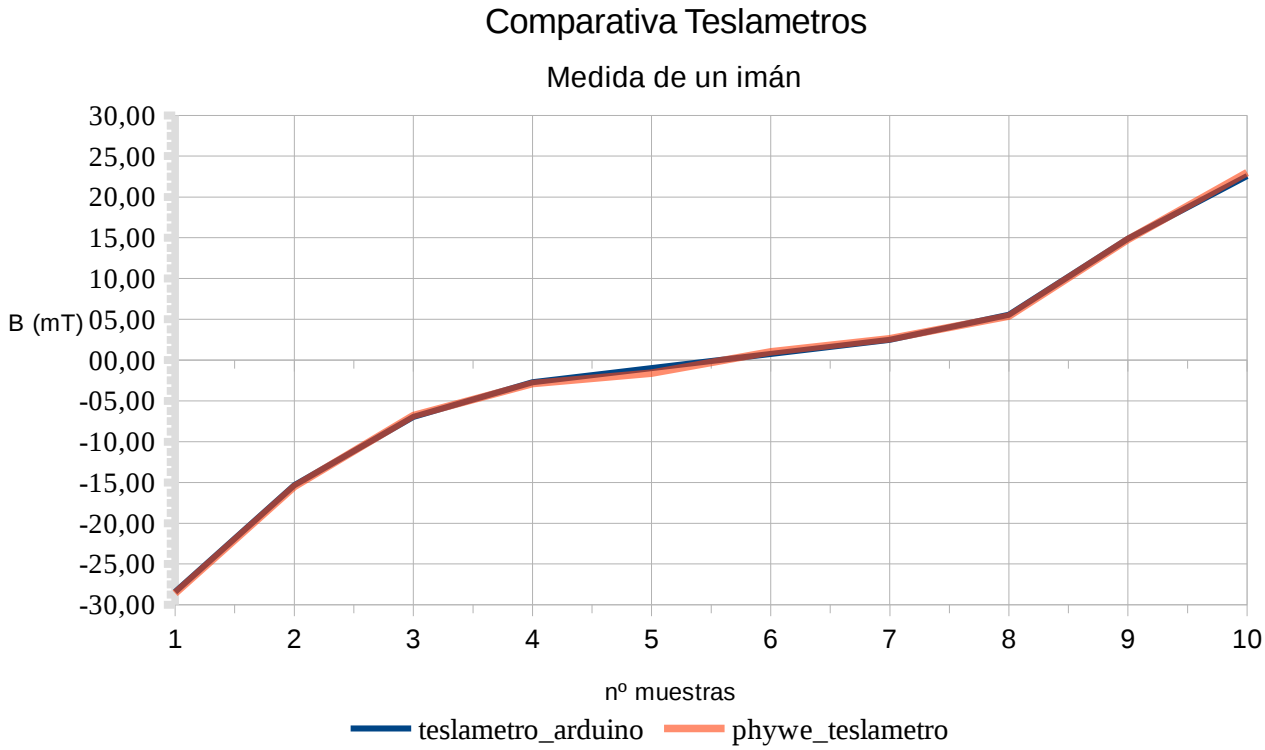

Figura 5.2: Gráfica comparativa teslametros midiendo un imán.

Capitulo 6 : Presupuesto

En este capitulo se desglosa la organización temporal seguida para el desarrollo de este proyecto, como además los presupuesto del hardware usados y del software desarrollado.

6.1 Diagrama de Gannt

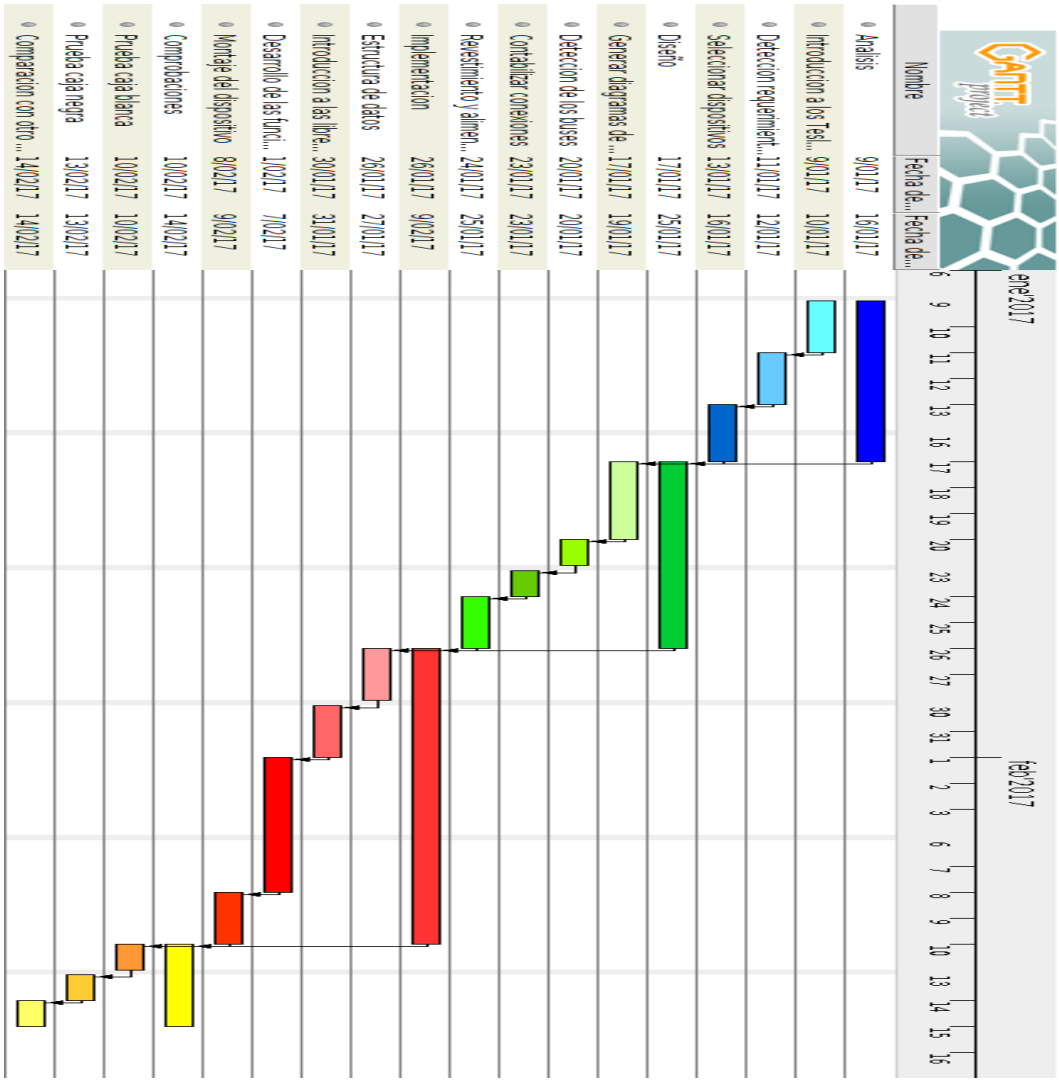

Figura 6.1: Planificación temporal para la realización del trabajo

En la figura 6.1 se puede observar la planificación temporal realizada para la elaboración del proyecto.

6.2 Presupuesto hardware

Los componentes del dispositivo se han comprado en la tienda on-line y en una tienda de electrónica convencional.

Componentes	Precio	Unidades	Precio total unidades
Arduino Due	33,06 €	1	33,06 €
Display LCD	7,87 €	1	7,87 €
Adalogger FeattherWing	9,22 €	1	9,22 €
Sensor Hall A1318	1,413 €	1	1,41 €
PCB matriz 70x70mm	4,73 €	1	4,73 €
Pulsadores	0,2664 €	6	1,5984 €
Resistencia fija 10kΩ	0,816 €	6	4,90 €
Potenciómetro 50kΩ	0,89 €	1	0,89 €
Cables de conexionado	0,08 €	40	3 €
		Total sin IVA	66,7674 €
		IVA 21%	14,00 €
		Total	80,68 €

Tabla 6.2: Presupuesto hardware de cada dispositivo.

6.3 Presupuesto software

En este último apartado del capitulo 6, vamos a presentar un presupuesto al trabajo realizado. Según el sueldo marcado por el **XVI CONVENIO COLECTIVO ESTATAL DE EMPRESAS DE CONSULTORÍA Y ESTUDIOS DE MERCADOS Y DE LA OPINIÓN PÚBLICA**, y según el tiempo planificado en el capitulo 6.1, obtenemos:

Trabajo	Horas	Precio hora	Precio del trabajo
Análisis	40	8 €	320 €
Diseño	40	8 €	320 €
Implementación	80	8 €	640 €
Comprobación	20	6 €	120 €
		Total sin IVA	1400 €
		IVA 21%	294 €
		Total	1694 €

Tabla 6.3: Presupuesto del trabajo por desarrollo del dispositivo.

6.4 Comparación con un Teslametro comercial

Modelo	Precio	Rango	Sensibilidad	Portabilidad
Teslametro Arduino	80.68 €	[-126.923 mT, 126.923 mT]	0,247 mT	Transformador 220 V AC a 5-7V DC o batería portátil USB
BST600 Gaussmeter	305,24 €	DC*1: 0.00~200.00mT DC*10: 0.0~2000.0mT AC*1: 0.00~200.00mT AC*10: 0.0~2000.0mT	DC*1 : 0.01mT DC*10 0.1mT AC*1 : 0.01mT AC*10 0.1mT	Transformador 220V AC a 9V DC o batería de 9V DC
Phywe Teslametro digital	2000 €	DC*1: 0.00~20.00mT DC*10: 0.00~200.00mT DC*100: 0.0~2000.0mT AC*1: 0.00~20.00mT AC*10: 0.00~200.00mT AC*100: 0.0~2000.0mT	DC*1: 0.01mT DC*1: 0.1mT DC*10: 1mT AC*1: 0.01mT AC*10: 0.1mT AC*100: 1mT	Conexión a 220V AC

Tabla 6.4: Tabla comparativa Teslametros

Capitulo 7 : Resultados y conclusiones

En este ultimo capitulo, vamos a exponer las mayores dificultades encontradas durante el desarrollo del proyecto, las experiencias del trabajo realizado y posibles ampliaciones en un futuro.

7.1 Problemas y contratiempos

En la elaboración del Magnetómetro, se han planteado algunas problemáticas, relacionadas con el software desarrollado y por el hardware escogido.

- La problemática más importante ha sido escoger la caja del dispositivo. La decisión de la auto-construcción de ella, ha sido por motivos ideológicos, teniendo en cuenta la cantidad de residuos generados y desechados, por lo que me veo obligado a contribuir en la desaparición y gestión de los residuos. Por otra parte es complicado escoger un tupper-ware de unas dimensiones adecuadas a todo el cableado y componentes del dispositivo. Al igual que las perforaciones de los conectores, botones y pantalla; donde realizamos unas cuantas pruebas antes de realizar las perforaciones finales, pero aun así los cortes no son perfectos y en ellos se ve la artesanía manual de su elaboración.

- Otra problemática software, es la forma de detectar la activación de los pulsadores. En un primer instante se planteo por comprobación de estado, resulto ser un fracaso; ya que muchas pulsaciones no eran detectadas, y ralentizaba la interacción del usuario con el dispositivo. Por tanto nos planteamos la detección por interrupciones, al principio de la implementación, no se había tenido en cuenta el efecto rebote o debounce. De esta manera el dispositivo se saturaba con interrupciones y dejaba de funcionar. Tras una pequeña inspección del problema se observo que requería de protección contra dicho efecto y se dispuso de un condensador de 6 uF en el circuito físico de la botonera, además de la implementación software explicada al principio del capitulo 4.3.2.7.

- Y la última complicación sufrida, recae en el proceso de adaptación de la señal captada por el sensor a la transformación del valor del campo en T o G. Habiendo estado más tiempo del planificado planteando posibilidades para la implementación, de las cuales todas eran planteamientos erróneos. Con unas cuantas semanas de planteamientos teóricos y probándolos sobre el dispositivo, obtuvimos el modelo correcto explicado en el capitulo 3.1.6.

7.2 Experiencias y valoraciones del trabajo realizado

En el transcurso del trabajo, nos hemos aproximado a los Magnetómetros y a la fabricación de dispositivos de laboratorio, con un precio muy reducido.

Hemos sido participes de la formación del dispositivo, desde la observación y análisis de los modelos comerciales, hasta su diseño e implementación, y como no de su comprobación y puesta a punto. Con ello hemos desarrollado nuestras capacidades de elaborar dispositivos, ya sea comprendiendo y desarrollando el software, como la detección y selección del hardware y puesto en practica de los métodos de pruebas de verificación del buen funcionamiento del dispositivo.

Con ello hemos practicado mucho del conocimiento adquirido durante el grado de informática. Y puesto en practica con un ejemplo real, el cual se va a replicar y utilizar en los laboratorios de física aporta una gran relevancia a la experiencia obtenida.

7.3 Ampliaciones

Para finalizar, vamos a exponer brevemente alguna de las posibles ampliaciones que da pie este proyecto. Desde la ampliación del Magnetómetro resultante, como la modificación del software para hacer que el dispositivo cumpla otro cometido.

- **Ampliación del Magnetómetro:** modificando el sensor Hall por uno del tipo vectorial, ampliaríamos nuestras mediciones del campo magnético en los tres vectores espaciales (X, Y, Z). Por otra parte, se puede modificar el firmware del puerto USB-Nativo para que se conecte a un PC y desde él puedas acceder a los datos almacenados en la tarjeta SD. Otra posible ampliación, seria una conexión bluetooh, para transmisión de datos en tiempo real de forma inalámbrica, o para acceder a los datos almacenados en la tarjeta SD. Y por supuesto se pueden añadir canales de medición adicionales, solamente con añadirle sensores; o se podría extraer el sensor a una sonda.

- **Ampliación software:** se puede ampliar el código fuente, introduciendo nuevos funcionamientos al dispositivo, como podría ser un generador de señales, un osciloscopio, u otros dispositivos de laboratorio. De esta manera conseguiríamos obtener en un único dispositivo varios aparatos necesarios para desarrollo de dispositivos electrónicos. Además, deja la posibilidad de rehacer el código en C++, creando una librería para la IDE de Arduino, o de escribir el funcionamiento del Magnetómetro a bajo nivel, realizando todo el proceso que en este proyecto se ha omitido utilizando las librerías disponibles en Arduino.

De esta manera se podría conseguir un dispositivo con una programación a alto nivel implementada en el IDE de Arduino, fácil de explicar y utilizar como ejemplo en asignaturas como DIP. Se tendría también el código escrito en C++ formando una librería, con la que se podría emplear en asignaturas como MEC; y por ultimo el código

a bajo nivel útil para asignaturas como AIC, AAV, DSD, IIN. Pudiendo tener un ejemplo común entre por ejemplo DIP, MEC y IIN las cuales están relacionadas pero sin más conexión que las temáticas tratadas, es decir, carecen de ejemplos comunes que hagan fácil visualizar la relación entre los distintos puntos de vista impartidos en cada una de las ellas.

7.4 Conclusiones

1. Es posible construir un Teslametro de un coste de construcción de 80 €.
2. Es posibles reciclar materiales para la construcción de la caja del Teslametro.
3. Es posible desarrollar un Teslametro con la plataforma Arduino.
4. Es posible conseguir una medidas comparables a las de un Teslametro comercial.

Bibliografía

[1] Adafruit. RTClib. Consultado el 05/02/2017

https://github.com/adafruit/RTClib/blob/master/RTClib.h

[2] Arduino. AnalogReadResolution. Consultado el 02/02/2017

https://www.arduino.cc/en/Reference/AnalogReadResolution

[3] Arduino. Arduino Due. Consultado el 01/02/2017

https://www.arduino.cc/en/Main/ArduinoBoardDue

[4] Arduino. Forum fat16lib. Consultado el 01/02/2017

https://forum.arduino.cc/index.php?topic=65558.0

[5] Arduino. HallEfect. Consultado el 03/02/2017

http://playground.arduino.cc/Code/HallEffect

[6] Arduino. LiquidCrystal. Consultado el 04/02/2017

https://www.arduino.cc/en/Reference/LiquidCrystal

[7] Arduino. Productos de Arduino. Consultado el 12/01/2017

https://www.arduino.cc/en/Main/Products

[8] Arduino. SD. Consultado el 05/02/2017

https://www.arduino.cc/en/Reference/SD

[9] Arduino. SPI. Consultado el 06/02/2017

https://www.arduino.cc/en/reference/SPI

[10] Arduino. Wire. Consultado el 07/02/2017

https://www.arduino.cc/en/Reference/Wire

[11] Boletin oficial del Estado. XVI CONVENIO COLECTIVO ESTATAL DE EMPRESAS DE CONSULTORÍA Y ESTUDIOS DE MERCADOS Y DE LA OPINIÓN PÚBLICA Consultado el 13/02/2017
https://www.boe.es/boe/dias/2009/04/04/pdfs/BOE-A-2009-5688.pdf

[12] BST600. AlliExpress. Consultado el 08/02/2017
https://es.aliexpress.com/store/product/BST600-Gaussmeter-Teslameter-AC-DC-magnetic-field-gauss-meter-Tesla-BST-600-0-200mT-2000mT/1422139_32622700574.html

[13] Datasheet A1318. AllegroMicro. Consultado el 08/02/2017
https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&ved=0ahUKEwiJl-6z-8bSAhWpBcAKHUVtB5MQFghEMAc&url=http%3A%2F%2Fwww.allegromicro.com%2F~%2Fmedia%2FFiles%2FDatasheets%2FA1318-A1319-Datasheet.pdf&authuser=1&usg=AFQjCNHQKewhUh9X4PWVjf1DSYEHHxV0Sw&sig2=OiXHcD2P8VVRrHC8rNQLPA&cad=rja

[14] Efecto Hall. Edu GVA. Consultado el 12/01/2017
http://intercentres.edu.gva.es/iesleonardodavinci/Fisica/Electromagnetismo/Electromagnetismo07b.htm

[15] Real Academia de la Legua Española. Definición de magnetómetro. Consultado el 15/01/2017
http://dle.rae.es/?id=NtmaBLd

[16] RSComponents. Productos. Consultado el 13/01/2017

http://es.rs-online.com/web/

[17] Magnetómetro. 5HertzElectronica. Consultado el 12/01/2017

http://5hertz.com/tutoriales/?p=623#mgn2.

[18] Wikipedia. Magnetómetro. Consultado el 12/01/2017

https://es.wikipedia.org /wiki/Magnet%C3%B3metro

Apéndice

Código fuente

```
0  #include <LiquidCrystal.h>
1  #include <SPI.h>
2  #include <SD.h>
3  #include <Wire.h>
4  #include "RTClib.h"
5
6  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
7  ////////////////////////////////////////// Definición de variables ////////////////////////////////////////////////
8  ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
9  //acceso rápido al pin dedicado a botones o pines de actuadores o sensores
10 //botones llamada a ISR
11 #define HOLD  20
12 #define RANGO 3
13 #define UNITS 18
14 #define BOTONLLUM 22
15 #define BORRADOR 19
16 #define MAXMIN 21
17 #define EXTRACT 2
18 //sensor
19 #define SENSOR A0
20 //salida iluminacion pantalla
21 //#define LLUM  10
22 //entrada cs modulo sd
23 #define CS  53
24
25 //el reloj rtc de adaloger usa los puerto SDA1 y SCL1 en Arduino Due. En Arduino Mega usan
26 //SDA y SCL que son el pin 20 y 21, tenemos un conflicto de pines para Arduino Mega, con
27 //Hold y MaxMix
28
29 RTC_Millis rtc;
30
31 //variables para medición tiempo
32 unsigned long currentMillis, previousMillis = 0;
33 const long tiempo = 500, tiempoDisplay = 1000;
34
35 //para anular el debounce de la botonera
36 const int timeThreshold = 150;
37 int counterHold, counterLuz, counterRango, counterUnits, counterMaxMin, counterExtract= 0; int
38 counterBorrado = 0;
39 unsigned long currentHold, currentLuz, currentRango, currentUnits, currentMaxMin, currentExtract,
40 currentBorrado = 0;
```

```
41
42 //variables volátiles para la gestión de interrupciones
43 volatile boolean HOLDS = false; //botón hold
44 volatile boolean TESLA = true;  //botón units
45 volatile boolean MAXIMOMINIMO = false; //botón MAXMIN
46 volatile boolean extraida = false; //botón MAXMIN
47
48 //variables sd
49 boolean iniSd = false;
50 boolean errorWrite  = false;
51 unsigned long numMuestra = 0;
52 String nombreFichero = "datos.txt";
53
54 //variables para aplicar a la función hold y maxMin
55 struct lecturaSensor {
56   int valorOriginal;
57   double valor;
58   String polaridad;
59   int rango;
60   boolean unidad;
61   String magnitud;
62   DateTime fechadora;
63 };
64
65 //estructura de datos para almacenar la información acerca de una muestra.
66 DateTime dt0 (0, 0, 0, 0, 0, 0);
67 lecturaSensor muestra, valorMantener, muestraMax = {1000, 0, "", 0, true, "", dt0};
68 lecturaSensor muestraMin = {0, 0, "", 0, true, "", dt0};
69
70 //variables de rangos;
71 double rangoT[3]= {0.001, 1.0, 1000.0};
72 double rangoG[3]= {0.001, 1, 1000};
73 int RANGOACTUAL = constrain (RANGO,ACTUAL, 0, 2);
74
75 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
76 ////////////////////////////////////Variables y métodos Modulo SD/////////////////////////////////////////////////////
77 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
78 Sd2Card card;
79 SdVolume volume;
80
81 File archivoDatos;
82
83 void inicializaSD(){
84 //Serial.println("Inicialización del modulo SD:");
85 //intentamos inicializar la tarjeta sd. si no inicializa lo marcamos con la variable iniSd a false
86 //para reintentarlo mas tarde
```

```
 if (!SD.begin(CS)) {
          iniSd = false;
     }else{
          iniSd = true;
     }
   //habilita la tarjeta de forma que se puede leer su configuración
     if (!card.init(SPI_HALF_SPEED, CS)) {
       //Serial.println("Inicializada la tarjeta");
     }
     //comprueba si existe el archivo de texto, sino esta lo crea y lo cierra.
     if(!SD.exists(nombreFichero)){
        creaFichero();
        //Serial.println("Inicializado el fichero");
     }
   //Serial.println("Completada");
}

//si abre bien el archivo imprime la lectura y el estado de la memoria, sino modifica la variable
//errorWrite para mostrar un error en display lcd
void guardarMuestra(lecturaSensor ls){
   //Serial.println("Proceso de guardado:");
   archivoDatos = SD.open(nombreFichero, FILE_WRITE);
    //Serial.println("  Se ha abierto el fichero: "+nombreFichero);
    if (archivoDatos){
          //Guarda numMuestra ls.polaridad ls.valor magnitud Maximo: muestraMax.polaridad
muestraMax.valor magnitud Mínimo: muestraMin.polaridad muestraMin.valor magnitud
          //archivoDatos.println("N 30 mG; Maximo; Mínimo; Hora ... ");
          archivoDatos.println(ls.polaridad+" "+ls.valor+" "+ls.magnitud+ "; "+
muestraMax.polaridad+" " +muestraMax.valor+" "+muestraMax.magnitud+ "; "+
muestraMin.polaridad+" "+muestraMin.valor+" "+muestraMin.magnitud+ "; "+
ls.fechaYHora.day()+"/"+ls.fechaYHora.month()+"/"+ls.fechaYHora.year()+ "; "+
ls.fechaYHora.hour()+":"+ls.fechaYHora.minute()+":"+ls.fechaYHora.second());
          archivoDatos.close();
          errorWrite = false;
          //Serial.println("Se ha guardado la muestra: "+numMuestra);
    }else{
          //Error al abrir el archivo
          errorWrite = true;
          //Serial.println("Error al guardar la muestra: "+numMuestra);
    }
}

void borradoInformacion(){
  archivoDatos = SD.open(nombreFichero, FILE_WRITE);
  SD.remove(nombreFichero);
  //Serial.println("Se ha borrado el archivo: "+nombreFichero);
```

```
  creaFichero();
}
void creaFichero(){
    DateTime ahora = rtc.now();
    archivoDatos = SD.open(nombreFichero, FILE_WRITE);
    archivoDatos.print("Lectura del "+ahora.day());
    archivoDatos.print(+"/"+ahora.month());
    archivoDatos.print(+"/"+ahora.year());
    archivoDatos.print(+" a las "+ahora.hour());
    archivoDatos.print(+":"+ahora.minute());
    archivoDatos.print(+":"+ahora.second());
    archivoDatos.println();
    archivoDatos.close();
    //Serial.println("Se ha creado el archivo: "+nombreFichero);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////// variables de pantalla ////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//definición de la pantalla lcd con parámetros RS pin 8, E pin 9, d4-7 pines 4-7
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

//variable control iluminacion
boolean apagada = true;

//Inicializa la pantalla lcd y muestra el mensaje inicial
void inicializizaPantalla(){
    //Serial.println("Inicialización del display:");
    lcd.begin(16, 2);              // Inicializar el LCD
    lcd.clear();                  //limpia la pantalla
    //Serial.println("Completada");
}

void impresionMuestra(lecturaSensor ls){

    if(!MAXIMOMINIMO){
        //Serial.println("Se imprime en el display la muestra actual:");
         lcd.setCursor(1,0);
         //muestra la polaridad de la señal magnética captada.
         lcd.print(ls.polaridad);

         //imprime en la parte superior de la pantalla el valor obtenido junto con la magnitud en
          //la que esta expresado

         if(ls.valor<0){
           lcd.setCursor(3,0);
```

```
179             lcd.print(ls.valor);
180             lcd.print(ls.magnitud);
181            }else{
182             lcd.setCursor(4,0);
183             lcd.print(ls.valor);
184             lcd.print(ls.magnitud);
185            }
186           //Serial.println(ls.polaridad+" "+ls.valor+" "+ls.magnitud);
187           lcd.setCursor(1,1);
188           if(HOLDS)  lcd.print("H");
189           lcd.setCursor(3,1);
190           if(TESLA)  lcd.print("T");
191           else lcd.print("G");
192           lcd.setCursor(5,1);
193           lcd.print(RANGOACTUAL);
194           //Serial.print("Mantener: ");
195           //Serial.print(HOLDS);
196           //Serial.print(", Unidades(Tesla = true, Gauss = false): ");
197           //Serial.print(TESLA);
198           //Serial.print(", Rango: ");
199           //Serial.print(RANGOACTUAL);
200           //Serial.print(", ");
201           //muestra información de la sd
202           lcd.setCursor(7,1);
203            if(!extraida){
204              if(!iniSd){  lcd.print("Fallo SD");
205                           //Serial.print("Modulo SD: FalloSD ")
206              }else{
207                switch (card.type()) {
208                    case SD_CARD_TYPE_SD1:
209                      lcd.print("SD1 ");
210                      //Serial.print("Modulo SD: SD1 ");
211                    break;
212                    case SD_CARD_TYPE_SD2:
213                      lcd.print("SD2 ");
214                      //Serial.print("Modulo SD: SD2 ");
215                    break;
216                    case SD_CARD_TYPE_SDHC:
217                      lcd.print("SDH ");
218                      //Serial.print("Modulo SD: SDH ");
219                    break;
220                    default:
221                      lcd.print("NN ");
222                      //Serial.print("Modulo SD: ERROR DISK ");
223                }
224                if(!errorWrite){
```

```
                lcd.print("FAT");
                //Serial.print("FAT ");
                //indica el tipo de formato de fat que tiene la sd
                if (!volume.init(card)){
                   lcd.print(" ERROR");
                   //Serial.print("ERROR ");
                   SD.begin(CS);
                }else
                   lcd.print(volume.fatType(),DEC);
                   //Serial.print(volume.fatType(), DEC);
              }else{
                 lcd.print("ERR W");
                 //Serial.print("Error de escritura");
              }
            }
          }else{
            lcd.print("Extract");
            //Serial.println("Tarjeta extraida");
          }
      }else{
       //Serial.println("Se imprime al display las muestras máxima y mínima: ");
        lcd.setCursor(2,0);
        //muestra la polaridad de la máxima señal magnética captada.
        lcd.print(muestraMax.polaridad);

        //imprime en la parte superior de la pantalla el valor obtenido junto con la magnitud en la
        //que esta expresado
        lcd.setCursor(4,0);
        lcd.print(muestraMax.valor);
        lcd.print(muestraMax.magnitud);
        //Serial.println("Maximo: "+muestraMax.polaridad+" "+muestraMax.valor+"
"+muestraMax.magnitud);
        lcd.setCursor(2,1);
        //muestra la polaridad de la señal magnética captada.
        lcd.print(muestraMin.polaridad);

    //imprime en la parte superior de la pantalla el valor obtenido junto con la magnitud en la
    //que esta expresado
        lcd.setCursor(4,1);
        lcd.print(muestraMin.valor);
        lcd.print(muestraMin.magnitud);
        //Serial.println("Mínimo: "+muestraMin.polaridad+" "+muestraMin.valor+"
"+muestraMin.magnitud);
      }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
271 ////////////////////////////////////////////// variables de sensor hall ////////////////////////////////////////////////
272 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
273
274 //variables para el tratamiento del valor del sensor hall.
275
276 //1.3mV = 1Gauss
277 const double VMax = 3300.0;
278 const double bitsResolucion = 1024;
279 const double VSensor = 1.3;
280
281 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
282 ////////////////////////////////////////////// métodos sensor hall ////////////////////////////////////////////////////
283 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
284
285 lecturaSensor leeValorSensor(){
286   //Serial.println("Lectura del Sensor");
287   int valorOriginalNou = analogRead(SENSOR)-18;
288   double valorNou = convertirValor(valorOriginalNou);
289   String polNou = "";
290   if(valorOriginalNou >= bitsResolucion/2)
291         polNou = "S";
292    else
293         polNou = "N";
294
295 lecturaSensor nova ={valorOriginalNou, valorNou , polNou, RANGOACTUAL, TESLA, "", rtc.now()};
296   nova = magnitud(nova);
297   numMuestra++;
298   return nova;
299 }
300
301 //* Método de transformación del valor recogido por el sensor. Dependiendo de la unidad
302 seleccionada, se transformara a Teslas o a Gauss, viene dado por el valor de la variable
303    boleana Tesla, si esta a true, esta seleccionada la magnitud estándar del sistema internacional Tesla,
304 mientras que si esta en false se mostrara en Gauss.
305    Este método retorna un double, con el valor transformado.
306    En definitiva el dispositivo puede medir entre [-128 mT. 128 mT]
307 */
308
309 double convertirValor(int val){
310 //Serial.println("Se procesa el valor captado aplicándole la ecuación de adaptacion: (val*((VMax/
311 (VSensor*10))/(bitsResolucion)))-((VMax/(VSensor*10))/(2))");
312   double res = (val*((VMax/(VSensor*10))/(bitsResolucion)))-((VMax/(VSensor*10))/(2)); //esto
313 calcula el valor en mT
314   if(TESLA){//en tesla
315         res = res * rangoT[RANGOACTUAL];
316   }else{//en gauss
```

```
317         res = res * rangoG[RANGOACTUAL]*10000;
318   }
319   //Serial.println("Finalizada el calculo de la ecuación de adaptacion de la señal.");
320   return -res;
321 }
322
323 lecturaSensor magnitud(lecturaSensor ls){
324     if(ls.unidad){
325       if(ls.rango==0)
326           ls.magnitud = " T";
327       else if(ls.rango==1)
328           ls.magnitud = " mT";
329       else if(ls.rango==2){
330           ls.magnitud = " uT";
331       }
332     }else{
333         if(ls.rango==0)
334           ls.magnitud = " kG";
335         else if(ls.rango==1)
336           ls.magnitud = " G";
337         else if(ls.rango==2){
338           ls.magnitud = " mG";
339         }
340     }
341     return ls;
342 }
343 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
344 ////////////////////////////////////////////// Tratamiento de interrupciones ////////////////////////////////////////////////
345 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
346
347 //interrupción de mantener
348 void mantener(){
349             //Serial.println("Invocación de la rutina mantener()");
350             currentHold = millis();
351             if (currentHold > counterHold + timeThreshold) {
352                         HOLDS = !HOLDS;
353                         counterHold = millis();
354             }
355           //Serial.println("Finalizada la rutina mantener()");
356 }
357 //interrupción de unidades
358 void unidades() {
359             currentUnits = millis();
360             if (currentUnits > counterUnits + timeThreshold){
361                         TESLA = !TESLA;
362                         counterUnits = millis();
```

```
363             }
364 }
365 //interrupción de iluminacion
366 /*void iluminacion(){
367             currentLuz = millis();
368             if (currentLuz > counterLuz + timeThreshold) {
369                     if (apagada) {
370                             //enchufar la luz
371                             digitalWrite(LLUM, HIGH);
372                     }
373                     else {
374                             //apagar la luz
375                             digitalWrite(LLUM, LOW);
376                     }
377                     apagada = !apagada;
378                     counterLuz = millis();
379             }
380 }*/
381 //Incrementa el RANGOACTUAL de forma cíclica
382 void rangos(){
383             //Serial.println("Invocación de la rutina rangos()");
384             currentRango = millis();
385             if (currentRango > counterRango + timeThreshold) {
386                     if (RANGOACTUAL == 2)
387                             RANGOACTUAL = 0;
388                     else
389                             RANGOACTUAL++;
390                     counterRango = millis();
391             }
392            //Serial.println("Finalizada la rutina rangos()");
393 }
394 //interrupción de MAXIMOMINIMO
395 void maximoMinimo(){
396             //Serial.println("Invocación de la rutina maximoMinimo()");
397             currentMaxMin = millis();
398             if (currentMaxMin > counterMaxMin + timeThreshold) {
399                     MAXIMOMINIMO = !MAXIMOMINIMO;
400                     counterMaxMin = millis();
401             }
402             //Serial.println("Finalizada la rutina maximoMinimo");
403 }
404 //interrupción de EXTRACT
405 void extraerConSeguridad(){
406            //Serial.println("Invocación de la rutina extraerConSeguridad()");
407             currentExtract = millis();
408             if (currentExtract > counterExtract + timeThreshold) {
```

```
409                     if (extraida) {
410                             iniSd = true;
411                             extraida = false;
412                     }
413                     else {
414                             extraida = true;
415                              iniSd = false;
416                     }
417                     counterExtract = millis();
418             }
419             //Serial.println("Finalizada la rutina extraerConSeguridad()");
420 }
421 //método de borrado del archivo de datos. Al ser invocado se eliminara el archivo de datos dejándolo
422 otra vez vacío
423 void borrado(){
424   //Serial.println("Invocación de la rutina borrado()");
425   currentBorrado = millis();
426     if (currentBorrado > counterBorrado + timeThreshold) {
427      borradoInformacion();
428     counterBorrado = millis();
429    }
430   //Serial.println("Finalizada la rutina borrado()");
431
432 }
433 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
434 ////////////////////////////////////////Tratamiento de la lectura de la muestra//////////////////////////////////////
435 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
436
437 / * Este método toma una lectura del sensor actualizando la struct muestra.
438 * Luego dependiendo de las interrupciones HOLD, UNIT, MAXMIN modifican el comportamiento
439  *
440  *      -HOLD: dependiendo de una variable boolean utiliza la struct muestra o valorMantener
441  *      -UNIT: varia entre Tesla o Gauss, las unidades de medida de campo magnético.
442  *      -RANGO: varia la variable  rangoActual entre 0, 1, 2. donde además varia el valor en la
443  *        conversión del valor leído del sensor.
444  * /
445 void tratarMuestra(){
446     //Serial.println("Inicio de la gestión de la muestra: "+numMuestra);
447     //tomamos una muestra
448     muestra = leeValorSensor();
449     //tratamiento de los máximos y mínimos, queda definir la diferencia entre los rangos
450     if(muestra.valorOriginal>muestraMax.valorOriginal){
451       muestraMax = muestra;
452      //Serial.println("La muestra es maximo");
453     }
454     if(muestra.valorOriginal<muestraMin.valorOriginal){
```

```
455       muestraMin = muestra;
456       //Serial.println("La muestra es mínimo");
          }
457
         //tratamiento de hold
458      if(HOLDS){
459       impresionMuestra(valorMantener);
460       //Serial.println("Se congela la muestra a imprimir");
461      }else{
462       impresionMuestra(muestra);
463       valorMantener = muestra;
464       //Serial.println("Se imprime la muestra actual");
465      }
466      if(!extraida){
467       //Serial.println("Tarjeta no extraida, se procede a guardar la muestra: "+numMuestra);
468       guardarMuestra(muestra);
469      }
470  }
471  /*
472     Primero que todo se inicializan los pines para los botones HOLD, UNITS, RANGO, MAXMIN,
473  LLUM, CS, BOTONLLUM
474       luego se inicializan las interrupciones asociadas a cada botón con su respectivo código de
475  tratamiento de interrupciones
476     Posteriormente se calibra el sensor, y se inicializan el resto de módulos (Display LCD, SD, RTC)
477     */
478  void setup() {
479     //Serial.begin(9600);
480  //Serial.println("Inicio del registro prueba caja blanca:");
481   //inicialización de los pines
482   //Serial.println("Inicialización de los pines HOLD, UNITS, RANGO, MAXMIN, CS, EXTRACT,
483  BORRADOR y LED_BUILDIN");
484   pinMode(HOLD, INPUT);
485   analogWrite(HOLD, LOW);
486   pinMode(UNITS, INPUT);
487   analogWrite(UNITS, LOW);
488   pinMode(RANGO, INPUT);
489   analogWrite(RANGO, LOW);
490  pinMode(MAXMIN, INPUT);
491   analogWrite(MAXMIN, LOW);
492   pinMode(CS, INPUT);
493    pinMode(BORRADOR, INPUT);
494   analogWrite(BORRADOR, LOW);
495   pinMode(LED_BUILTIN, OUTPUT);
496   //Serial.println("Inicialización de los pines completada.");
497   //Serial.println("Inicialización de las interrupciones HOLD, UNITS, RANGO, MAXMIN,
498  EXTRACT y BORRADOR");
```

```
499  //inicialización de las interrupciones
500  attachInterrupt( digitalPinToInterrupt(HOLD), mantener, FALLING );
501  attachInterrupt( digitalPinToInterrupt(UNITS), unidades, FALLING );
502  attachInterrupt( digitalPinToInterrupt(RANGO), rangos, FALLING );
503  attachInterrupt( digitalPinToInterrupt(MAXMIN), maximoMinimo, FALLING );
504  attachInterrupt( digitalPinToInterrupt(EXTRACT), extraerConSeguridad, FALLING );
505  attachInterrupt( digitalPinToInterrupt(BORRADOR), borrado, FALLING );
506  //En Arduino Due los pines digitales funcionan a 3,3V, por lo tanto no se puede usar la iluminación,
507 por contra podemos disponer de un calibrado del sensor por ejemplo.
508  //attachInterrupt( digitalPinToInterrupt(BOTONLLUM), iluminacion, FALLING );
509
510  //invocación de métodos de inicialización del sensor y pantalla
511  calibradoInicial();
512  inicializizaPantalla();
513  inicializaSD();
514  //inicializando el relog
515  //Serial.println("Inicialización del modulo RTC");
516  rtc.begin(DateTime(F(__DATE__), F(__TIME__)));
517 //Serial.println("Completada");
     //Serial.println("Finalizada la inicialización del Teslametro. Activación de su funcionamiento:");
     }

     /*
      El cuerpo principal del magnetómetro, realiza una limpieza de pantalla para posteriormente tomar
      una lectura del sensor actualizando el estado de la struct lecturaSensor, mostrando el valor en
      pantalla y almacena una linea mas en un txt de la sd. Seguido de un delay, con el tiempo mínimo
      para que se guarde y se muestre correctamente
      */

     void loop() {
      currentMillis = millis();
      if(currentMillis - previousMillis > tiempo){
           previousMillis = currentMillis;
            digitalWrite(LED_BUILTIN, HIGH);
           lcd.clear();
           //invocación al método principal
           tratarMuestra();
        }else digitalWrite(LED_BUILTIN, LOW);
        //Serial.println("Finaliza la muestra: "+numMuestra);
     }
```