Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Calibrador de Procesos

Autor:

Pérez Trigueros, Samuel

Tutor:

González de la Fuente, José Manuel

Dpto. de Tecnología Electrónica

Valladolid, Marzo, 2019.

Resumen

En este Trabajo de Fin de Grado se ha realizado el diseño y construcción de un prototipo de calibrador de procesos. Se trata de un instrumento portátil que permite la generación de tensión, corriente y frecuencia de forma precisa, así como la simulación de un termopar. La programación y control se basan en un Arduino Mega 2560.

Palabras clave

Arduino, calibrador, generación, prototipo, termopar

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

La calibración se entiende como la operación de comparar la salida de un equipo de medida frente a la salida de un patrón de exactitud conocida, cuando la misma entrada (magnitud medida) es aplicada a ambos instrumentos. En otras palabras, se trata de medir el buen funcionamiento de unos aparatos de medición usando equipos específicos cuya precisión ha sido a su vez comprobada.

Refiriéndose más específicamente al ámbito industrial, la calibración de los distintos instrumentos de medición y regulación presentes tanto en los procesos productivos como en los de control, calidad y seguridad es de vital importancia, igual o más que la propia ausencia de tales instrumentos, dado que podría suponer la total falta de cumplimiento de los objetivos de estos procesos.

Concretando, hay varias razones principales para efectuar la calibración de los distintos equipos, como son: el cumplimiento de la normativa, por ejemplo, en cuanto a las estrictas reglas sobre contaminación cuya infracción repercute tanto en el impacto ambiental de la empresa como en problemas legales; el cumplimiento de los criterios de calidad, en particular las normas ISO 9000, cuya certificación implica auditorías y revisiones periódicas; el rendimiento industrial, dado que una mínima desviación en un parámetro puede suponer que el producto pierda propiedades y por tanto valor, o que los consumos de algunas materias primas se podrían incrementar inútilmente, impactando la rentabilidad de la empresa; y la seguridad, ya que en procesos delicados, una variación fuera de los rangos aceptados podría representar un riesgo.

Poniéndose por ejemplo un proceso en el que un determinado compuesto químico debe permanecer a una temperatura concreta, se mide dicha temperatura con un sensor apropiado. Un error de 1mV en la señal que transmite dicho sensor puede corresponder con un error de 20 °C en la medida de la temperatura, lo que podría suponer desde la pérdida de propiedades del producto final hasta un riesgo para la propia seguridad de la planta.

Los equipos específicos que se utilizan para llevar a cabo la calibración de los instrumentos de medición son los llamados calibradores de procesos. Estos calibradores suelen ser equipos de mano portátiles, ligeros, compactos y robustos, ya que suelen ser empleados tanto en los bancos de pruebas en los laboratorios de calibración como en la propia planta o en campo. Se emplean para medir o generar señales de varios tipos, como eléctricas (tensión o corriente), de temperatura, de presión, o una combinación de ellas.

2. Estado del Arte

En el mercado se pueden encontrar una amplia variedad de calibradores de procesos portátiles diseñados para el ámbito industrial por multitud de marcas especializadas en la fabricación de instrumentos de medición y calibrado.

2.1. Fluke 726

Una de las principales empresas del sector de la instrumentación es *Fluke Corporation.* Este fabricante cuenta con multitud de tipos de calibradores, tanto multifunción (*Fluke 726),* que sirven para medir y calibrar casi cualquier aparato presente en un proceso industrial, como específicos para calibrar un solo tipo de señal. Ejemplos de estos calibradores son los de lazo (*Fluke 709*), dedicados a medir y servir de fuente de alimentación en un bucle; de presión (*Fluke 721-3630*), pudiendo realizar mediciones simultáneas de presión estática y diferencial; o de temperatura, ya sean termopares (*Fluke 741B*) o RTD (*Fluke 712B*).

Figura 1.Fluke 726

Atendiendo más concretamente al modelo de calibrador multifunción *Fluke 726* se pueden describir algunas de sus características principales:

- Medición, generación y simulación de tensión, mA, RTD, termopares, resistencia, frecuencia y presión.
- Precisión en la medición y generación de tensión y mA de hasta el 0.01%.

- Medición de tensión de hasta 30.000V, generación de 10.000V.
- Medición y generación de hasta 24.000mA.
- Medición y generación de señales de forma simultánea mediante dos canales independientes.
- Medición y generación de presión mediante uno de los módulos de presión *Fluke 700Pxx*.
- Genera mA con medida de presión simultánea para realizar pruebas de I/P y válvulas.
- Capacidad de memoria de hasta 8 resultados de calibración para su análisis posterior.
- Cálculo de error del transmisor e interpretación de los resultados de calibración.
- Efectúa comprobaciones de linealidad con las funciones incremento y rampa automáticas.
- Almacenamiento de las configuraciones de prueba más usuales para facilitar futuras calibraciones.
- Capacidad de alimentar hasta 24V los transmisores mientras se mide simultáneamente su salida en mA.
- Comunicación HART.
- Las curvas RTD personalizadas añaden constantes de calibración para la realización de calibraciones a RTD homologadas en calibraciones de temperatura.
- Batería formada por 4 pilas alcalinas AA.
- Peso 650g.

Se comprueba que este calibrador es muy versátil y preciso, por lo que puede utilizarse para la calibración de casi cualquier instrumento presente en la industria de procesos, además de facilitar la calibración gracias a la capacidad de almacenar e interpretar los resultados obtenidos.

2.2. Beamex MC6

Otra de las empresas líder en proveer soluciones para la calibración es *Beamex.* Un modelo de calibrador más exigente que el anterior de *Fluke* lo encontramos de la mano de esta marca con su *Beamex MC6,* un calibrador de campo de gran precisión y comunicador. Algunas de sus características principales son:

- Medición de voltaje de hasta 60V, con una resolución en el rango más amplio de 0.01mV.
- Generación de hasta 24V, con una resolución de 0.0001V.

- Medición de corriente de ±101mA con una resolución de 0.001mA.
- Generación de corriente de ±55mA con resolución 0.001mA.
- Medición y generación de frecuencia (0,0005 – 50000Hz).
- Medición y generación de pulsos (0 – 10M pulsos).
- Medición y simulación de resistencias (0 – 4000 Ω).
- Medición y simulación de termopares y RTD.
- Medición de presión (módulos internos y externos).
- Función como registro de datos para almacenar la medición de señales recogidas durante periodos de tiempo.
- Función como documentador de las calibraciones realizadas.
- Comunicación HART, FOUNDATION Fieldbus, Profibus PA.
- Pantalla LCD táctil de 5.7".
- Batería Li-Po de 4200mA, con autonomía de 10-16h.
- Peso 1.5Kg - 2.0Kg (dependiendo del modelo de caja).

Figura 2. Beamex MC6

Este calibrador es muy avanzado y cuenta con unas características que lo hacen idóneo para instrumentos y procesos que requieran más fiabilidad y precisión, así como versatilidad y eficiencia en la recogida de datos y documentación de la calibración.

2.3. PCE-123

Los dos calibradores anteriores soy muy precisos, pero también tienen un precio muy elevado. Por ello, para tener un marco de referencia más completo se va a poner como último ejemplo el calibrador de procesos *PCE-123* del fabricante *PCE Instruments,* que tiene unas características más modestas y por tanto un precio mucho más reducido. Algunas de sus características son:

- Generación de tensión: 0 – 100.00mV / 0 – 1.0000V / 0 – 12.000V
- Generación de corriente de hasta 24.000mA
- Simulación de termopares J, K, E, T con una resolución de 1°C.
- Generación de frecuencia: 1 – 125Hz (resolución de 1Hz) / 126 – 62500Hz (resolución de 604 pasos).
- LCD de 5 posiciones.
- Alimentación mediante 1 batería de 9V, o 6 de 1.5V.
- Consumo de entre 60mA y 180mA dependiendo de la salida.
- Peso 330g.

Figura 3. PCE - 123

Este calibrador ofrece menos bondades que los dos anteriores, ya que solo tiene función como generador de señales y no como medidor, no trabaja con RTDs ni con presión y la precisión es menor. Por otro lado, se asemeja más a las especificaciones que se pueden llegar a conseguir en el diseño de calibrador de procesos sobre el que trata el presente proyecto.

3. Objetivo

El objetivo de este proyecto es diseñar y construir un prototipo de calibrador de procesos funcional. El diseño incluirá el estudio y elección de los componentes que mejor posibiliten las características buscadas, la construcción de un prototipo con los componentes escogidos, y el diseño del programa que permita su control y funcionamiento.

Este prototipo deberá asemejarse a los calibradores de procesos que se pueden encontrar en el mercado, pero siendo conscientes de que las especificaciones conseguidas en los modelos más precisos no se van a poder lograr con los recursos, tiempo y conocimientos propios de un Trabajo de Fin de Grado. Por esto, el fin que se plantea para este calibrador está relacionado con el ámbito académico, de forma que se pueda convertir en un recurso disponible para la enseñanza. También se espera que el diseño sea fácilmente repetible, de forma que se puedan fabricar réplicas del calibrador para su utilización por parte del alumnado, y sin conllevar ello el alto coste monetario que supondría su compra.

Para decidir las funciones que tendrá el calibrador se tomará como referencia el calibrador *PCE-123* por tener unas características más asumibles que otros modelos más costosos, pero siendo suficientes para conseguir satisfacer las expectativas de uso que se le pudiera llegar a dar. De esta forma se pretende que el calibrador sea capaz de suministrar tensión, corriente y frecuencia controladas, así como poder simular la respuesta de temperatura de algunos modelos de termopar. Además, debe ser portátil, con una autonomía aceptable, intuitivo de utilizar y posible de replicar.

4. Diseño Propio

4.1. Especificaciones

El Calibrador de Procesos diseñado en este proyecto debe poder suministrar tensión, corriente, frecuencia y simular termopares. Los rangos de las distintas magnitudes generadas deben ser de un orden similar a los ofrecidos por un calibrador de procesos comercial, ya que son los valores típicos que se presuponen necesarios para la mayoría de fines a que están destinados. De esta forma, atendiendo a las especificaciones generales del calibrador *PCE-123*, se ha decidido que la tensión comprenda el rango de 0.0000V – 10.000V; la corriente tenga el rango de 4.000mA – 20.000mA; la frecuencia alcance entre 1Hz y 62500Hz; y se puedan simular termopares de los tipos J, K, E y T en todo el rango para los cuales están definidos cada uno.

Otra de las características buscadas en el calibrador es que la interfaz sea lo más sencilla e intuitiva posible, sin perder por ello funcionalidades. Para ello se ha decidido que los valores de salida se cambien por medio de un encoder, se disponga de pulsadores para cambiar entre los diferentes modos de funcionamiento, la información sobre el valor de salida y modo actual puedan leerse en una pantalla, y que los conectores de salida sean los menos posibles para no tener que conectar y desconectar constantemente. Adicionalmente, se dispondrá de un interruptor general para encender y apagar el dispositivo.

Los calibradores de procesos comerciales suelen ser portátiles y robustos, por lo que el calibrador de procesos diseñado también debe ser lo más compacto y transportable posible, alimentándose por medio de una batería que permita unas horas de funcionamiento aceptables y disponiendo de un conector para cargarla. Además, deberá mostrar una señal de aviso por pantalla cuando el nivel de la batería sea bajo y sea necesario cargarla. El proceso de carga no debe influir en el comportamiento del calibrador, de forma que pueda ser posible utilizarlo mientras se encuentra conectado al cargador.

Por último, el prototipo de calibrador debe tener un diseño fácilmente replicable, de forma que cuando el resultado final sea satisfactorio, se puedan construir más instrumentos funcionales para su utilización en el ámbito académico.

4.2. Diseño Hardware

4.2.1. Arduino

Arduino es una plataforma electrónica de software y hardware libre. Sus placas PCB se basan en un microcontrolador ATMEL AVR (con excepciones como el Arduino DUE, que contiene un ARM) y están diseñadas para poder usarlas fácilmente en multitud de proyectos. Estas placas pueden ser usadas para controlar una gran variedad de sensores y actuadores, ya que pueden recibir y emitir diversas señales digitales y analógicas, así como comunicarse con otros sistemas complejos como procesadores o controladores. Tanto el lenguaje de programación de Arduino (basado en *Wiring*) como su entorno de programación IDE (basado en *Processing*) pueden utilizarse y descargarse de manera gratuita.

En este proyecto se ha utilizado una placa Arduino Mega 2560 (revisión 3). Está basada en un microcontrolador ATmega2560, tiene 54 pines de entrada/salida digital (de los cuales 15 pueden ser usados como salida PWM), 16 entradas analógicas, 4 puertos UART, un oscilador de cristal de cuarzo de 16MHz, una conexión USB, un conector jack para alimentación, un cabecero de pines ICSP y un botón de reset, además de todos los componentes necesarios para dar soporte al microcontrolador. El principal motivo de elegir esta placa es su número de entradas/salidas digitales, ya que para nuestro proyecto se han necesitado 22 pines, además de 1 entrada analógica y los pines de alimentación. Con un Arduino UNO, por ejemplo, no habría sido posible porque solo dispone de 14 entradas/salidas digitales. Adicionalmente, esta placa trabaja a una tensión de 5V, lo que coincide con la mayoría de componentes utilizados (el Arduino DUE, aunque tiene suficientes pines digitales, trabaja a 3.3V, por lo que el Mega 2560 es mejor opción).

Figura 4. Arduino Mega 2560

- **Alimentación**

El Arduino Mega 2560 se puede alimentar mediante conexión USB o con una fuente de alimentación externa. La forma de alimentación se elige automáticamente.

Si escogemos alimentarlo sin la conexión USB, podemos utilizar tanto un adaptador de corriente como una batería. El adaptador se puede conectar a través del jack de 2.1 mm, con el positivo en el centro, de que dispone la placa. Si alimentamos con batería podemos conectar los polos a los pines GND y Vin del conector POWER. Nuestro prototipo pretende ser portátil, por lo que se ha utilizado una batería y se ha conectado de la última manera descrita.

La tensión de alimentación que soporta la placa es de 6–20V, aunque la tensión recomendada es 7–12V, ya que menos de 7V podría suponer que el convertidor interno no llegue a suministrar los 5V a los que funcionan nominalmente los componentes de la placa. Así mismo, si suministramos más de 12V corremos el riesgo de que el convertidor se sobrecaliente y dañar la placa. En nuestro caso, la batería proporciona nominalmente 7.4V, por lo que entramos en el rango recomendado.

- **Entradas y Salidas**

Cada uno de los 54 pines digitales del Mega 2560 se puede usar tanto como entrada como salida, operan a 5V, y la corriente de salida/entrada recomendada es de 20mA, y la máxima 40mA. Tienen una resistencia interna de pull-up (desconectada por defecto) de 20-50 kΩ.

Adicionalmente, tenemos pines que además tienen funciones más específicas:

- Comunicación serie: tenemos 4 puertos de comunicación serie, que se usan para recibir (RX) y transmitir (TX) datos TTL. El puerto 0 lo forman los pines 0 (RX) y 1 (TX), que además son los que corresponden con la comunicación por USB; el puerto 1 son los pines 19 (RX) y 18 (TX); el puerto 2 los pines 17 (RX) y 16 (TX); y el puerto 3 los pines 15 (RX) y 14 (TX). De este tipo de comunicación solo se ha utilizado la comunicación por USB para cargar los programas en la placa.

- Interrupciones externas: son los pines 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2). Estos pines pueden configurarse como disparadores de una interrupción, ya sea por nivel bajo, flanco de subida, bajada o cambio. En este proyecto no hemos servido de las interrupciones 0, 1, 5, y 4 activadas por flanco de subida correspondiendo al funcionamiento de los botones F1-4 de selección de modo.

- Salida PWM: los pines 2 a 13 y 44 a 46 pueden emitir una señal PWM de 8 bits. Esta función no ha sido necesaria en el proyecto.

- Comunicación SPI: los pines 50 (MISO), 51 (MOSI), 52 (SCK) y 53 (SS) permiten la comunicación SPI. Además, en la placa tenemos un cabecero de pines ICSP que también permite este tipo de comunicación. La comunicación SPI ha sido el método seleccionado para comunicar el Mega 2560 con los principales integrados del proyecto, y se explicará más detenidamente en los apartados correspondientes.

- La placa trae por defecto un LED integrado conectado al pin digital 13. El LED se encuentra encendido cuando la salida del pin en HIGH, y apagado cuando es LOW. Esta función no tiene relevancia en este proyecto.

- Comunicación TWI: los pines 20 (SDA) y 21 (SCL) permiten la comunicación TWI. No se ha utilizado en este proyecto.

El Arduino Mega 2560 cuenta con 16 entradas analógicas, con una resolución de 10 bits. Por defecto el rango de medida es desde masa hasta 5V, aunque es posible cambiar el valor alto de este rango mediante el pin AREF, cuyo valor se convertiría en esa tensión máxima. Se ha utilizado la entrada analógica A2 para monitorizar el nivel de carga de la batería. Adicionalmente tenemos el pin de RESET que está conectada con el pin de reset del microcontrolador.

4.2.2. Generación de tensión

Una de las funciones principales de nuestro calibrador es la generación de una tensión controlada, ya sea para simular la señal de un termopar como para la generación de una tensión concreta para otro fin. Después de considerar diversas opciones para conseguir esta función, se optó por el circuito integrado DAC1220 de Texas Instruments. Este integrado es un convertidor digital/analógico de 20 bits de precisión. Se vale de la modulación delta-sigma para conseguir una respuesta lineal con un bajo consumo energético y en un encapsulado pequeño. Está diseñado para aplicaciones de control en lazo cerrado en procesos industriales que necesitan una gran precisión. También es ideal para aplicaciones de control remoto, instrumentos alimentados por batería y sistemas aislados.

Entre sus principales características encontramos:

- Precisión de 16 o 20 bits, pudiendo elegir entre ambos modos.
- Error lineal de ± 15 ppm del fondo de escala.
- Tensión de salida entre 0 y 2 x V_{ref}, siendo la V_{ref} recomendada 2.5V.
- Corriente de salida 0.5mA.
- Tiempo de establecimiento máximo (a ± 0.012%) de 2ms.
- Tensión de referencia de entrada 2.25-2.75V, siendo la típica de 2.5V.
- Tensión de alimentación 4.75-5.25V.
- Diferencia máxima entre AV_{DD} y DV_{DD} de ±0.3V.
- Impedancia de entrada de 100kΩ.
- Familia lógica CMOS, compatible con TTL.
- Disipación de potencia máxima 3.5mW, 0.45mW en modo reposo.
- Rango de temperatura soportado entre -40 y +85 °C.
- Sistema de auto-calibración.

El núcleo de funcionamiento del DAC1220 está compuesto por un filtro de interpolación y un modulador delta-sigma de segundo orden. La salida de este modulador se transmite a un filtro capacitivo conmutado de primer orden en serie con otro filtro de tiempo continuo de segundo orden, que es el que genera la tensión de salida.

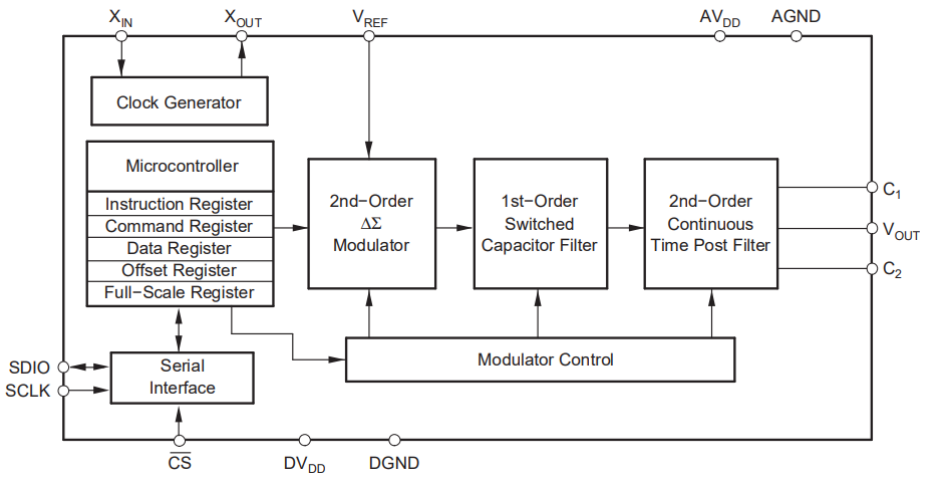


Figura 5. Diagrama de funcionamiento interno del DAC1220

El encapsulado del DAC1220 es de tipo SSOP-16, y los pines con los que cuenta son demasiado pequeños para poder soldarlos adecuadamente con los medios de los que dispone el laboratorio en el que se ha construido el prototipo que nos ocupa en este proyecto. Por esta razón se ha escogido utilizar la placa para prototipos DAC1220EVM de *Texas Instruments*, la cual nos permite trabajar con el integrado de una forma mucho más cómoda.

Figura 6. DAC1220EVM

La placa DAC1220EVM está diseñada de forma que puede ser utilizada para comprobar el funcionamiento del DAC1220, ya que incluye todos los componentes necesarios para su utilización y pruebas. También puede ejercer la función de un empaquetamiento más grande y más fácil de conectar para el integrado, ya que todos los pines del DAC1220 son accesibles a través de los pines de la placa.

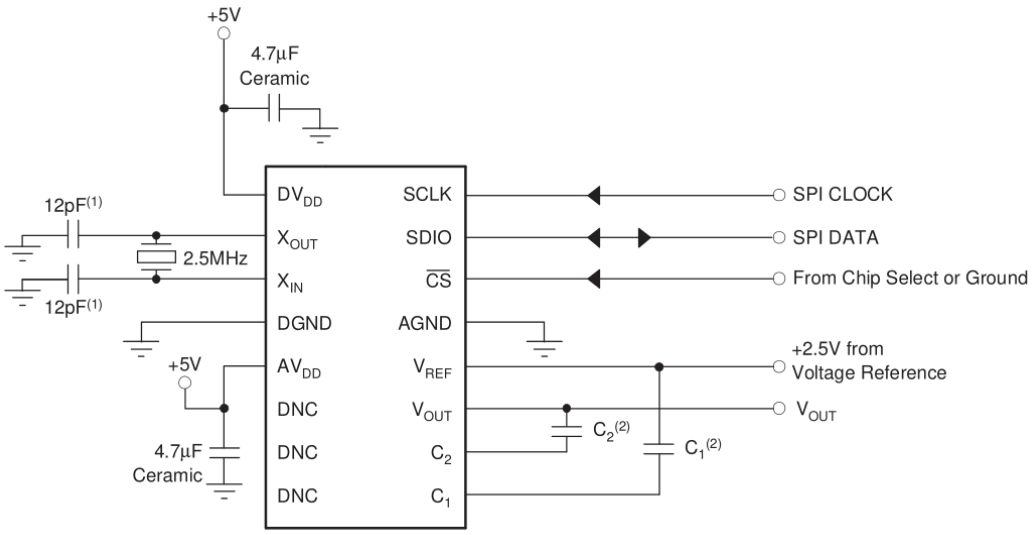


Figura 7. Esquema de las conexiones básicas del DAC1220

Como accesorios complementarios al DAC1220 encontramos:

- Los condensadores externos que necesita el filtro de tiempo continuo del integrado para funcionar. Los valores de estos condensadores son distintos dependiendo si queremos una resolución de 16 ó 20 bits. En este caso, la placa incorpora los valores correspondientes a la mayor resolución y no cuentan con un jumper o un conmutador para elegir este valor porque las entradas son muy sensibles, aunque estos condensadores podrían cambiarse fácilmente porque por ello se han montado sobre una superficie más grande de lo necesario.

Condensador	Modo 16 bits	Modo 20 bits
C_1	2.2nF	10nF
C_2	0.22nF	3.3nF

Tabla 1. Valores recomendados de los condensadores exteriores al DAC1220

- Un circuito de referencia de tensión basado en la referencia de tensión REF1004-2.5, de Texas Instruments. Este circuito puede proporcionar una señal de 2.5V precisa y de bajo ruido al pin V_{Ref}. Si queremos utilizar una referencia externa a la placa podemos utilizarla cambiando la posición del conmutador S1 de la posición OB a EXT.

- Un cristal oscilador de 2.4576MHz, y aunque la frecuencia típica requerida por el DAC1220 es de 2.5MHz, este cumple su función

perfectamente. Si quisiéramos utilizar un oscilador externo podríamos seleccionarlo cambiando el conmutador S1 de la posición XTAL a EXT.

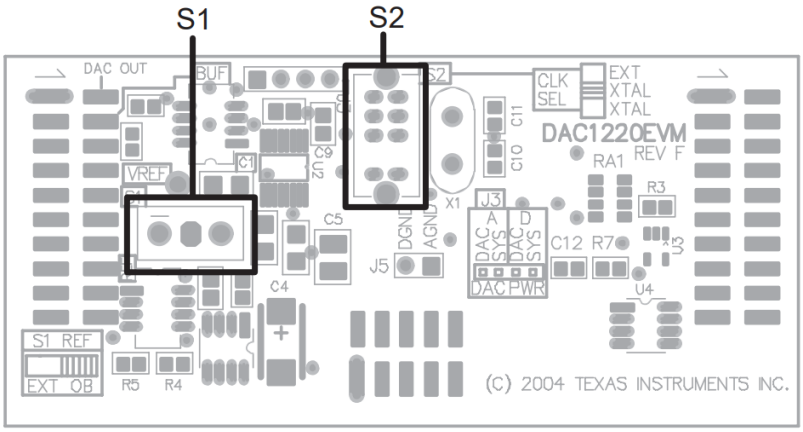


Figura 8. Situación de los conmutadores en el DAC1220EVM

- El DAC1220 tiene alimentaciones analógica y digital separadas, aunque ambas son a 5V, y la tensión de la alimentación digital nunca puede ser más de 300 mV mayor que la analógica. Por esto para la mayoría de los diseños, incluido éste, se utiliza una única fuente de alimentación. La PCB DAC1220EVM incorpora un *jumper* que conectado une ambas alimentaciones, pero que podemos quitar para suministrarles tensión por separado. Incluye también los condensadores de desacoplo sugeridos por el fabricante para conseguir un funcionamiento óptimo.

  Es adecuado comentar también que, aunque hay un *jumper* específico para unir o separar las masas analógica y digital, realmente los pines de masas del integrado están conectadas a la masa de la propia tarjeta, sin posibilidad de desconectarlas si no es desoldando. Esta es la principal razón por la cual se ha utilizado una única señal de alimentación, ya que, aunque tuviésemos dos circuitos diferenciados en nuestro proyecto para las señales analógicas y digitales, al conectar el DAC1220EVM las masas de ambos circuitos se conectarían, y dejaríamos de tener los circuitos separados.

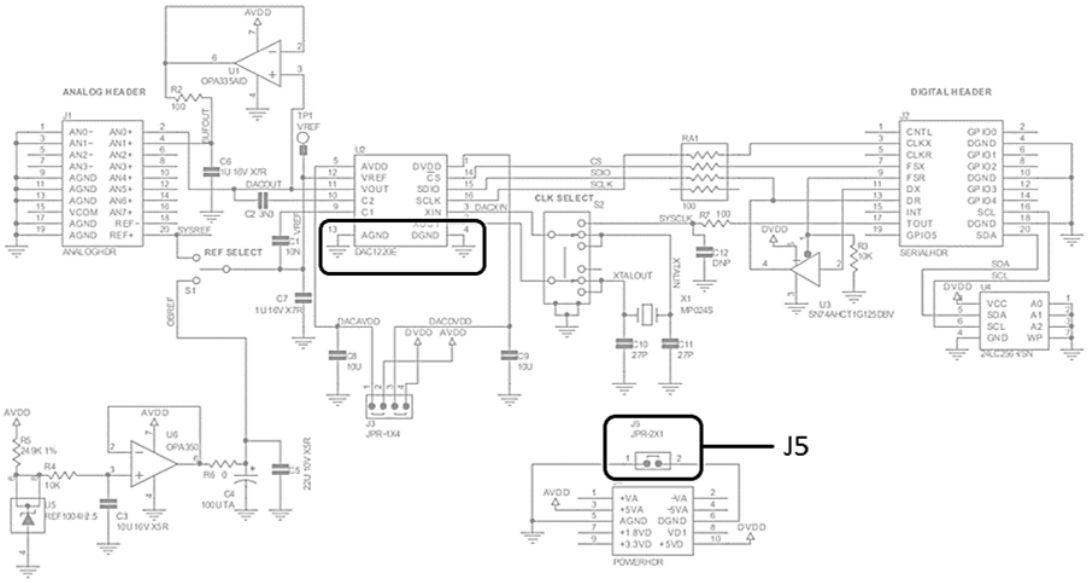


Figura 9. Esquema del DAC1220EVM. Se encuentran resaltados los pines de masa del DAC1220 y el jumper J5 de unión

- Un *buffer* amplificador que podemos utilizar por si la carga a la salida es tal que no sea suficiente con la nativa del DAC1220. Para este proyecto no ha sido necesario utilizarlo.

- Una memoria, para poder comunicarnos con el integrado por I2C, y un circuito con un *buffer* triestado por si en nuestro microcontrolador tenemos diferentes líneas para transmitir y recibir señales por comunicación SPI. Estos circuitos no ha sido necesario utilizarlos en este proyecto.

Una vez vistos los componentes adicionales al DAC1220 que nos proporciona la placa para prototipos, es el turno de explicar cómo se han realizado las conexiones con dicha placa. El DAC1220EVM cuenta con 3 cabeceros de pines, y aunque en total son 50 pines de conexión, son pocos los que tienen una función o nos son necesarios conectar.

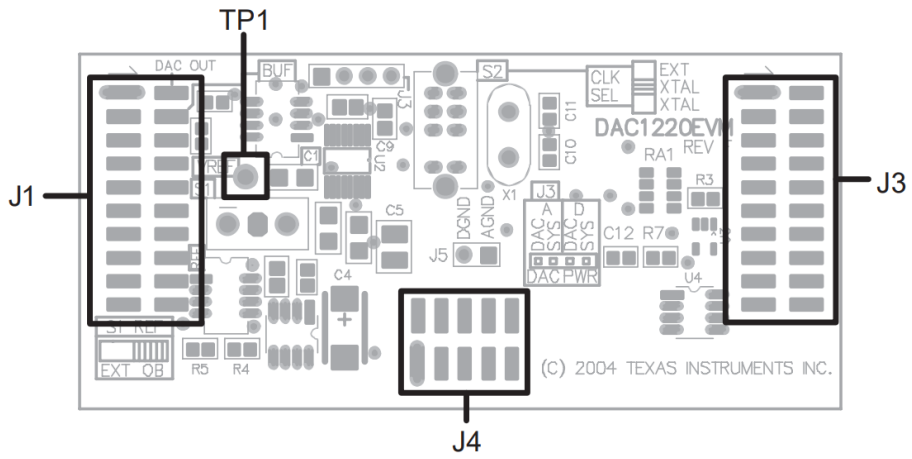


Figura 10. Situación de los conectores y el testpoint en el DAC1220EVM

El primer conector J1 es el conector en el que se encuentran las señales analógicas y la salida del DAC. En este conector se han utilizado los pines 1, que es la masa, y el pin 2, que es la salida del DAC1220.

El segundo conector J2 es en el que conectamos las entradas digitales para la comunicación serie. En este conector se han utilizado los pines 3 (SCLK), 7 (CS), 9 (SDIEN) y 13 (SDIO). La razón de utilizar estos pines y su función se explicará en el apartado posterior correspondiente a la comunicación SPI.

El último conector J4 es el correspondiente a la alimentación. Al utilizarse una única fuente tanto para la parte digital como la analógica, se han utilizado los pines 3 y 10 conectados a la alimentación de 5V, y el pin 5 a la masa general de la placa.

Además, dado que la referencia se puede acceder a través de un *testpoint*, se ha utilizado para la generación de tensión en la parte de simulación de termopares.

Recordemos por último que para nuestro proyecto los conmutadores deben estar en las posiciones correspondientes a utilizar tanto la referencia como el reloj integrados en la propia placa, así como todos los *jumpers* conectados porque solo se utiliza una fuente de alimentación.

El rango de tensión de salida que queremos alcanzar con este calibrador es de 0-10V, pero el DAC1220 solo alcanza una salida de 5V. Para solucionar esto se ha implementado un circuito amplificador. Este circuito es un circuito amplificador no inversor basado en un amplificador operacional.

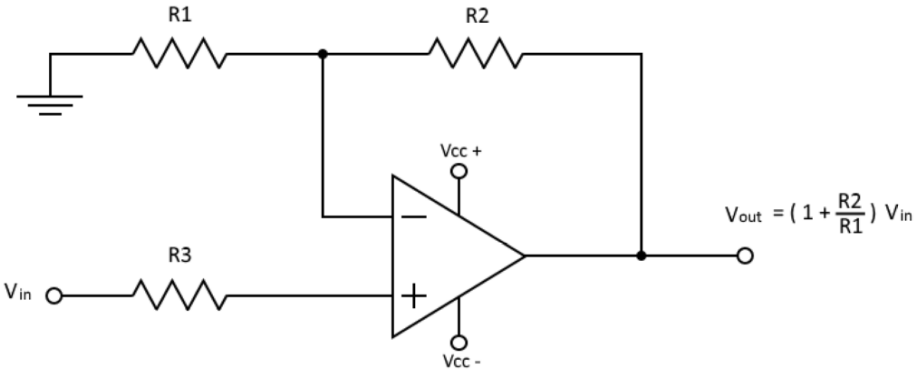


Figura 11. Circuito amplificador no inversor

Para este circuito se ha utilizado un LM258A, integrado que contiene 2 AO, aunque solo se ha utilizado uno de ellos. Para alimentarlo se ha empleado un convertidor DC/DC TMR-1, al que suministramos 5V de tensión desde el Arduino y nos proporciona una salida aislada de 12V. La salida positiva del DC/DC (12V) se ha conectado a la alimentación positiva del AO, y la referencia de la salida (GND) se ha conectado a la alimentación negativa del AO. Esto se puede hacer porque no necesitamos conseguir una tensión negativa, aunque el inconveniente es que la mínima salida real que podemos obtener es de 3.6mV.

El DAC en la práctica no llega a dar la salida especificada de 5V, sino que proporciona poco más de 4.6V. Por esto, para que la salida alcance 10V, la relación entre la salida y la entrada del amplificador no es 2, sino 2.5, motivo por el cual la resistencia de la realimentación es de 1500Ω y la resistencia R1 es de 1000Ω.

$$V_{out\ máx} = \left(1 + \frac{R2}{R1}\right) \cdot V_{in\ máx} = \left(1 + \frac{1500\ \Omega}{1000\ \Omega}\right) \cdot 4.6\ V = 11.5\ V \qquad (1)$$

A la hora de diseñar la PCB se cometió un error (relacionado con una versión del circuito anterior y el cambio en las huellas de los componentes) y la realimentación se realizó en la patilla no inversora del AO, lo cual se tuvo que solucionar cambiando la posición en la que se habían soldado las resistencias.

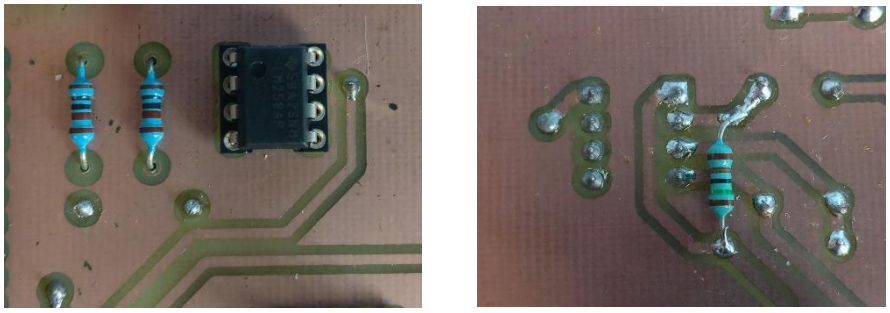


Figura 12. Imagen de la PCB. Detalle del circuito de AO. Cara superior (izquierda), cara inferior (derecha)

Para la simulación de termopares hace falta una tensión de mucha precisión y en un rango tanto positivo como negativo, y cercano a 0V. Por esto, para esta aplicación se ha decidido utilizar la referencia de 2.5V, accesible a través de un *testpoint* en la placa DAC1220EVM como referencia de tensión.

$$V_{termopar} = V_{DAC1220} - V_{ref\ tensión} \quad (2)$$

Con esto conseguimos que la salida que programemos en el DAC1220 siempre se encuentre en un valor intermedio al rango que puede suministrar, no se va a encontrar forzado y vamos a tener una precisión mayor; el ruido será menor porque podemos tener un conector específico para esta parte y situarlo en la placa más cerca de la salida; y vamos a poder simular valores de tensión negativos de forma que los termopares que simulemos serán como si tuviesen la parte de la unión fría a 0°C, tal y como vienen en las tablas de valor normalizadas. El conector de esta parte es un conector hembra en ángulo para soldadura en PCB, de 2 contactos y paso 5.08mm, de *RS Components.* El conector macho que encaja con éste se conecta con cables por medio de aprisionamiento por tornillo. Para evitar el ruido se han utilizado 2 cables cortos y trenzados, y en su otro extremo se ha colocado un conector de termopar tipo Cu (cobre), porque es el que no crea una tensión de compensación por ser ambos contactos del conector del mismo metal (los conectores específicos para un tipo de termopar tienen sus contactos de la misma dupla de metales que el termopar para que sirven, porque si son de metal distinto influye en la diferencia de potencial generada).

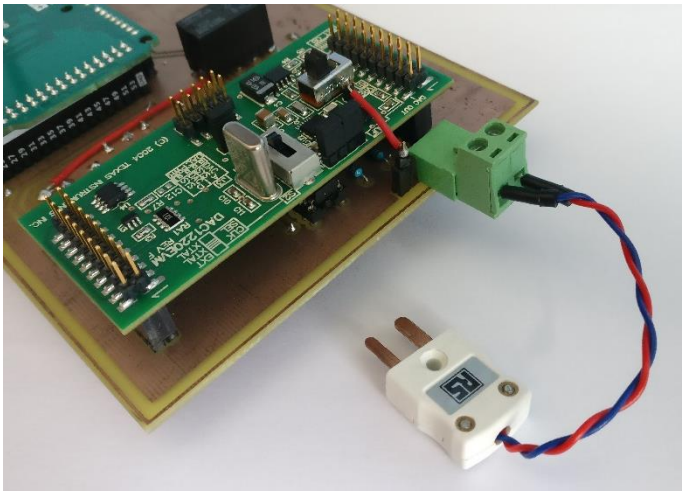


Figura 13. Imagen de la PCB. Detalle del DAC1220EVM, conexión de la referencia de tensión y el conector de salida de la señal de termopar

4.2.3. Generación de corriente

La siguiente función que vamos a abordar es la de generar una corriente controlada. Puesto que en la industria se utiliza el estándar para comunicaciones de 4-20mA, se ha decidido que ese sea el rango de corriente que es capaz de suministrar el calibrador de procesos. Para ello se ha utilizado el convertidor AD421 de *Analog Devices*. Este convertidor está diseñado para, en un circuito de lazo de alimentación, proveer una señal digital entre 4 y 20mA, lo que lo hace ideal para extender la resolución de un transmisor inteligente a bajo coste.

Entre sus principales características podemos destacar:

- Resolución de 16 bits.
- Error total máximo en la salida a 20mA de ±0.2% del fondo de escala.
- Tiempo de establecimiento típico para fondo de escala de 8ms.
- Modos de funcionamiento normal (4-20mA) y alarma (0-32mA).
- Cuenta con un regulador de tensión de 3, 3.3 ó 5V con el que se pueden alimentar otros dispositivos.
- La tensión de lazo que soporta es entre V_{CC}+2V (en nuestro caso 7V) y la tensión de ruptura del transistor de empobrecimiento que utilicemos (en nuestro caso 300V).
- Impedancia de salida de 25MΩ.
- Encapsulado tipo SOIC de 16 pines.
- Máxima disipación energética de 450mW.

- Rango de temperatura soportado entre -40 y +85 °C.

El AD421 está diseñado para ser utilizado en un circuito de alimentación por lazo. Esto es, que la salida de corriente se transmita por el mismo par de líneas que conducen la alimentación del lazo. Las funciones principales que cumple nuestro integrado son convertir las señales digitales de un microcontrolador a señales analógicas, amplificar la corriente que circula por el lazo, y regular la tensión de alimentación que se obtiene del lazo para alimentar otros componentes de la parte de control.

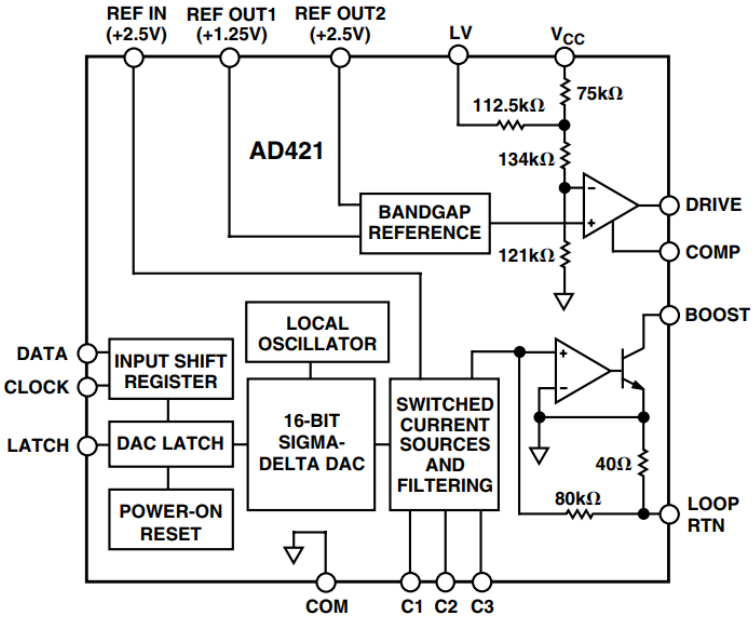


Figura 14. Diagrama de funcionamiento interno del AD421

- **Regulador de tensión**

  El circuito de regulación de tensión consiste en un amplificador operacional, una referencia de tensión y un transistor FET de empobrecimiento externo. Este circuito regula la tensión en el lazo para alimentar el propio AD421 y el resto de componentes de control que necesitemos.

  La tensión que podemos conseguir en V_{CC} depende de la señal en el pin LV y puede ser de 3, 3.3 ó 5V. Si conectamos LV a COM, la tensión regulada será de 5V. Si conectamos LV a V_{CC} la tensión será de 3V, y si la conexión con V_{CC} se hace incluyendo un condensador de 0.01µF entre medias, la tensión será de 3.3V. En la Figura 14 se puede observar el circuito correspondiente a 3.3V, y en la Figura 15 se muestra el esquema simplificado y las ecuaciones correspondientes al caso de 5V.

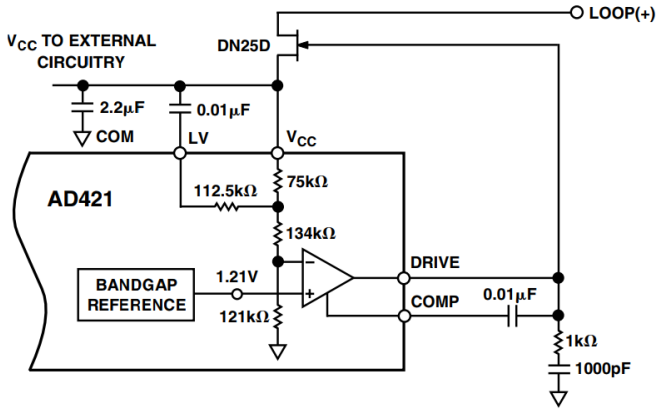
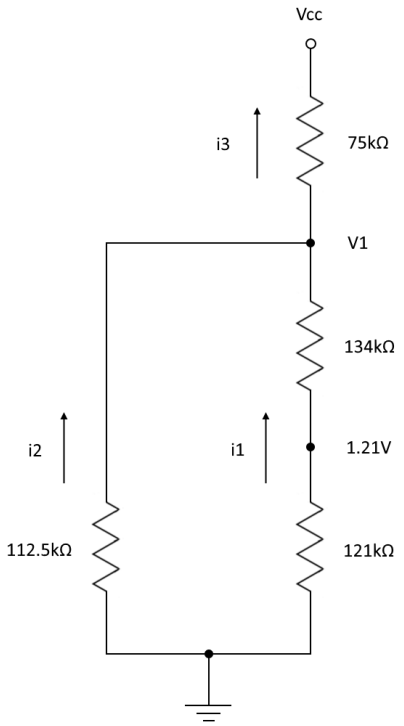


Figura 15. Esquema de regulador de tensión correspondiente a Vcc = 3.3V



$$i1 = \frac{1.21\ V}{121\ k\Omega} = 0.01\ mA \quad (3)$$

$$V1 = 0.01\ mA \cdot (121\ k\Omega + 134\ k\Omega) = 2.55\ V \quad (4)$$

$$i3 = \frac{2.55\ V}{\frac{(121\ k\Omega + 134\ k\Omega) \cdot 112.5\ k\Omega}{(121\ k\Omega + 134\ k\Omega) + 112.5\ k\Omega}} = 3.267\ mA \quad (5)$$

$$Vcc = 2.55\ V + 3.267\ mA \cdot 75\ k\Omega = 5\ V \quad (6)$$

Figura 16. Esquema y ecuaciones correspondientes al caso de regulación a 5V

El rango de tensiones que se pueden utilizar en el lazo viene determinado por las tensiones de ruptura y saturación del FET que utilicemos. Además, parámetro como la tensión de estrangulamiento $V_{gs(off)}$, la corriente de saturación I_{DSS} o la transconductancia deben ser elegidos de forma que la salida del AO a través del pin DRIVE pueda

controlar el punto de operación del FET. Las características que debe cumplir son las siguientes:

Característica	Requisitos mínimos	DN2530
Tipo de FET	Empobrecimiento, canal N	Empobrecimiento, canal N
I_{DDS}	24mA mín	200mA mín
BV_{DS}	(V_{LOOP} – V_{CC}) mín = 7V mín	300V mín
$V_{PIINCHOFF}$	V_{CC} máx = 5V máx	-3.5V máx
Disipación de potencia	24mA · (V_{LOOP} – V_{CC}) mín = 168mW	740mW

Tabla 2. Características del FET del lazo de alimentación

El FET sugerido en el *datasheet* del AD421 es el DN25D de *Supertex*, pero este transistor se encuentra descatalogado y no se ha podido conseguir. En su lugar se ha optado por el DN2530 de *Microchip*, que cuenta con unas características muy similares al recomendado.

Adicionalmente tenemos que incluir componentes externos para compensar el regulador y asegurar la estabilidad en la operación. Para estabilizar la regulación del lazo es necesario un condensador de 2.2µF entre V_{CC} y COM; un condensador de 0.01µF entre COMP y DRIVE, y una resistencia de 1kΩ en serie con un condensador de 1000pF entre DRIVE y COM.

- **Sección de conversión**

  El AD421 contiene un DAC sigma-delta de 16 bits para convertir la información digital recibida en una corriente. El DAC consiste en un modulador de segundo orden seguido de un filtro de tiempo continuo. Los bits discretos del modulador controlan una fuente de corriente conmutada. Esta fuente se filtra después por un filtro R-C de tres secciones. Las resistencias de este filtro están contenidas dentro del mismo integrado, pero los condensadores no, y hay que conectarlos entre los pines C1, C2 y C3, y COM. Los valores recomendados de estos condensadores son C1 = 0.01µF, C2 = 0.01µF, y C3 = 3.3nF. En este proyecto se ha utilizado en vez de este último uno de 2.2nF por ser el valor más aproximado de entre los que disponía en el laboratorio.

- Amplificador de corriente

  La corriente de salida del DAC controla una segunda sección compuesta por un AO y un transistor NPN, que actúa como un amplificador de corriente para estableces la corriente que circula por el pin LOOP RTN.

  Una resistencia de 80kΩ conectada entre la salida del DAC y la parte de retorno del lazo se utiliza como una resistencia de muestreo para determinar la corriente. La corriente de base del NPN controla la tensión que cae en la resistencia de 40Ω para igualarla a la que cae en la de 80kΩ.
  El pin BOOST se conecta normalmente al pin V_{CC}. Como el código de entrada del DAC varía desde solo ceros hasta escala completa, la salida de corriente del transistor NPN, y por consiguiente la corriente del lazo, variarán entre 4 y 20mA. Con los pines BOOST y V_{CC} conectados, el FET externo debe suministrar también todo el rango de corriente (4-20mA).
  Esta amplificación de corriente se realiza a partir de la tensión en BOOST, la cual en el caso de este proyecto son 5V, por lo que la carga máxima que puede soportar la salida es de 250 Ω como se muestra en la ecuación (7).

$$R_{OUT,máx} = \frac{V_{BOOST}}{I_{LOOP,máx}} = \frac{5V}{20mA} = 250\ \Omega \qquad (7)$$

- Sección de referencia

  El AD421 incluye una referencia de banda de 1.21V que se utiliza en la parte de regulación de la tensión de lazo. Otra referencia del mismo tipo se aprovecha para generar dos referencias de tensión para uso externo al integrado. En el pin REF OUT 1 tenemos una referencia de 1.25V y en el pin REF OUT 2 una de 2.5V. Ambas pueden proveer hasta 0.5mA de corriente. Dado que el propio AD421 necesita una referencia de 2.5V, podemos utilizar la del pin REF OUT 2 conectándolo a el pin REF IN.

  Cuando utilizamos alguna de estas dos referencias necesitamos utilizar condensadores de 4.7µF entre la referencia utilizada y COM para compensar y asegurar la estabilidad de la tensión de referencia. Si no se utiliza alguna basta con dejar el pin sin conectar.

- Lazo de salida

  La función que nosotros queremos conseguir es la de producir una corriente controlada en la salida del calibrador. El AD421 no realiza esta función como si de una fuente de corriente se tratase, sino que precisa de un lazo con alimentación, y regula la corriente que circula por dicho lazo.

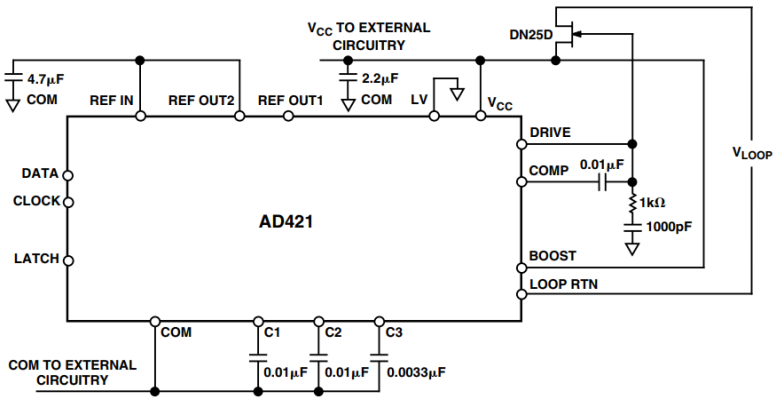


Figura 17. Esquema de las conexiones básicas del AD421

Para alimentar el lazo de salida se ha utilizado un convertidor DC/DC TMR-1 de 12V de salida, un modelo igual al utilizado en la parte de la generación de tensión. La corriente que circula por el lazo la medimos entre el pin LOOP del AD421 (borne más positivo de corriente) y la masa del convertidor (borne más negativo). El FET del lazo soporta hasta 300V, y dado que en el circuito implementado caen 7V (12V de la alimentación del lazo menos los 5V que utiliza el AD421 como alimentación), en los bornes de salida se podría conectar una tensión de hasta 293V (polo positivo en el borne conectado a la masa del DC/DC para que las tensiones se sumen), aunque aumentar la tensión del lazo no aumenta la carga máxima que se puede conectar, ya que la amplificación de corriente se genera a partir de los 5V a los que se alimenta el integrado).

4.2.4. Relé de salida

Una vez descritos dos de los circuitos principales del calibrador, el siguiente paso es decidir cómo vamos a poder acceder a las salidas de ambos circuitos desde el exterior del calibrador de procesos. Para tener los mínimos conectores posibles se ha utilizado un relé G5V-2-H1 de dos polos y alta sensibilidad. Este relé tiene una resistencia en los contactos muy pequeña (máxima de 100mΩ)

por lo que la caída de tensión que puede sufrir nuestra señal por atravesar el relé es mínima. Además, la tensión soportada en los contactos es de 125V y la corriente de 2A, por lo que puede soportar perfectamente las salidas de nuestro calibrador.

Los contactos de la bobina del relé se han conectado entre el pin 46 del Arduino y masa, de forma que mientras no enviemos tensión por ese pin los polos de relé se encontrarán en una posición, y si enviamos 5V cambiarán los polos de posición. Cuando dejemos de enviar tensión volverán a su estado inicial. Se ha decidido manejar directamente la bobina del relé desde un pin de Arduino y no mediante un transistor por simpleza en la implementación del circuito.

La idea original de implementación de contactos de salida del calibrador era tener 3 contactos: uno específico para la salida de corriente, otro para la salida de tensión y una masa común. El circuito que corresponde a este diseño es el que se implementó en la PCB.

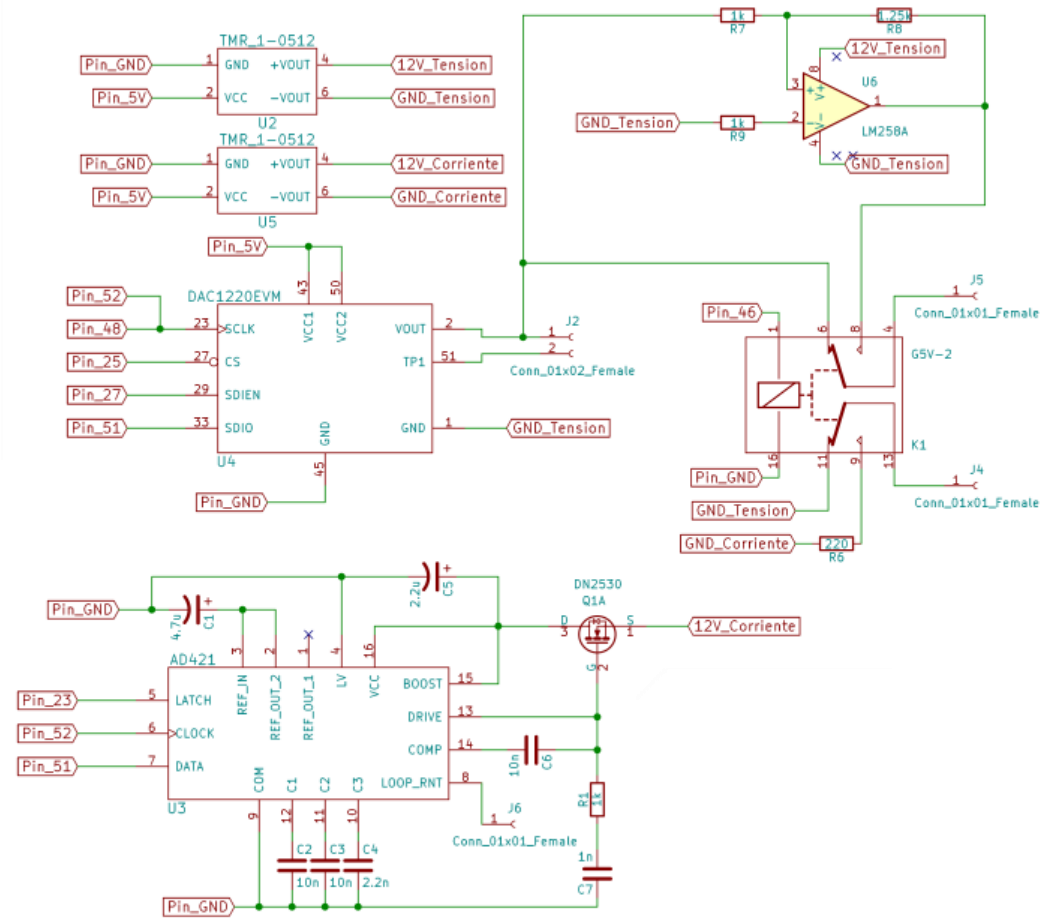


Figura 18. Esquema implementado en la PCB. Secciones de generación de tensión, corriente y relé de salida

Dado que el relé tiene dos polos, el primero se utilizó para decidir, en el contacto de tensión, de donde obteníamos dicha tensión: esto es, si directamente del DAC1220, en un rango entre 0-4.5V y con mayor precisión, o del circuito amplificador en un rango hasta 10V y con menos resolución.

El segundo polo se usaba para distinguir que referencia se utilizaba en la salida. El circuito de generación de tensión proporciona un voltaje referido a la masa del DC/DC que alimenta el AO. Puesto que la salida cuando se obtiene directamente del DAC1220 está referida al COM del integrado, y la tensión de ambos circuitos debe estar referida desde la misma masa, se han unido ambas masas, coincidiendo además con la del Arduino y utilizándose como la general de la PCB.

La segunda referencia que podemos elegir en este polo no es una masa, sino un punto del lazo del circuito de generación de corriente. Este lazo no puede compartir un punto con la masa de la placa porque no estaría aislado y no se podría controlar la corriente que circula por él. Por esto se ha tenido que utilizar un DC/DC distinto al del circuito de generación de corriente, aunque por potencia suministrada (1W) debería bastar con uno solo.

El problema de este planteamiento es que, aunque el relé tiene dos polos aislados, también tiene dos estados, de forma que ambos se encuentran o bien en la posición de reposo o bien activados. Esto conlleva que no podamos conseguir los 3 estados que necesitamos conseguir: no podemos cambiar entre la salida de tensión sin amplificar y amplificada sin cambiar a la vez de referencia en el conector de la masa.

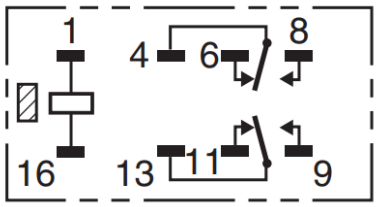


Figura 19. Esquema de los contactos del relé

La solución realizada es omitir la salida proveniente directamente desde el DAC1220 y que todo el rango de tensión se obtenga a través del circuito amplificador. El objetivo inicial de mantener más resolución en los valores más bajos de tensión se descarta y se decide que todo el rango tenga la misma precisión de 0.1mV. Este cambio libera uno de los polos del relé, por lo que se ha aprovechado para cambiar entre la salida de tensión y la de corriente, con lo que se ha eliminado un conector, dejando solo dos, y consiguiendo un resultado final más intuitivo y elegante.

Además en esta sección se ha eliminado la resistencia de carga que se tenía inicialmente en la salida de la generación de corriente, con lo que se aumenta la carga máxima externa que se puede conseguir.

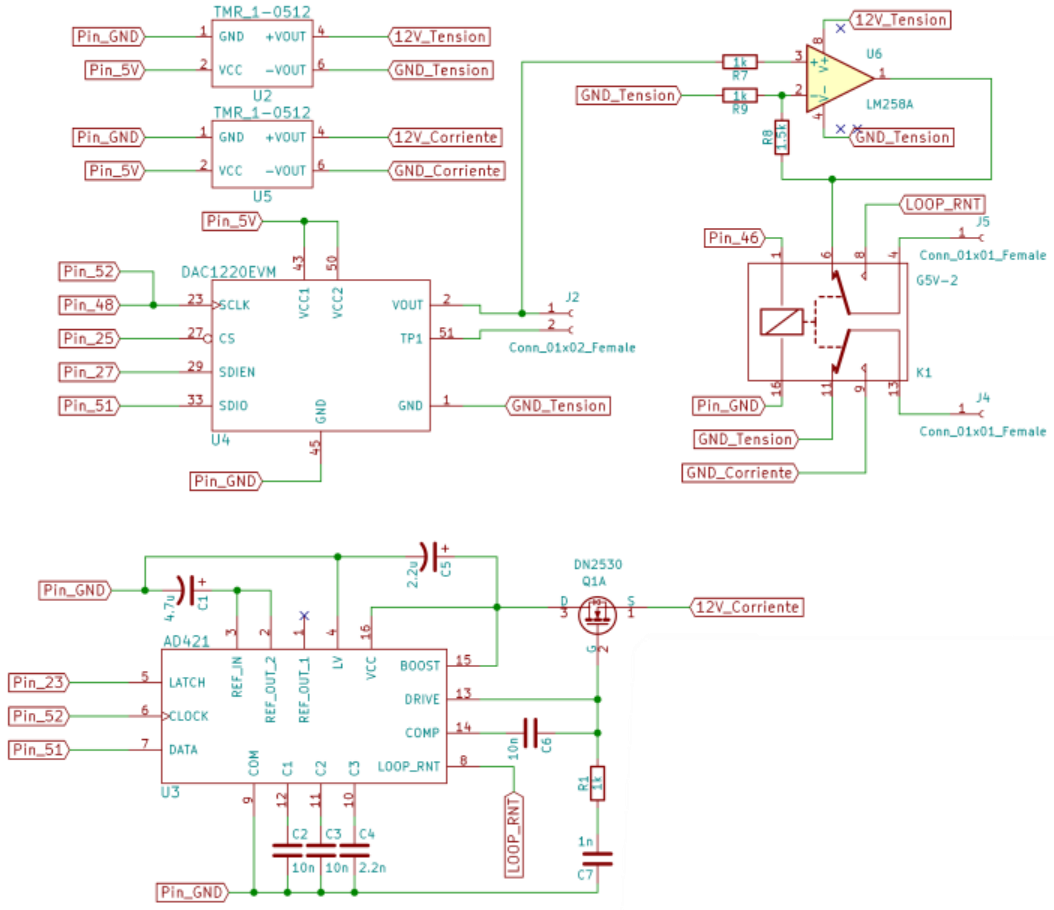


Figura 20. Esquema con las correcciones implementado en la PCB. Secciones de generación de tensión, corriente y relé de salida

Los conectores elegidos son dos conectores hembra SLB4-F de 4mm compatibles con los conectores macho de los cables utilizados típicamente en instrumentos eléctricos de medición como polímetros. Contamos con un conector rojo para las salidas positivas de tensión o corriente, y uno negro para las referencias. Estos conectores están soldados a la placa a través de dos cables, ya que si estuviesen directamente en la placa no podríamos acceder a ellos desde fuera de la carcasa con la que protegeremos la PCB y los componentes. Estos cables se han obtenido de una fuente de alimentación, por lo que son más gruesos que el resto de conductores utilizados y ofrecen menos resistencia, por lo que tendremos menos caída de tensión desde la placa hasta los conectores externos.

Figura 21. Conector se salida de tensión

4.2.5. Generación de frecuencia

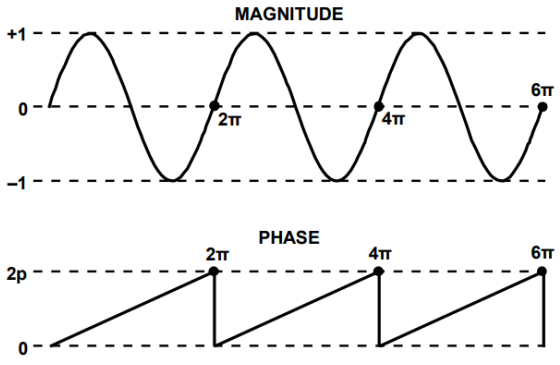
La última función que vamos a implementar es la generación de frecuencia. Esta función puede ser de gran utilidad porque es necesaria en muchas aplicaciones de medida, testeo o TDR (*Time Domain Reflectometry)* entre otras. Para ello se ha utilizado un generador de formas de onda programable, el circuito integrado AD9833. Este generador puede producir ondas senoidales, triangulares y cuadradas, pudiendo programarlas por comunicación SPI.

Las características más relevantes son:

- Resolución del DAC de salida de 10 bits.
- Tensión de salida entre 38mV y 0.65V.
- SNR (relación entre la señal y el ruido) típica de 60dB.
- THD (distorsión armónica total) típica de -66dBc.
- Tensión de alimentación de 2.3V a 5.5V, aunque la tensión de funcionamiento del integrado es de 2.5V.
- Corriente de alimentación máxima de 5.5mA.
- Encapsulado MSOP de 10 pines.
- Rango de temperatura soportado entre -40 y +105°C.

El AD9833 es un circuito integrado de síntesis digital directa (DDS). Puede crear ondas senoidales de hasta 12.5MHz, en conexión con un reloj de referencia y condensadores de desacoplo. Además, puede programarse para generar patrones simples o complejos de modulación (aunque esta función no se ha integrado en este proyecto).

El funcionamiento del AD9833 se basa en que la frecuencia de una onda senoidal puede expresarse como función de un intervalo de tiempo (o una frecuencia de muestreo) y el incremento de la fase de un punto de la onda en ese intervalo.



$$A = \sin(\omega t) \quad (8)$$

$$\omega = 2\pi f \quad (9)$$

$$\Delta Fase = \omega \cdot \Delta t = 2\pi f \cdot \Delta t \quad (10)$$

$$f = \Delta Fase \cdot \frac{1}{2\pi \cdot \Delta t} = \\ = \Delta Fase \cdot \frac{f_{CLK}}{2\pi} \quad (11)$$

Figura 22. Gráfica y ecuaciones que relacionan la frecuencia y el incremento de fase

Esta ecuación puede implementarse en un circuito en tres secciones: un oscilador controlado numéricamente (NCO), una memoria (SIN ROM), y un convertidor DAC.

En el AD9833 encontramos un NCO basado en un acumulador de fase de 28 bits. Este acumulador escala el rango que puede tomar una fase, 2π, a una palabra de 28 bits, por lo que el rango será de 2^{28} valores. La entrada del acumulador proviene de uno de los dos registros de frecuencia con los que cuenta el integrado, pudiéndose elegir de cuál de los dos tomaremos el valor de frecuencia que deseamos obtener. El NCO genera valores de fase de forma continua, lo que evita las discontinuidades cuando cambiamos de frecuencia.

A la salida del NCO tenemos un sumador que podemos utilizar para modular la fase y realizar una compensación si fuera necesario. Este sumador utiliza registros de 12 bits, cuyo valor se puede sumar a los bits más significativos de la salida del NCO.

El siguiente paso es convertir el valor de la fase en el valor de amplitud que corresponda en ese punto de la onda senoidal. Para ello tenemos una memoria SIN ROM que, mediante una tabla, convierte la fase en amplitud. La salida del NCO se trunca a 12 bits, porque si no sería necesaria una tabla de 2^{28} entradas y, además, al ser el DAC de 10 bits, tanta resolución sería poco práctica e innecesaria. Se utilizan 12 bits porque el error cometido al truncar sigue siendo menor que la resolución del DAC.

La salida final proviene de un convertidor DAC, que recibe el valor digital de la amplitud de la SIN ROM y lo transforma en un valor analógico correspondiente. El DAC genera típicamente una tensión de salida de 0.6V p-p. El integrado no necesita una resistencia de carga externa porque ya incluye una interna de 200Ω.

El encapsulado de este integrado es demasiado pequeño para soldarlo con los medios de que disponemos en el laboratorio. Por ello se ha utilizado un módulo que incluye, además del AD9833 soldado, el cristal de reloj de 25MHz y los condensadores de desacoplo que necesita el integrado como componentes externos

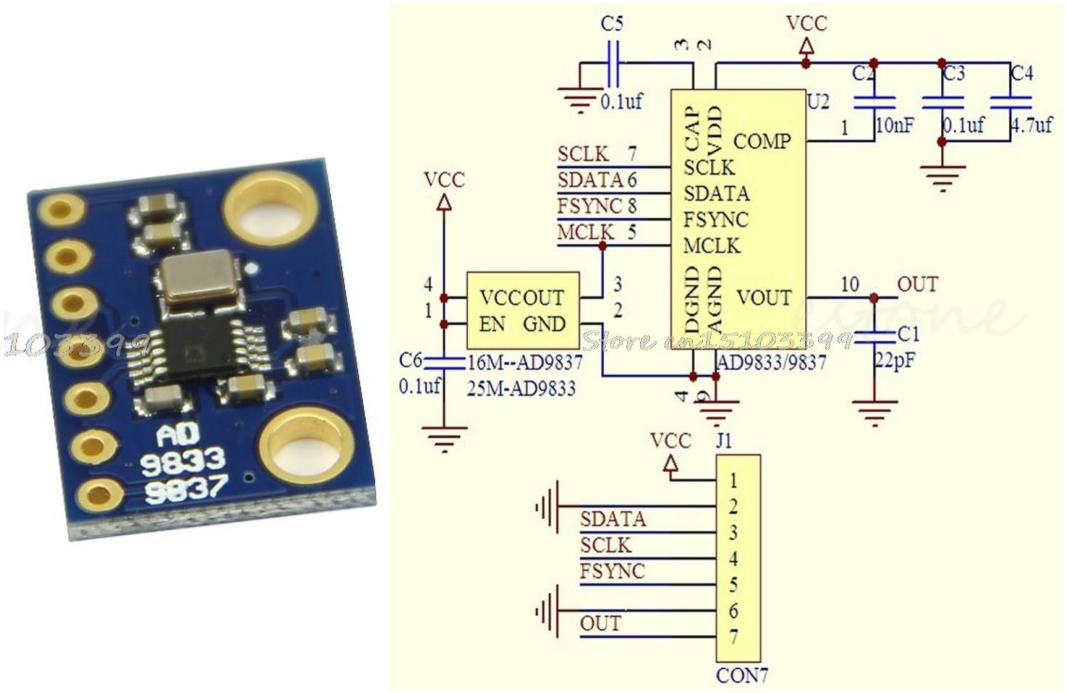


Figura 23. Módulo AD9833 (izquierda) y esquema interno de conexiones (derecha)

El módulo cuenta con un conector de 7 pines: uno de alimentación, al que suministramos 5V desde el Arduino; los 3 pines para la comunicación SPI; los dos pines de masa digital y analógica, que conectamos a la masa de la placa; y el pin de salida. En la placa se ha colocado el módulo en lo más cerca posible del borde de la placa en la parte inferior, de forma que quede lo más cerca de su conector de salida, que es específico para esta aplicación. Este conector es un conector hembra en ángulo, de 2 contactos y paso 5.08mm, de *RS Components*, el mismo modelo que el conector específico de la salida de termopares.

4.2.6. Display LCD

Comenzando con los elementos de la interfaz del calibrador de procesos, tenemos un display LCD *MC21605B6WD-SPTLY* del fabricante *Midas.* Este display cuenta con 2 filas y 16 columnas, con el fondo en color verde-amarillo, los caracteres en negro (monocromo), y retroiluminación por LED.

Figura 24. Display LCD MC21605B6WD-SPTLY de Midas

El display dispone de un cabecero de 16 pines (2x8). El pin 1 es el pin VSS, que conectamos a la masa de la PCB; el segundo es el pin VDD de alimentación, por el que suministramos 5V; el tercero es el pin V0 de control de ajuste, al que conectamos un potenciómetro conectado a su vez entre 5V y GND, y con el que podemos cambiar el contraste de los caracteres sobre el fondo del display; el cuarto es RS (selección de registro) y lo conectamos al pin 42 del Arduino; el quinto es el pin R/W, y como siempre va a recibir datos el display, lo conectamos a masa; el sexto es el pin E (*Enable),* que activa o desactiva el controlador del display. Este pin lo conectamos al pin 40 de Arduino. Los pines 7 a 14 son los pines de transmisión de datos. Para enviar datos al display se utilizan estos pines, de forma que para enviar una palabra de 8 bits se mandan todos a la vez en paralelo. Otra forma de transmisión que admite es utilizando solo los últimos 4 pines (bits 4-7) y enviando la palabra en dos operaciones consecutivas. Esta última forma es la que se ha implementado en este proyecto, porque el tiempo de envío no es crítico y ahorramos utilizar 4 pines del Arduino. Los últimos dos pines son el positivo y el negativo de la alimentación de los LED de retroiluminación.

Llegados a este punto cabe resaltar que, aunque este es el modelo de display que incluye el prototipo final de calibrador, no es sobre el que se trabajó para hacer el diseño. El primer display era un modelo *MC21605J6W-SPR-V2* de *Midas*. Ese modelo tenía la misma disposición de pines y se controlaba igual, pero no contaba con retroiluminación. Por esto en la huella de la PCB faltan los dos pines de alimentación (uno conectado a 5V y otro a la masa de la placa). Se ha solucionado conectando, sobre el propio display, el pin 15 (LED+) con el pin 2 (VDD) por deber ambos estar conectados a 5V; y el pin 16 (LED-) con el

pin 1 (VSS) por ir ambos a GND. Otro de los cambios que se ha debido hacer sobre el display es conectarle un potenciómetro para el ajuste de contraste, ya que en el primer display, al no tener retroiluminación, el contraste con el que mejor se apreciaban los caracteres era con el máximo (V0 conectado a masa).

Figura 25. Display utilizado en el prototipo

Por último, el display debe colocarse en la carcasa que proteja la PCB, de forma que se vea desde el exterior. Por esto se ha decidido conectarlo a la placa de calibrador mediante un cable IDE con un conector de 20 pines (10x2).

4.2.7. Encoder

Como método escogido para cambiar el valor de las distintas magnitudes de salida del calibrador tenemos un encoder rotativo incremental de la marca *ALPS,* aunque no se puede conocer el modelo exacto. Esto se debe a que, en afán por ahorrar en la construcción de calibrador y por ser un prototipo, se han reciclado materiales encontrados en el laboratorio que debieron pertenecer a otros equipos, pero que por no ser funcionales o ser obsoletos se decidió desmontar y aprovechar sus piezas. Ejemplos de componentes reciclados en este proyecto son el encoder, el display LCD, la botonera y la mayoría de cables.

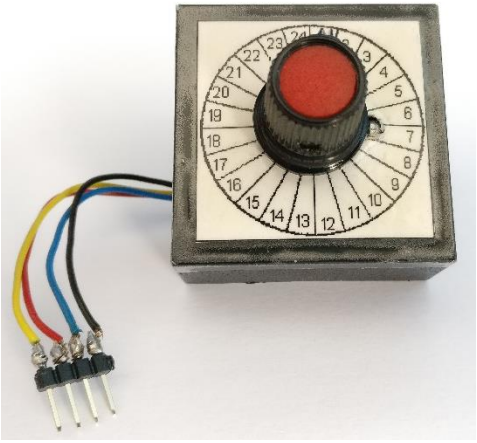


Figura 26. Encoder utilizado en el prototipo

El encoder cuenta con 4 pines, dos de alimentación (5V y masa) y las dos salidas A y B, que conectadas a los pines 11 y 12 del Arduino transmiten la posición relativa del encoder, la cual se compara en el controlador para determinar si la posición ha cambiado y en qué dirección. En este encoder encontramos que los dos pines A y B están conectados a GND mediante resistencias de pull-down.

El encoder, al igual que los demás elementos de la interfaz, se conecta a la PCB mediante un conector con cable para poder acceder a él desde el exterior de la carcasa.

4.2.8. Botonera

Debido a que el calibrador tiene diversas funciones, se ha incluido como método para elegir la que queremos utilizar en cada momento, una botonera. Es también reciclada de algún equipo antiguo, y al ser del fabricante *KEYMAT,* podía ser de algún tipo de cerradura.

Esta botonera cuanta con 4 pulsadores nombrados de F1 a F4 y que se han usado para cambiar, respectivamente, a los modos de generación de tensión, generación de corriente, simulación de termopar, y generación de frecuencia, además de accionándolo varias veces cambiar entre los submodos que cada uno ofrece.

Figura 27. Botonera utilizada en el prototipo

La conexión de esta botonera está compuesta de 5 pines, uno de alimentación (5V) que conecta con cada uno de los pulsadores, y los otros 4 son el otro contacto de cada pulsador. Cada uno de esto pulsadores se ha conectado a un pin de Arduino correspondiente a una interrupción (pines 2, 3, 19 y 18) y a masa mediante una resistencia de pull-down, de forma que mientras un pulsador esté en reposo, el Arduino recibirá en el pin correspondiente una señal baja (0V), y si se acciona llegará una señal alta.

4.2.9. Batería

El calibrador de procesos está diseñado para ser portátil, por lo que necesitamos una forma de alimentar los circuitos internos acorde a esta condición. Se ha decidido utilizar una batería de ion-litio modelo 18650CA-2S-3J del fabricante *BAK.* Es una batería de dos celdas, 2250 mAh, y tensión nominal de 7.4V. Además, incluye un circuito PCM de protección que mide la carga de la batería y corta la tensión tanto cuando la batería se está descargando y no puede suministrar más carga, como cuando se está cargando y llega a su límite. Según su *datasheet,* la temperatura ambiente recomendada cuando se efectúa la carga es de 0 – 45ºC (-20 a +60ºC en descarga), que al ser el rango más restrictivo de los materiales que componen el calibrador, será el que defina la temperatura de operación del prototipo.

La batería está compuesta de dos celdas de ion-litio en serie, cada una capaz de suministrar hasta 4.2V a carga completa, disminuyéndose este valor hasta

3V, valor en el que el circuito de protección corta el suministro de tensión. Esto se traduce en que el rango de voltaje que suministra la batería completa durante su ciclo de descarga comprende desde 8.4V hasta 6V.

Figura 28. Batería de ion-litio

Esta batería es la encargada de alimentar la placa Arduino Mega 2560, que admite una tensión recomendada entre 7 y 12V. Si la alimentamos con menos de 7 V se corre el riesgo de que no se lleguen a obtener los 5V a los que funcionan los componentes. Por esto, cuando la carga de la batería disminuye de forma que la tensión que proporciona es menor a 7V, se muestra por la pantalla del calibrador un aviso de que es necesaria la carga. Para que el Arduino conozca el estado de carga de la batería se ha conectado al pin analógico A2 la salida de un divisor de tensión, de forma que lee el voltaje correspondiente a la mitad de la tensión en los polos de la batería.

El conector de la batería es un conector JST de 3 pines: el rojo es la salida positiva, el negro la negativa, y el amarillo conecta con un termistor para comprobar la carga de la batería, pero este contacto no se ha utilizado. Para conectarlo a la placa se ha utilizado un cabecero de 3 pines macho.

El proceso de carga de las celdas de ion-litio consta de dos fases: la primera fase debe realizarse a corriente constante (a mitad del amperaje máximo), hasta que se alcanza la tensión de carga de la celda, 4.20V. En este punto se cambia a mantener la tensión constante mientras se disminuye la corriente de carga, y se mantiene de esta forma hasta que la carga de la celda se complete. En nuestro caso, como la batería está compuesta por dos celdas, la tensión constante a la que tenemos que alimentar es de 8.4V. Este proceso de carga permite alargar la vida útil de la batería.

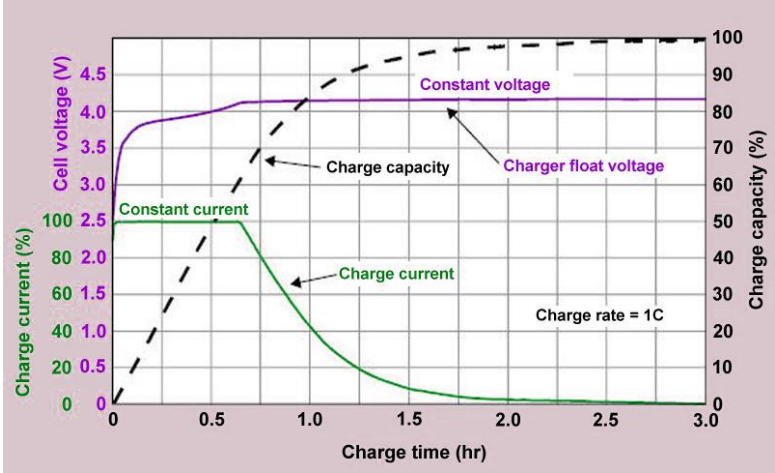


Figura 29. Ciclo de carga de una célula de ion-litio

El cargador escogido es el 2240 LI del fabricante *Mascot*. Este cargador incluye un transformador que admite 90-264V a 47-63Hz, por lo que permite conectarlo a las tomas de corriente normalizadas en Europa (230V, 50Hz). Este modelo permite cargar las baterías de ion-litio según el proceso CC/CV, suministrando 1.3A en la fase de corriente constante, y 8.4V en la de tensión constante. Además dispone de un LED en el transformador que se muestra en rojo mientas se esté cargando la batería, y cambia a verde cuando la carga está completa. Empíricamente se ha comprobado que el tiempo de carga de la batería, desde completamente descargada a completamente cargada, es inferior a 2 horas y media.

Para conectar el cargador a la batería se ha soldado un conector jack hembra de 2.1mm 2.5 mm en el extremo inferior de la placa, de forma que mediante un orificio en la carcasa se puede conectar el conector del cargador. Como elemento final del subcircuito de alimentación encontramos un interruptor deslizante monopolar de 2 vias (200mA, 12V) del fabricante *Knitter-Switc.* Este interruptor es el general del calibrador y está conectado entre el polo positivo de la batería y el pin Vin del Arduino. Está soldado a la placa mediante dos cables para poder acceder a él desde el exterior de la carcasa.

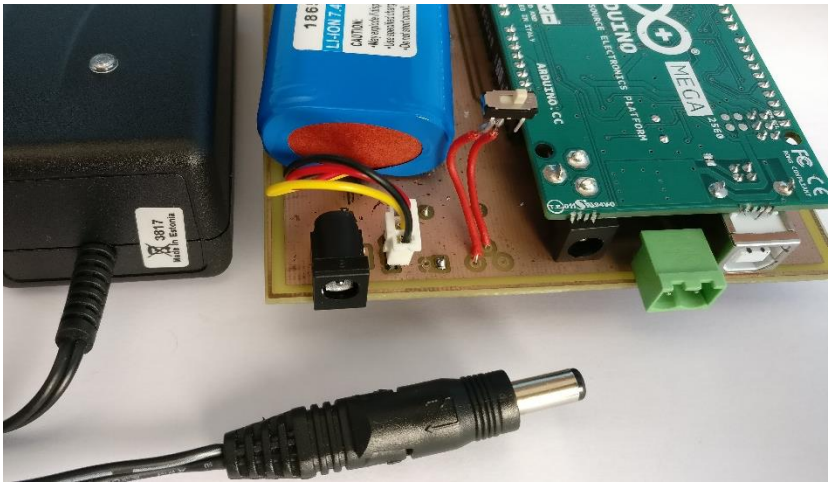


Figura 30. Imagen de la PCB. Detalle de los conectores del circuito de alimentación, la batería, el interruptor y el cargador

Para conocer el tiempo de uso que nos puede proporcionar la batería necesitamos conocer el consumo del calibrador. Para conocer el consumo teórico acudimos a los *datasheet* de los componentes.

	Valor típico	Valor máximo
Arduino	93mA	-
DAC1220	600µA	-
AD421	575µA	750µA
AD9833	4.5mA	5.5mA
Display LCD	20mA	30mA
Convertidor DC/DC	40mA	107.8mA
Relé	30mA	-
Total	228.675mA	375.45mA

Tabla 3. Consumo teórico de los componentes

Cabe aclarar algunos valores de la Tabla 3. El consumo del Arduino se refiere a un estado de reposo del mismo y sin componentes conectados. El consumo típico del convertidor DC/DC es sin carga y el máximo es suministrando el máximo de corriente, 83mA, con un rendimiento del 77%.

El consumo del calibrador varía según la utilidad que queramos aprovechar, ya que la función de generación de corriente es en la única en que consume el relé, además de que, a mayor valor de corriente en el lazo, mayor será el consumo del DC/DC.

Para comprobar el consumo real se ha medido la corriente que suministra la batería en los diferentes modos de funcionamiento. En los casos de generación de tensión, simulación de termopar y generación de frecuencia el consumo es

constante independientemente del valor de salida, mientras que en la generación de corriente el consumo sí que aumenta ante un valor de salida mayor.

Generación de tensión	188mA
Generación de corriente (lazo abierto)	221mA
Generación de corriente (4mA)	230mA
Generación de corriente (10mA)	238mA
Generación de corriente (20mA)	261mA
Simulación de termopar	192mA
Generación de frecuencia	217mA

Tabla 4. Consumo real del prototipo

De la Tabla 4 se puede concluir que el consumo mínimo del calibrador será en el modo de generación de tensión, con 188mA, y el mayor se obtendrá cuando se generen 20mA, con 261mA de consumo. Teniendo en cuenta que la capacidad de la batería son 2250mAh, la duración estimada de uso deberá ser entre 11 horas y 58 minutos, y 8 horas y 37 minutos, según las ecuaciones (12) y (13).

$$Duración\ máxima = \frac{2250mAh}{188mA} = 11.97h \quad (12)$$

$$Duración\ mínima = \frac{2250mAh}{261mA} = 8.62h \quad (13)$$

$$Duración\ 10mA = \frac{2250mAh}{238mA} = 9.45h \quad (14)$$

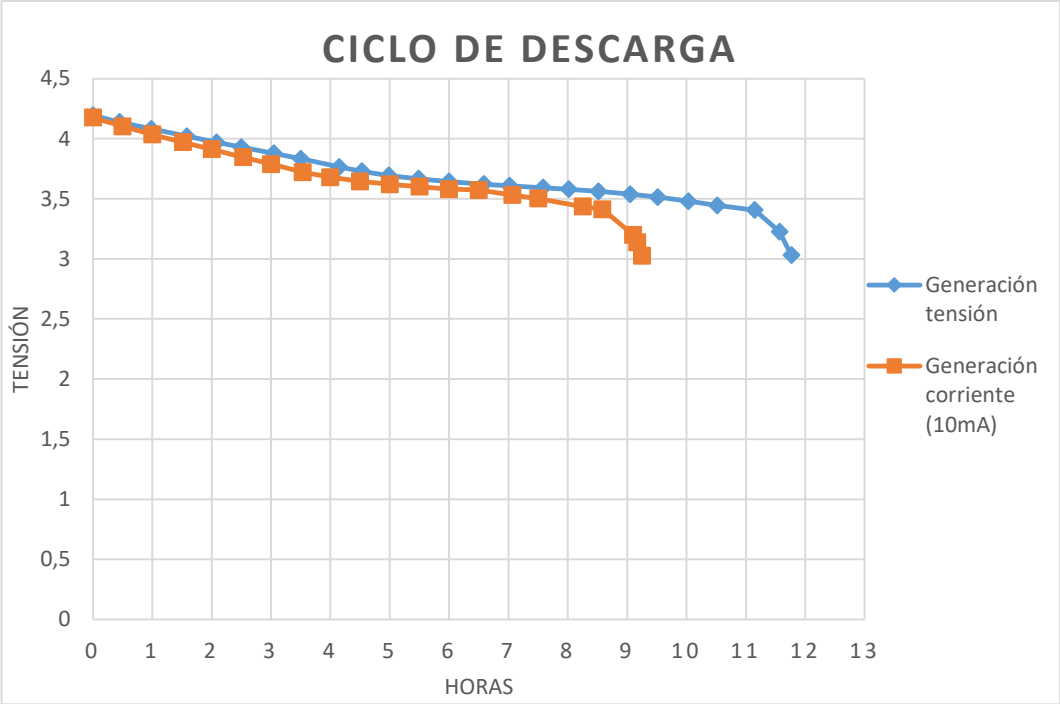


Figura 31. Gráfica del ciclo de descarga de la batería

Para comprobar que la duración de la batería coincide con la calculada se han realizado dos ciclos de descarga completa, uno en el modo de generación de tensión (5V) y otro en el de generación de corriente (10mA). Los tiempos conseguidos de funcionamiento han sido de 11 horas y 46 minutos en el modo de tensión, y 9 horas y 15 minutos en el de corriente, 12 minutos menos que los calculados en las ecuaciones (12) y (14) respectivamente. Esto se puede explicar atendiendo a que el circuito protector de la batería corta el suministro cuando la carga almacenada baja a niveles extremadamente bajos, por lo que no se aprovechan los 2250mAh al máximo.

4.2.10. Diseño de la PCB

El diseño del circuito esquemático y de la placa PCB del calibrador de procesos se ha realizado con el paquete de software para la automatización del diseño electrónico *KiCAD*. Este software es de código libre y multiplataforma. Cuenta con un editor para crear esquemáticos (*Eeschema*) y un entorno de diseño de circuitos impresos (*Pcbnew*). Se ha elegido este software porque, en comparación con otros entornos de edición gratuitos, permite crear placas PCB más grandes y complejas.

Pese a que este software cuenta con librerías en internet, la mayoría de los componentes utilizados en el calibrador no cuentan con un símbolo adecuado en el editor de esquemas. Además, una de las particularidades de KiCAD es que las librerías de componentes no cuentan con un símbolo esquemático y una huella asociada, sino que las librerías de símbolos y huellas están diferenciadas y es cuando se exporta el diseño del esquema a el editor de PCBs cuando se tiene que relacionar cada símbolo con una huella. Por esto, se han tenido que crear la mayoría de símbolos y huellas de los componentes.

En las siguientes figuras 32 y 33 se muestran el circuito completo implementado en la PCB del calibrador y el circuito resultante de realizar las correcciones realizadas sobre la propia placa respectivamente. Estos cambios incluyen la conexión de los pines de retroiluminación y el potenciómetro del display, la correcta realimentación del AO, el cambio de 3 bornes de salida de tensión y corriente a solo 2, la eliminación de la resistencia de carga en la salida de corriente y la conexión del polo positivo del circuito de alimentación directamente al pin Vin del Arduino. Ambos pueden encontrarse con mayor resolución en los anexos de esta memoria.

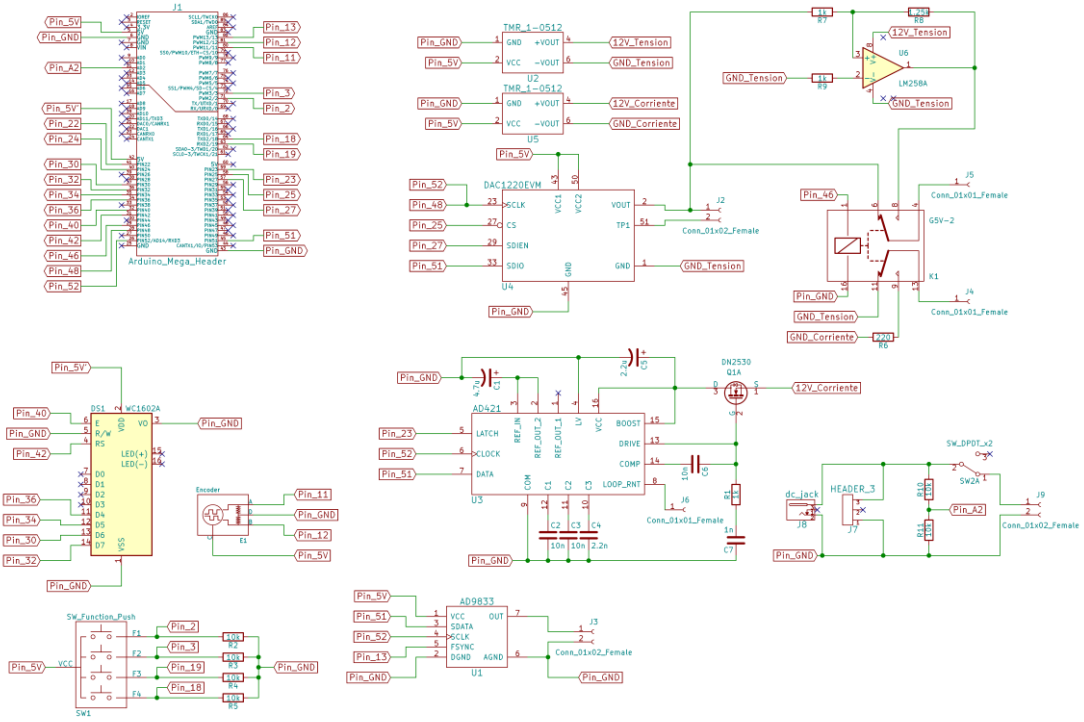


Figura 32. Esquema del prototipo. Versión primera

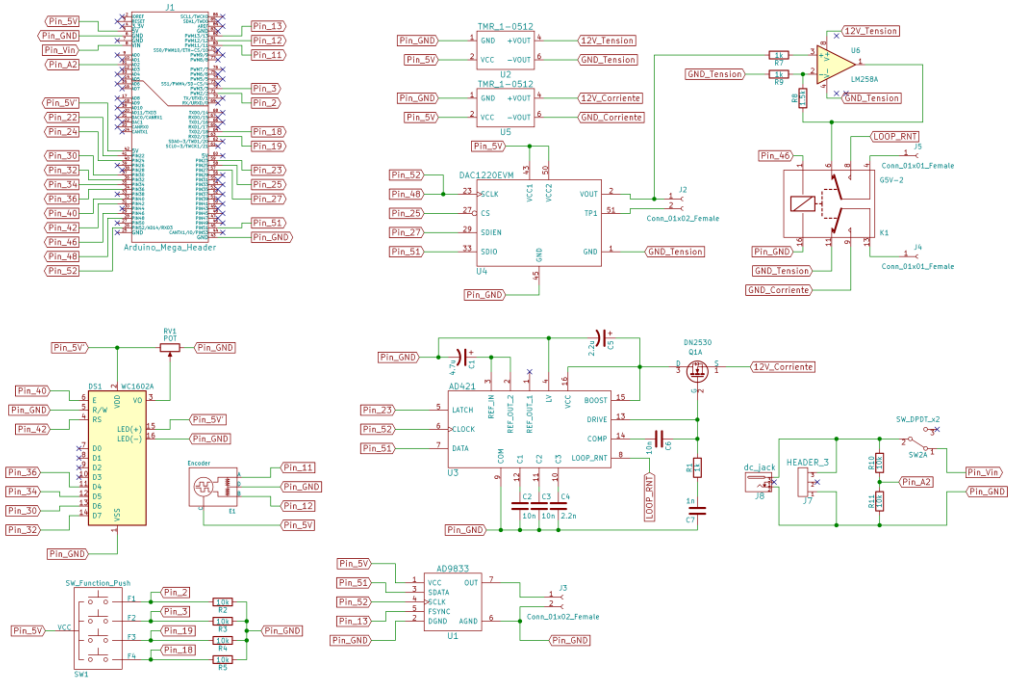


Figura 33. Esquema del prototipo. Versión segunda

Una vez completado el esquema y asociado cada símbolo a una huella a través de una *netlist* (así lo requiere el flujo de trabajo en KiCAD) se diseña la PCB. Uno de los motivos de utilizar KiCAD es que el tamaño de la placa no está limitado a una superficie máxima, por lo que se puede diseñar del tamaño necesario.

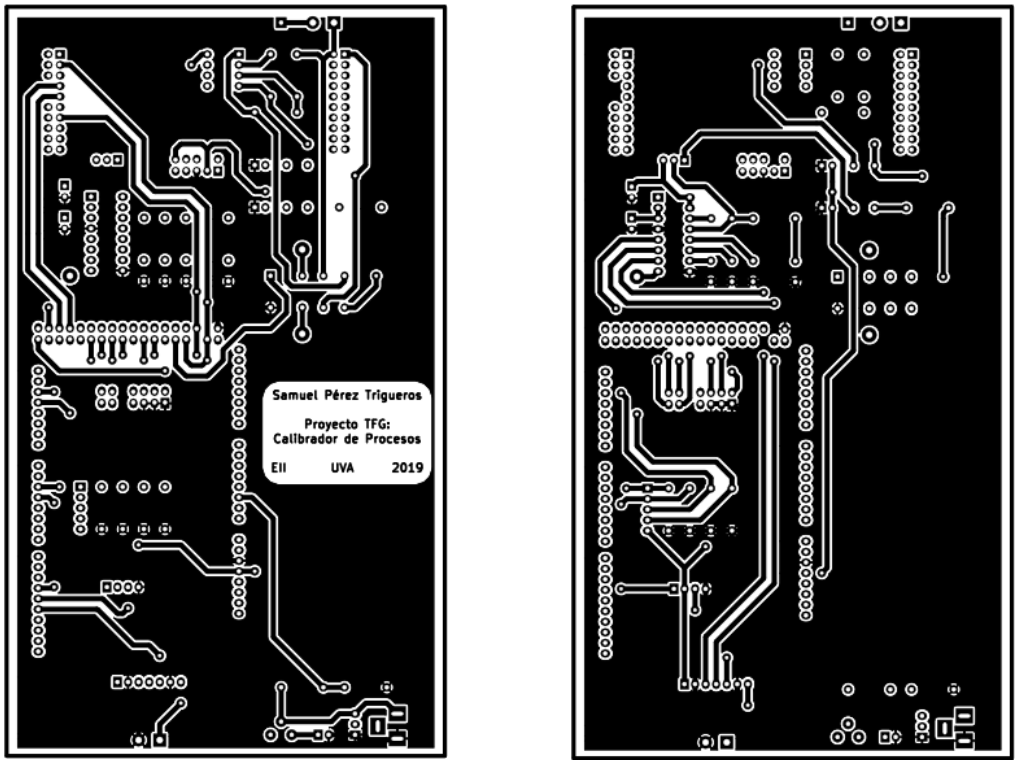


Figura 34. Footprints de la cara superior (izquierda) e inferior (derecha) de la PCB

La colocación de los componentes se ha realizado de forma que se optimice la superficie utilizada, se minimicen la longitud y número de pistas necesarias, y se mantengan agrupados los componentes de cada subcircuito. Además, se han omitido algunos pines de cabeceros de conexión por no estar utilizados y porque facilitaban la disposición de las pistas (dos pines del conector del display, dos del conector del Arduino, y tres del conector del DAC1220EVM).

Para minimizar el tamaño de la PCB se han dispuesto componentes por ambas caras de la placa. En la cara superior se sitúan el subcircuito del AD421, el módulo AD9833, los dos convertidores DC/DC, los conectores del display, el encoder, la botonera y los cables de los conectores de salida, así como las resistencias del divisor de tensión del circuito de medición de carga de la batería, las resistencias de pull-down de la botonera y la resistencia de realimentación del circuito amplificador no inversor. En la cara inferior se encuentran el resto de componentes soldados a la placa, esto es, las cabeceras de pines del Arduino Mega 2560 y del módulo DAC1220EVM, el relé de la etapa de salida, el circuito del AO (a excepción de la resistencia de realimentación, colocada en la cara superior para corregir un fallo), los conectores del circuito de la batería, y los conectores de las salidas de generación de frecuencia y simulación de termopares. Además, la conexión entre la placa y el DAC1220EVM lo aleja de la superficie de la PCB, por lo que ese hueco se ha

utilizado para optimizar el espacio y se ha colocado ahí el subcircuito del AO. Todos los circuitos integrados y componentes más sensibles se han insertado en la placa mediante zócalos por si fuera necesario quitarlos o cambiarlos por algún motivo, así como para protegerlos de las temperaturas que se alcanzan en el proceso de soldado.

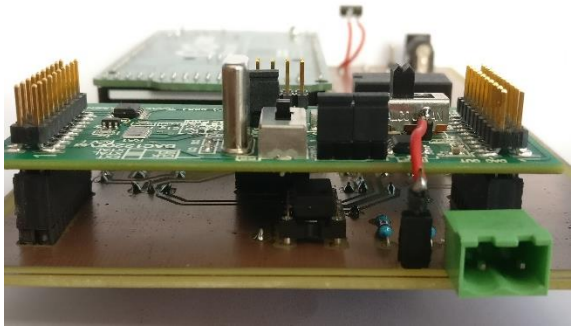


Figura 35. Imagen de la PCB. Detalle de la situación del AO debajo del DAC1220EVM

En la zona inferior derecha de la placa se ha dejado un espacio sin colocar pistas ni componentes porque es el espacio donde se situará la batería. De esta forma no se corre el riego de que algún roce por el contacto con la batería raye la superficie de cobre y se pierda la integridad de las pistas.

La PCB cuenta con un plano de masa en cada una de las dos caras, ambos conectados a la referencia del Arduino, que coincide con la referencia de todos los subcircuitos menos el del lazo de corriente. Estos planos favorecen a la disminución de ruidos en las señales que se transmiten por la PCB.

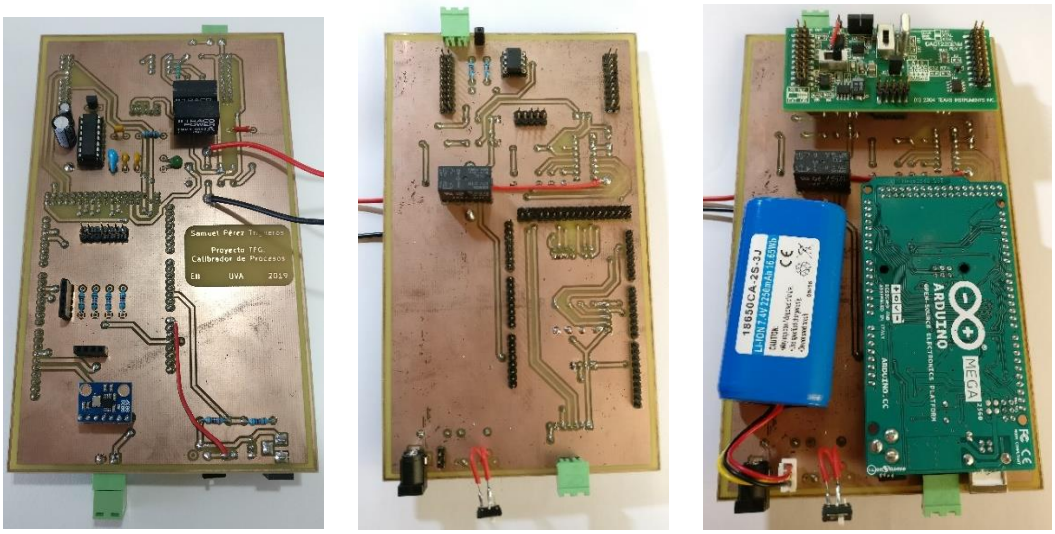


Figura 36. Imágenes de la PCB. Cara superior (izquierda), cara inferior de la placa (centro), cara inferior con el Arduino, DAC1220EVM y batería conectados (derecha)

Los conectores de salida de los circuitos de generación de frecuencia y simulación de termopares, así como el jack de carga, el conector USB tipo B del Arduino y el interruptor, se encuentran en los lados superior e inferior de la placa (los lados más cortos del rectángulo que forma la PCB). Esto se debe a que la placa está diseñada de forma que se sujete en la carcasa por medio de dos rieles laterales, motivo por el cual no se pueden colocar conectores en los dos lados más largos de la placa. Esta también es la razón de que la PCB no cuente con orificios para insertar tornillos de sujeción a la carcasa. Los elementos que componen la interfaz con el usuario (display, encoder y botonera) y los conectores de las salidas de tensión y corriente, se unen mediante cables a la PCB para poder colocarlos en la cara superior de la carcasa, de forma que sean accesibles desde el exterior.

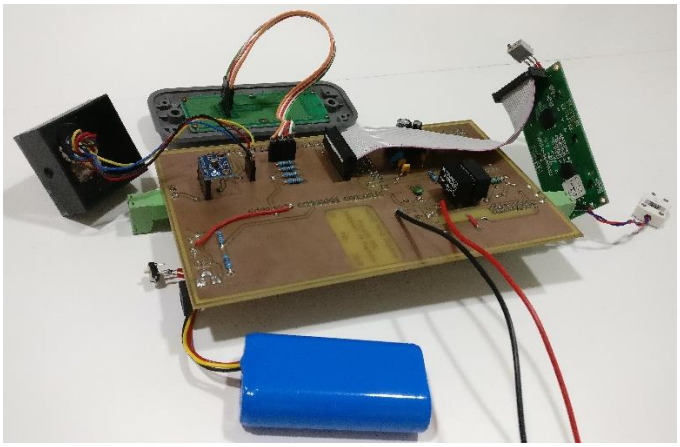


Figura 37. Imagen de la PCB con todos los componentes conectados

4.3. Diseño Software

4.3.1. Programar en Arduino

Las aplicaciones que se crean para controlar las placas Arduino se desarrollan en un entorno de programación propio (IDE). Este entorno se basa en otro entorno, también de código abierto, llamado *Proccesing.* El IDE de Arduino contiene un editor de texto para escribir el código de los programas (llamados en Arduino *sketches,* con extensión *.ino*), un área de mensajes, una consola de texto y una barra de herramientas para acceder a los menús y sus distintas funcionalidades. El código en que se desarrollan los *sketches* se basa en *Wiring* otra plataforma de código abierto. Este lenguaje se escribe en C/C++, y cuenta con una amplia variedad de librerías y funciones concretas para controlar las funciones de las placas Arduino.

Los *sketches* de Arduino responden por lo general a una estructura concreta y que consta de dos funciones: la función *setup()*, que se ejecuta solo una vez al inicio del programa y que se utiliza para configurar la placa; y la función *loop()*, que se ejecuta cíclicamente y que contiene el grueso de las funcionalidades del programa. Fuera de estas funciones podemos encontrar también otros apartados típicos de la mayoría de lenguajes de programación como la creación de variables, la definición de otras funciones, la llamada a interrupciones, etc.

4.3.2. Comunicación SPI

La comunicación entre los circuitos integrados y el Arduino en este proyecto se ha realizado mediante comunicación SPI (Interfaz Periférica en Serie). Este protocolo se basa en el sistema maestro/esclavo, y utiliza cuatro líneas, tres comunes a todos los dispositivos (MISO, MOSI, SCK), y una específica de cada uno (SS):

- MISO (*Master In Slave Out*): el esclavo envía datos al maestro.
- MOSI (*Master Out Slave In*): el maestro envía datos al esclavo.
- SCK (*Serial Clock*): pulsos de reloj para sincronizar la transmisión de datos
- SS (*Slave Select*): pin en cada esclavo cuya línea utiliza el maestro para seleccionar a cuál de todos es al que corresponde estar activo en ese momento.

En la placa Arduino Mega 2560 el pin MISO es el pin 50, el MOSI es el 51, el SCK el 52 y el SS el 53. Además, contamos con cabecero de pines ICSP que también permite este tipo de comunicación, aunque es este proyecto no se ha utilizado.

En el calibrador el Arduino actúa como maestro y los diferentes integrados como esclavos. En ningún caso necesitamos recibir información de los circuitos (tampoco están diseñados para ello), así que la línea MISO no se va a utilizar. Así mismo, para evitar posibles problemas relacionado con que el Arduino cambie a trabajar como esclavo, se define el pin SS como salida.

El protocolo SPI precisa de una serie de parámetros para poder funcionar correctamente. El primero de ellos es la velocidad a la que se va a transmitir la información, esto es, la frecuencia del reloj SCK. El Arduino incluye un reloj de 16 MHz, por lo que esa es la máxima frecuencia a la que podría funcionar. La frecuencia también depende de la que puedan soportar los esclavos, en este caso 245 KHz el DAC1220 (10 veces menor que la frecuencia de reloj del cristal de cuarzo que incluye la placa DAC1220EVM), 10MHz el AD421, y 40MHz el AD9833. En nuestra aplicación, la velocidad de comunicación no es un factor crítico, porque la velocidad de la respuesta solo influye en la rapidez con que el usuario recibe la salida especificada, y la percepción humana no es capaz de distinguir tiempos tan pequeños de tiempo. Por esto, la frecuencia que se ha escogido para trabajar es de 125 KHz, mucho más baja que la máxima que soportan los dispositivos, por lo que serán menos probables los problemas en la comunicación.

Otro de los parámetros a definir es, al tratarse de una comunicación serie, el orden en que se transmiten los bits. En este caso se ha escogido MSB (el bit más significativo primero) porque, aunque el DAC1220 admite tanto MSB como LSB, el AD421 y el AD9833 necesitan que la transmisión sea MSB. También hay que tener en cuenta que las operaciones de escritura son de palabras de 8 bits, por lo que si necesitamos enviar una palabra de mayor longitud hay que dividirla.

El tercer parámetro es el modo de funcionamiento del reloj, es decir, si el estado de reposo del reloj es cuando está en alto o bajo, o si los datos se cargan en flanco de subida o bajada. La comunicación SPI contempla cuatro modos.

Modo	Polaridad del reloj	Fase del reloj	Flanco de salida	Captura de datos
SPI_MODE0	0	0	Bajada	Subida
SPI-MODE1	0	1	Subida	Bajada
SPI_MODE2	1	0	Subida	Bajada
SPI_MODE3	1	1	Bajada	Subida

El DAC1220 utiliza el modo 1, el AD421 el modo 0 y el AD9833 el modo 2, por lo que cada vez que iniciamos una comunicación tenemos que elegir el modo correspondiente.

4.3.3. Programación del DAC1220

Los circuitos integrados que utilizamos en este calibrador pueden utilizar la comunicación SPI, pero internamente cada uno se programa de una forma distinta. El DAC1220 opera con comandos para acceder a 4 registros, y en base al valor de éstos se cambia de modo de funcionamiento y de salida. Los comandos son palabras de 8 bits: 1 para indicar si la operación siguiente es de lectura o escritura, 2 para el número de bytes que se deben leer a continuación del comando (de uno a tres) y 4 para la dirección del registro en el que se realiza la operación. Para cargar la información por el pin de datos, el pin CS debe permanecer en estado bajo.

Los registros del DAC1220 son: El registro DIR (Registro de Entrada de Datos), de 3 bytes, almacena el valor de la salida de tensión en los 20 bits más significativos; el registro CMR (Registro de Comandos), de 2 bytes, contiene la configuración del DAC1220, parámetros como el modo de funcionamiento (normal, reposo o calibración), la resolución (20 o 16 bits), el orden de los bits (MSB o LSB) o el formato de los números en binario (normal o complemento a 2) entre otros. Los dos registros restantes, OCR (Registro de Compensación de la Calibración) y FCR (Registro de Calibración de escala Completa), de 3 bytes cada uno, son registros para almacenar valores relacionados con la calibración de la tensión de salida del DAC1220.

El DAC1220 cuenta con tres modos de funcionamiento: el modo normal, es en el que la salida de tensión es la programada en el registro DIR; el modo reposo (*Sleep*), en el que la salida es alta impedancia, la circuitería interna se apaga y el consumo de corriente es mucho menor; y el modo de auto-calibración (*Self-Calibration*), que ejecuta una secuencia interna de calibración y modifica el valor de los registros OCR y FCR. Esta última función no se ha implementado porque el ajuste se ha realizado directamente en el programa.

Para inicializar correctamente el DAC1220 se debe realizar una secuencia de encendido, que consiste en un patrón cargado en el pin de reloj. El pin 52, puesto que es el pin de reloj de la comunicación SPI, no puede utilizarse para la comunicación y a la vez como salida digital, por lo que se ha conectado este pin con el 48, de forma que cuando hay que realizar el patrón, se declara el pin 48 como salida, se realiza, y después se declara como entrada (alta

impedancia), de forma que en ningún momento se produce conflicto con el pin 52.

Otra de las particularidades a la hora de conectar las líneas de comunicación SPI de debe a que el módulo DAC1220EVM contiene un circuito con un *buffer* triestado para permitir la comunicación bidireccional: mientras que la comunicación SPI utiliza una línea para transmitir del maestro al esclavo (MOSI) y otra para la comunicación en sentido contrario (MISO), el DAC1220 utiliza un solo pin tanto para enviar como para recibir. Este *buffer* permite dividir esa señal en las dos propias del protocolo SPI, de forma que la línea MOSI se conecta en el pin 11 (SDI) y el MISO en el 13 (SDIO). Cuando se envía datos al integrado se mantiene el pin 9 (SDIEN) a tensión baja, y cuando se reciben datos se cambia a alta. En el calibrador de procesos no hace falta recibir del DAC1220, por lo que se "puentea" este circuito, utilizando como entrada de datos el pin 13 y manteniendo el pin 9 en tensión alta, de forma que las conexiones equivalen a como si se realizasen directamente sobre el DAC1220.

4.3.4. Programación del AD421

El AD421 puede trabajar en dos modos: normal y alarma. Para modo normal el valor de corriente debe estar codificado en una palabra de hasta 16 bits en binario natural, con lo que conseguiremos una salida entre 4-20mA. Para el modo alarma, el código debe ser también en binario natural pero el número de bits debe ser de entre 17 y 31 bits, pudiéndose obtener una salida entre 0-32mA, aunque el rango recomendado es de 3.5-24mA. El AD421 conoce el modo de funcionamiento porque cuenta el número de pulsos de reloj entre dos pulsos de LATCH consecutivos. El LATCH se encuentra normalmente en estado bajo, por lo que es necesario un flanco de subida y uno de bajada para que se contabilice un pulso.

4.3.5. Programación del AD9833

El AD9833 tiene los 3 pines típicos de la configuración SPI, de forma que mientras el pin FSYNC está en estado bajo se pueden cargar palabras de 16 bits en el pin de datos. Este integrado también se programa escribiendo en registros, de forma que encontramos el Registro de Control, de 16 bits, y que, en función del valor de sus distintos bits, se escogen los modos de funcionamiento, de qué registro se toma la frecuencia de salida, en qué registro se deben escribir los datos que se envían al integrado, etc. También

encontramos dos registros de frecuencia, de 28 bits cada uno, para almacenar la que queremos que sea la salida del AD9833; y dos registros de fase, de 12 bits, que se utilizan para compensar la frecuencia de salida de la sección del NCO.

Cuando inicializamos el AD9833 debemos también ejecutar la función de *Reset*, que cambia los registros internos apropiados (no los registros de control, frecuencia ni fase) a 0, para suministrar una salida analógica intermedia en el rango de valores posibles.

4.3.6. Descripción del programa realizado

El programa de funcionamiento del calibrador sigue el esquema básico de los programas en Arduino: primero se declaran las librerías y variables; se inicializan en el *setup()* los pines y demás funciones necesarias (como las interrupciones); y en el *loop()* se define el cuerpo principal del programa, ya que es lo que se va a repetir constantemente mientras el calibrador se encuentre encendido. Por último, se definen las funciones necesarias que se van a utilizar en el programa.

- Declaración de variables

  El programa comienza con la declaración de las librerías utilizadas, que son *LiquidCrystal.h,* que facilita el control del display; *Encoder.h,* que asigna un valor a la posición del encoder, de forma que el resultado de cambiarlo de posición es sumar o restar el número de posiciones en el valor almacenado, lo que proporciona una forma de controlarlo más sencilla; y la librería *SPI.h*, para poder realizar la comunicación SPI.

  A continuación, se declaran los pines que intervienen en la comunicación del Arduino con los distintos componentes, las variables globales que se utilizarán en las diferentes funciones y secciones del programa, y se inicializan, según se especifica en las librerías del display y el encoder, los *objetos* que controlarán esos mismos componentes. Los nombres de las variables son lo más autoexplicativos posible, y además viene comentada su función en el código.

- *Setup*

  En la función *setup()* se inicializan los pines utilizados como entradas o salidas, se asocian las interrupciones con su pin y función correspondientes, se inicializa la comunicación con el display y se crean

los caracteres especiales que indican los submodos de la generación de frecuencia (onda senoidal, cuadrada o triangular), la flecha que indica cuantas cifras cambiamos por cada posición del encoder, y la alerta por batería baja.

- *Loop*

  La función *loop()* contiene la parte principal del programa. Puesto que el calibrador cuenta con cuatro modos de funcionamiento (tensión, corriente, termopares y frecuencia), este cuerpo principal del programa se ha estructurado en cuatro secciones correspondientes a cada uno de estos modos. La decisión de ejecutar una sección u otra de toma en función de cuatro variables (*ISRCounterFx)*, de forma que solo una puede encontrarse en un valor positivo a la vez (solo podemos tener un modo activo simultáneamente). El valor de estas variables es lo que se modifica accionando las interrupciones, de forma que cada interrupción cambia el valor de una variable concreta, cambiando el resto a -1.

  Cada una de estas cuatro secciones se divide a su vez en una estructura típica de Arduino: *setup* y *loop.* En el *setup* se prepara la pantalla escribiendo la unidad de medida que indica cada modo, y se inicializa el componente asociado correspondiente (DAC1220 en tensión y termopar, AD421 en corriente y AD9833 en frecuencia). El DAC1220 necesita algún segundo para inicializarse correctamente, por lo que mientras se lleva a cabo este proceso se muestra por pantalla un mensaje de “Cargando…”.

  En el *loop* de cada sección (implementado mediante un bucle *while*) se comprueban el nivel de batería y el submodo en que quiere trabajar el usuario. El submodo se elige volviendo a pulsar el botón asociado al modo en que se encuentra actualmente. De esta forma, para cambiar entre el primer submodo de funcionamiento (el actual) y el tercero, se tendría que accionar el pulsador asociado dos veces, una vez para cambiar al segundo y otra para llegar al tercero. El cambio de submodo es cíclico, de forma que pulsando el botón estando en el último submodo se cambia al primero. Para cambiar por completo de modo es necesario pulsar el botón de ese nuevo modo en concreto.

  En las secciones de tensión y corriente los submodos cambian la cifra que modificamos de la salida al variar el encoder; en termopares se cambia el tipo simulado entre J, K, E y T; y en frecuencia se cambia tanto el tipo de onda (senoidal, cuadrada, triangular) como el multiplicador (entre Hz y KHz).

Elegido el submodo actual, se lee el valor almacenado del encoder y se compara con el leído en la anterior iteración del bucle. Si el valor ha cambiado, se almacena el nuevo valor, y si además han pasado más de 50ms desde la última actualización de la salida (para evitar posibles rebotes con la señal del encoder y no enviar valores de salida distintos demasiado consecutivos) se procede a cambiar la salida generada. Para ello, se actualiza el dato de salida sumando (o restando si el encoder se ha movido en la dirección contraria a las agujas del reloj) el incremento del encoder, multiplicado de forma que corresponda a la resolución de cambio correspondiente. Después se comprueba que el valor esté acotado al rango propio de la salida, y muestra por pantalla el valor de salida actualizado. Para hacer esto correctamente se necesita conocer el número de cifras que componen el número del dato de salida, por lo que se comprueba (también si es negativo para incluir el signo) y se escribe en el display en las posiciones que mejor muestren el dato. Por último, se realiza la función correspondiente en cada caso para enviar al componente asociado la instrucción de generar la señal de salida actualizada.

En el caso de la simulación de termopares, el esquema general es el mismo, pero incluye varios cambios. La actualización del dato requerido de salida se realiza en varias fases, de forma que el cambio del encoder cambia el valor de los grados a simular en resolución de un grado. Después se comprueba que los grados entren en el rango propio de cada tipo de termopar y se realiza una interpolación entre el dato de los grados y el cambio que corresponde en la tensión el cambiar un grado. Esto se realiza para ganar precisión, aunque no permite utilizar multiplicadores para que el cambio del encoder corresponda a un número distinto de grados. Esta interpolación depende del tipo de termopar que se quiera simular. El incremento resultado de la interpolación se suma a la tensión teórica que corresponde a los grados simulados en la iteración anterior, y se añade un factor de corrección hallado experimentalmente para compensar los errores cometidos por el hardware y que la señal generada corresponda con el valor real que se quiere simular.

- Funciones

  Por último, en el programa se encuentran las funciones que se utilizan para conseguir que el funcionamiento sea correcto.

- *Comienzo_DAC1220*: realiza la inicialización del DAC1220, incluyendo el patrón de reset en la entrada de reloj y la configuración de la comunicación SPI.

- *EscribeTensión*: envía al DAC1220 la tensión a generar. Se utiliza solo en la función de generación de tensión, ya que la recta de interpolación entre el valor analógico y digital incluye la amplificación del AO posterior a la salida del DAC1220. Esta recta de interpolación no es la teórica, ya que se han incluido factores de corrección para que la salida real sea más precisa.

- *EscribeTensionTermopar*: envía al DAC1220 la tensión a generar. Incluye la conversión analógica a digital según la recta de regresión teórica, ya que las correcciones para que la salida real sea más precisa se realizan en la propia sección de termopares.

- *EscribeSalida*: realiza el envío del código de salida al DAC1220 mediante comunicación SPI.

- *DAC1220_Normal*: realiza una escritura en el registro de comandos del DAC1220 para cambiar su modo de funcionamiento a normal.

- *DAC1220_Sleep*: realiza una escritura en el registro de comandos del DAC1220 para cambiar su modo de funcionamiento a reposo. Esto se realiza cuando la salida no es generada por el DAC1220, de forma que se reduce el consumo.

- *InterpolacionJ*: realiza la conversión entre grados y el incremento de tensión que supone el incremento entre ese valor de grados y su anterior consecutivo, para termopares tipo J.

- *InterpolacionK*: realiza la conversión entre grados y el incremento de tensión que supone el incremento entre ese valor de grados y su anterior consecutivo, para termopares tipo K.

- *InterpolacionE*: realiza la conversión entre grados y el incremento de tensión que supone el incremento entre ese valor de grados y su anterior consecutivo, para termopares tipo E.

- *InterpolacionT*: realiza la conversión entre grados y el incremento de tensión que supone el incremento entre ese valor de grados y su anterior consecutivo, para termopares tipo T.

- *Comienzo_AD421*: configura la comunicación SPI con el AD421.

- *EscribeCorriente*: envía al AD421 la corriente a estableces en el lazo de salida, previa conversión entre el valor analógico y el digital según una recta de regresión que añade unos factores de corrección a la recta teórica para mejorar la exactitud en la salida real.

- *Escribe_AD421*: realiza el envío del código de salida al AD421 mediante comunicación SPI.

- *AD9833_Reset*: realiza el reset del AD9833 para cambiarlo a su configuración inicial.

- *AD9833_SetFrecuencia*: envía al AD9833 la frecuencia y la forma de onda de la salida después de hacer la conversión entre el valor analógico y el digital.

- *AD9833_EscribeRegistro*: realiza el envío del código de salida al AD9833 mediante comunicación SPI.

- *interruptCountF1*: función de llamada cuando se activa la interrupción 0. Suma una unidad al contador asociado a la función F1 y mantiene ese valor entre 0 y 5, ya que esa función cuenta con 5 submodos. Cambia el resto de contadores a -1 para que permanezcan negativos y no se realicen sus respectivas funciones.

- *interruptCountF2*: función de llamada cuando se activa la interrupción 1. Suma una unidad al contador asociado a la función F2 y mantiene ese valor entre 0 y 3, ya que esa función cuenta con 3 submodos. Cambia el resto de contadores a -1 para que permanezcan negativos y no se realicen sus respectivas funciones.

- *interruptCountF3*: función de llamada cuando se activa la interrupción 4. Suma una unidad al contador asociado a la función F3 y mantiene ese valor entre 0 y 4, ya que esa función cuenta con 4 submodos. Cambia el resto de contadores a -1 para que permanezcan negativos y no se realicen sus respectivas funciones.

- *interruptCountF4*: función de llamada cuando se activa la interrupción 5. Suma una unidad al contador asociado a la función F4 y mantiene ese valor entre 0 y 6, ya que esa función cuenta con

6 submodos. Cambia el resto de contadores a -1 para que permanezcan negativos y no se realicen sus respectivas funciones.

- *cifrasFrec*: cuenta las cifras de un número tipo *unsigned long*, para mostrar correctamente el valor de salida de frecuencia en la función correspondiente.

- *cifrasTer*: cuenta las cifras de un número tipo *int*, para mostrar correctamente el valor de salida de grados en la función de simulación de termopares.

- *ComprobarBateria*: lee la entrada analógica A2 y comprueba la tensión que recibe. Si es inferior a 3.5V y no se estaba mostrando un aviso en la pantalla de batería baja, se escribe dicha alerta. Si por el contrario se estaba mostrando el aviso y la tensión es superior, se borra la alerta.

4.4. Cálculos

4.4.1. Rectas de interpolación analógico-digital

Los componentes que generan las magnitudes de salida en el calibrador incluyen un convertidor digital a analógico, por lo que el dato que necesitan recibir dichos componentes debe ser un valor digital. Por motivos de sencillez en el diseño del programa, con la interfaz del calibrador se escogen los valores analógicos que se quieren conseguir en la salida, por lo que esos valores analógicos hay que transformarlos en digitales para poder enviarlos a los componentes, y que cada componente transforme de vuelta ese dato a una salida analógica. Estas primeras transformaciones de analógico a digital se realizan mediante rectas de interpolación.

- DAC1220: la recta de interpolación del DAC1220 depende del formato en que se escriba en los registros, esto es, binario natural o complemento a 2, además de la resolución, 16 o 20 bits. En esta aplicación se ha utilizado el binario natural y resolución de 20 bits. La salida abarca desde 0V hasta $2*V_{REF}$ (siendo V_{REF} = 2.5V, por lo que el rango alcanza hasta 5V), y al tener una resolución de 20 bits se pueden obtener 2^{20} valores en ese rango.

$$V_{OUT} = 2 \cdot V_{REF} \cdot \frac{codigo}{2^{20}} = 5 \cdot \frac{codigo}{2^{20}} \quad (15)$$

$$codigo = \frac{V_{OUT}}{5} \cdot 2^{20} = \frac{V_{OUT}}{5} \cdot FFFFFh \quad (16)$$

$$codigo = \frac{V_{OUT} + 0.0833}{5 \cdot\ 2.5 \cdot 0.9984} \cdot 2^{20} = \frac{V_{OUT} + 0.0833}{5 \cdot\ 2.5 \cdot 0.9984} \cdot FFFFFh \quad (17)$$

La ecuación (15) es la recta de interpolación teórica digital a analógico, por lo que para obtener la recta analógica a digital hay que despejar el código respecto a la tensión de salida, lo cual puede encontrarse en la ecuación (16), que es la recta de interpolación teórica, utilizada en la aplicación de termopares. La ecuación (17) es la recta real utilizada en la aplicación de generación de tensión, ya que incluye el factor de multiplicación del circuito amplificador no inversor, así como factores de corrección hallados experimentalmente para minimizar el error cometido entre la señal deseada y la generada finalmente.

- AD421: el AD421 se va a utilizar siempre en su modo normal de funcionamiento (no el modo de alarma) por lo que el alcance de la salida comprende entre 4 y 20mA, siendo la resolución de 16 bits (2^{16} valores). En la ecuación (18) se expresa el valor analógico de salida ideal respecto del código digital utilizado. La ecuación (19) es la recta de interpolación teórica analógica a digital, y la ecuación (20) es la ecuación una vez se le han añadido factores de corrección para compensar la pérdida de idealidad en la implementación.

$$I_{OUT} = 16 \cdot \frac{codigo}{2^{16}} + 4 \quad (18)$$

$$codigo = \frac{I_{OUT} - 4}{16} \cdot 2^{16} = \frac{I_{OUT} - 4}{16} \cdot FFFFh \quad (19)$$

$$codigo = \frac{I_{OUT} - 4}{16} \cdot FFA8h - 15 \quad (20)$$

- AD9833: el rango teórico del AD9833 alcanza desde 0Hz hasta la frecuencia del cristal de reloj que se conecta para su funcionamiento, en este caso 25MHz. El tamaño del registro donde se escribe el valor digital es de 28 bits, aunque solo se cambian los 14 bits menos significativos del registro para escribir el dato. La ecuación (21) muestra la relación entre la frecuencia de salida y la palabra digital recibida, y la ecuación (22) es la recta de interpolación teórica, a la cual no ha sido necesario añadir factores de corrección por ser la salida

suficientemente cercana a la ideal, siempre y cuando permanezca en el rango para el que se ha acotado (0 – 50kHz).

$$f_{OUT} = 25 \cdot 10^6 \cdot \frac{codigo}{2^{28}} \quad (21)$$

$$codigo = \frac{f_{OUT}}{25 \cdot 10^6} \cdot 2^{28} \quad (22)$$

4.4.2. Ecuaciones de interpolación temperatura – tensión

Los termopares son sensores que generan una diferencia de potencial en función de la temperatura. La relación entre ambas magnitudes varía en función del tipo de termopar, y está recogida en tablas normalizadas, ya que el comportamiento de la relación no es lineal. La memoria del Arduino Mega 2560 no tiene la capacidad de almacenamiento suficiente para indexar dichas tablas, por lo que se ha optado por aproximar la relación por tramos mediante ecuaciones.

Las tensiones que se quieren generar para simular la respuesta de un termopar deben ser muy precisas, ya que la diferencia de tensión entre dos grados consecutivos llega a ser del orden de 10^{-6}V, y por lo tanto el más pequeño error puede suponer una diferencia de temperatura simulada de varios grados de temperatura. Para minimizar el error, se ha optado por tomar como salida de las ecuaciones de interpolación no el dato directo de la tensión a generar, sino la diferencia entre esa tensión y la correspondiente a el valor de los grados inmediatamente anterior. De esta forma, ya que el rango total de valores posibles es menor, el posible error cometido en la aproximación por una ecuación también es menor.

Para calcular las ecuaciones se han realizado las gráficas de tensión respecto a la temperatura. Se ha realizado un procedimiento de regresión polinomial por tramos de la salida de los termopares. Se han elegido los tramos de manera que se alcance un compromiso entre la simplicidad del polinomio de aproximación y la exactitud de la aproximación. De esta forma, se ha conseguido que la mayoría de errores se encuentran por debajo de ±0.001mV, aunque no ha sido posible evitar que una pequeña minoría tenga un error de ±0.002mV. Las gráficas y las ecuaciones de los 4 tipos de termopares se encuentran especificadas en las siguientes imágenes.

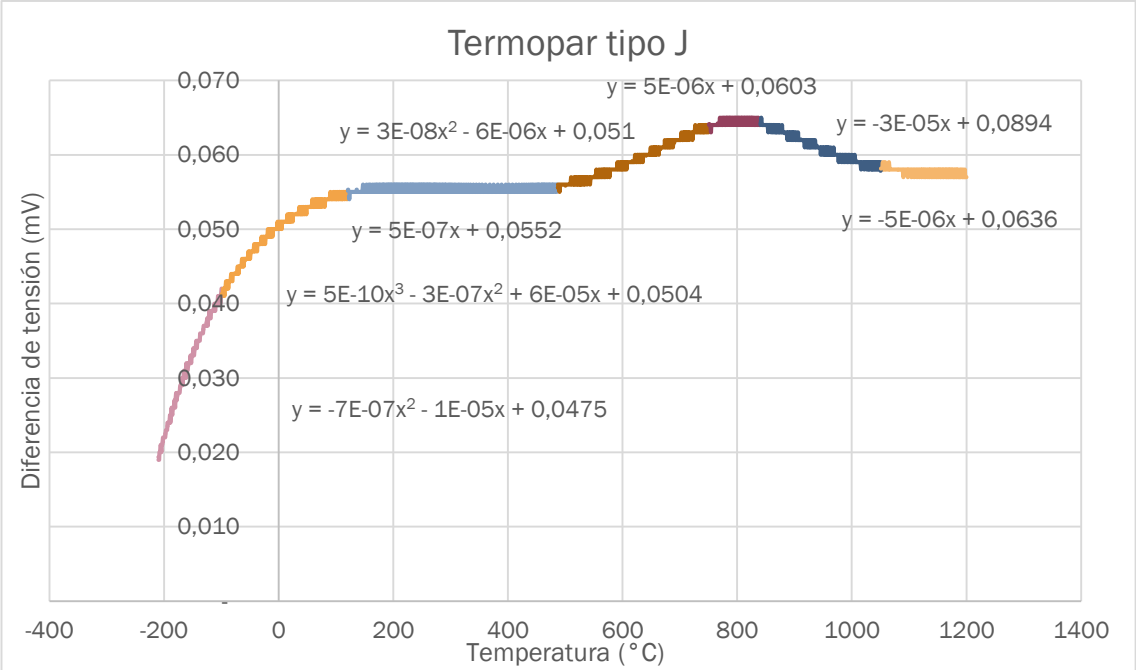


Figura 38. Gráfica de relación Tensión - Temperatura en termopares J

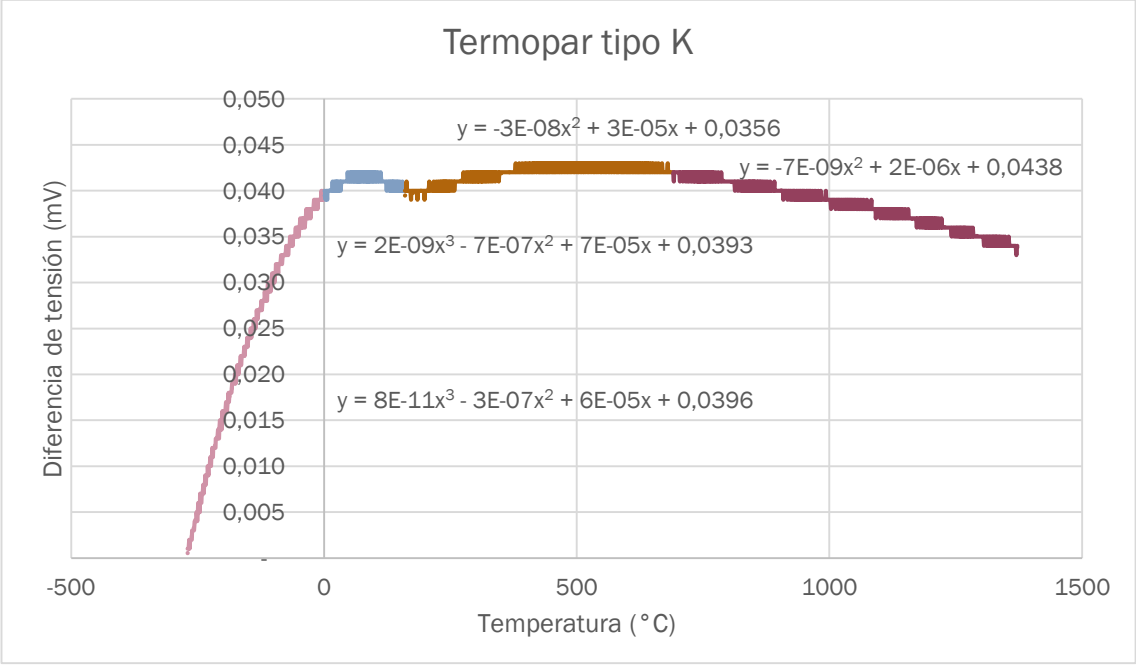


Figura 39. Gráfica de relación Tensión - Temperatura en termopares K

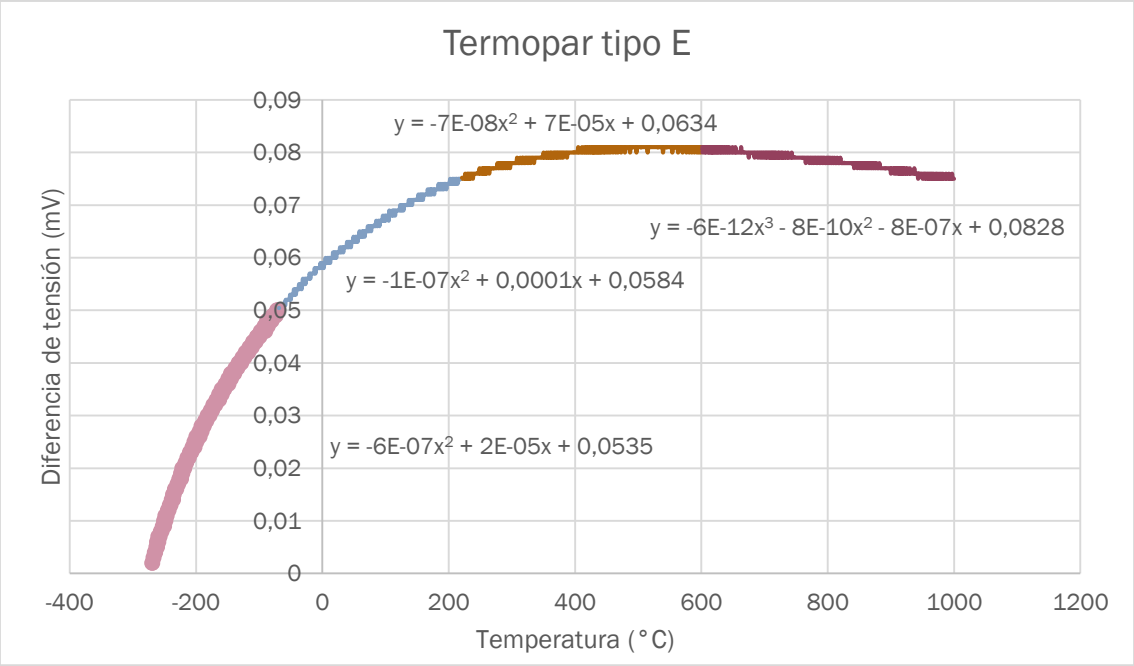


Figura 40. Gráfica de relación Tensión - Temperatura en termopares E

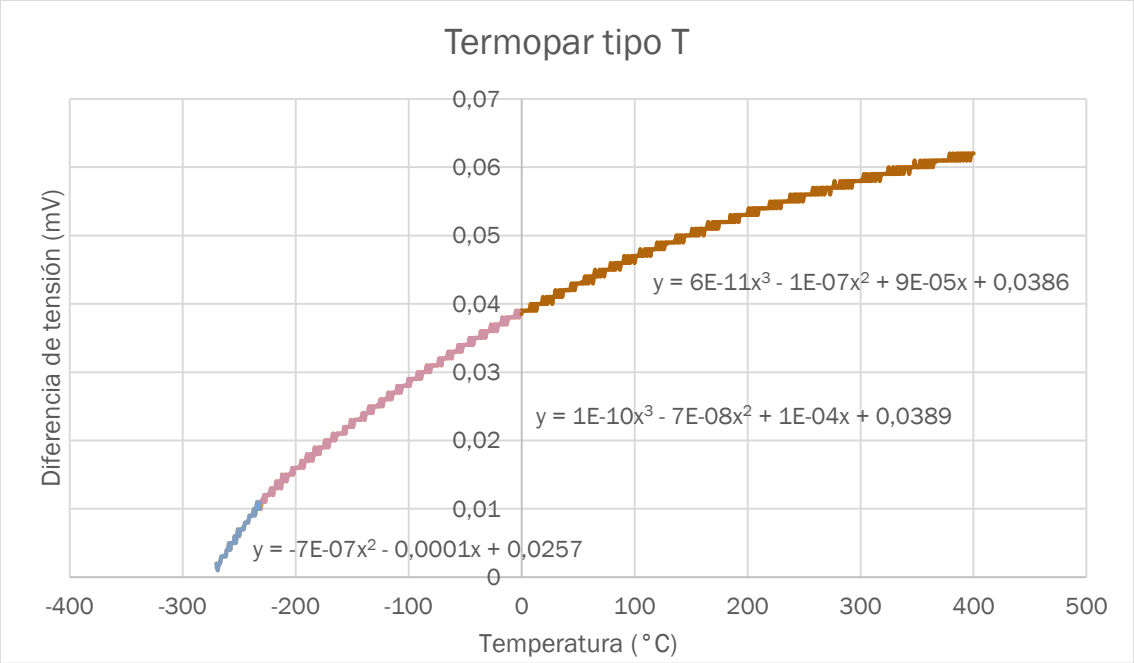


Figura 41. Gráfica de relación Tensión - Temperatura en termopares T

4.4.3. Cálculo de errores

En este apartado se va a calcular la precisión con la que se generan las distintas salidas. En primer lugar, en el *datasheet* del DAC1220 se especifica que el error de ganancia máximo es de ±150ppm FSR, a lo que podríamos añadir el error cometido por la resolución de 20 bits. Tomando como fondo de escala los 5V que puede proporcionar el DAC1220, en la ecuación (23) se calcula el error máximo que se puede cometer debido a el propio componente.

$$Error_{DAC1220,máx} = \pm\left(\frac{150}{10^6}\cdot 5V + \frac{1}{2^{20}}\cdot 5V\right) = \\ = \pm(0.75mV + 4.77\mu V) \approx \pm 0.755mV \quad (23)$$

En la aplicación como generador de tensión, la salida del DAC1220 atraviesa una etapa de amplificación, por lo que el error también aumenta. La relación de amplificación puede encontrarse en la ecuación (1), y depende de las resistencias R_1 (inversamente proporcional) y R_2 (directamente proporcional). Ambas resistencias tienen una tolerancia de un 2%. En la ecuación (24) puede encontrarse el error máximo después de atravesar la etapa con la máxima relación de amplificación posible.

$$Error_{Gen\ Tensión,máx} = \pm 0.755mV \cdot \left(1 + \frac{1500 + 1500\cdot 0.02}{1000 - 1000\cdot 0.02}\right) = \\ = \pm 1.93mV \quad (24)$$

Este error es demasiado alto para la aplicación que se quiere realizar, por lo que para minimizarlo se han incluido factores de corrección en el programa que controla el instrumento. La forma de calcularlos ha sido medir la salida con un multímetro *Fluke 177* y modificarlos hasta que la medición coincidiera con la salida requerida, hasta en ±1 cifra de la mayor resolución (±0.1mV en este caso). Según el *datasheet* de ese modelo de multímetro, su precisión en la medida de tensiones en CC es del 0.09%, por lo que se puede tomar ± (0.09% + 0.1mV) como error en la salida para la aplicación de generación de tensión.

La salida generada en la aplicación de simulación de termopares tiene más resolución que la precisión que alcanza el multímetro, por lo que se ha utilizado para realizar las mediciones un termómetro *TME MM2000*. Este termómetro puede leer una gran variedad de termopares, tiene una resolución de 0.1 °C y una precisión de ±0.15% de lectura ±0.2 °C. Además, cuenta con un circuito de compensación de la unión fría que, aunque no elimina completamente la desviación en función de la temperatura ambiente, si lo suficiente como para considerarse que no es necesario incluir por el momento un circuito de

medición de temperatura para corregir esa posible desviación. En todo caso, la temperatura a la que se ha trabajado se sitúa alrededor de los 20°C.

Las gráficas de interpolación de termopares se aproximan a la relación teórica de las tablas normalizadas con un error menor de 0.002mV. Además, después de la conversión se han incluido factores de corrección para que la salida requerida coincida con la medida en el termómetro. Aun así, debido variaciones en la tensión producidos por ruidos entre la salida del DAC1220 y la entrada del termómetro (se han intentado minimizar disminuyendo la distancia desde el DAC1220 hasta el conector de salida y trenzando el cable que conecta el calibrador con el termómetro, pero sigue sin ser posible eliminarlos del todo), la lectura de la salida no permanece constante, sino que varía alrededor de la salida deseada en un rango que puede alcanzar hasta ±2°C (aunque suele ser menor de ±1°C).

El AD421 cuenta con un error máximo total en la salida (a 20mA) de ±0.2% del fondo de escala, es decir de 20mA. Además podemos añadirle el error debido a la resolución, de 16 bits. Este error, que es el propio del componente, puede encontrarse en la ecuación (25).

$$Error_{AD421,máx} = \pm\left(0.002 \cdot 20mA + \frac{1}{2^{16}} \cdot 20mA\right) = \\ = \pm(0.04mA + 0.3\mu A) \approx \pm 0.04mA \quad (25)$$

Como en el caso del DAC1220, para compensar el error se han añadido factores de corrección por software y se ha hecho coincidir la medida de la salida por el multímetro con la salida especificada. La precisión de este aparato es del 1% y su resolución de 0.01mA. Aunque a priori el integrado tiene mayor precisión por tener menos error, la exactitud y fiabilidad de un instrumento comercial de medida estará mejor comprobada, por los posibles errores en los valores de los componentes y referencias del circuito de funcionamiento del AD421. Por esto se toma este error y no el teórico del *datasheet* del componente.

El *datasheet* del AD9833 especifica que el error de linealidad (sumando el integral y el diferencial) es de ±1.5 LSB, siendo la resolución la del DAC de la etapa de salida del integrado, 10 bits. Al igual que en los dos componentes anteriores, la salida se ha comprobado con un multímetro, coincidiendo las medidas en el rango 3Hz – 50KHz, en las tres formas de onda, con un error máximo de ±0.1Hz, a lo que hay que sumarle el error propio del multímetro, que es del ±0.1%. El rango de frecuencias ha sido este porque es el que, para las tres formas de onda, el error es pequeño. En la señal triangular, a partir de 50KHz el error crecía de forma exponencial rápidamente. En onda senoidal, este error se disparaba a partir de 55KHz, llegando como máximo a ofrecer

64KHz (aunque la salida deseada fuese mayor). Este error puede deberse también a que cuanto mayor es la frecuencia, menor es la tensión de pico, por lo que en esa frecuencia puede que se tome la salida prácticamente como una señal continua.

Los factores de corrección implementados mediante software se han calculado mediante el proceso reiterativo de calibración-ajuste: se comprueba con un instrumento de medida (el multímetro *Fluke 177* o el termómetro *TME MM2000*) la salida del calibrador y se toma esta salida como una recta teórica que vendría dada por un incremento de una unidad de resolución de la magnitud de salida en función de un incremento de posición en el encoder. Dicha recta se compara con la recta teórica ideal que supondría un error nulo. A continuación, se realiza el proceso de ajuste, que consiste en multiplicar la salida real por unos factores de corrección que equivalen a modificar la pendiente y la ordenada en el origen de la recta de salida. Una vez hecho esto, se repite el proceso de calibración-ajuste hasta que ambas rectas real e ideal coinciden en un grado satisfactorio.

5. Conclusiones

En este TFG se ha diseñado y construido un prototipo funcional de calibrador de procesos. Las funciones de este calibrador incluyen todas las planteadas inicialmente, como son la generación de tensión, la generación de corriente, la generación de frecuencia y la simulación de termopares tipo J, K, E y T.

La elección de componentes y el diseño realizado permite obtener una precisión en las salidas muy notable, sobre todo en las aplicaciones de generación de tensión y corriente. La función de simulación de termopares permite reproducir de forma aproximada la respuesta de termopares con una resolución en grados, aunque esta función puede mejorarse eliminando los ruidos en la salida e incluyendo un sensor de temperatura en el propio interior del prototipo para incluir en los cálculos el efecto de la temperatura ambiente en la unión fría del termopar simulado. En la función de generación de frecuencia, si bien la salida es muy precisa, siendo la resolución de 1Hz, el rango de frecuencias es ligeramente menor al típico de los calibradores de procesos industriales (62.5KHz frente a 50KHz), por lo que en un futuro desarrollo podría mejorarse esta función.

Otro de los objetivos cumplidos es hacer un instrumento portátil. La duración de la batería, de mínimo 8 horas y media, permite utilizar el calibrador sin cargarlo durante una jornada completa de trabajo. Aun así, el consumo interno del dispositivo podría llegar a reducirse (o aumentarse la capacidad de la batería), logrando aumentar la duración de funcionamiento. El principal objeto de desarrollo que debería plantearse para el calibrador es construir una carcasa más resistente que la implementada en cartón para contener este primer prototipo, carcasa que no se ha diseñado en el proyecto final por considerar que sobredimensiona la carga de trabajo propia de un Trabajo de Fin de Grado.

En cuanto a la elección de componentes, la botonera y el encoder han sido reutilizados aprovechando partes de otros instrumentos inútiles en afán de abaratar el coste de producción del prototipo. Por esta razón, para replicar el calibrador deben elegirse componentes que sustituyan a estos en su función.

El programa realizado en Arduino permite perfectamente controlar el dispositivo, así como proporcionar una interfaz en la pantalla intuitiva e implementar la corrección de errores de precisión por software. Pero como prácticamente cualquier programa, siempre se puede mejorar, ya sea implementando todas las funciones en una librería, para que sea más fácil exportar el código generado a otras aplicaciones similares, o corrigiendo

posibles bugs que se puedan presentar en casos muy específicos de comportamiento.

En el ámbito personal, este proyecto me ha servido para ampliar mi conocimiento sobre la generación de señales analógicas a partir de señales digitales, los transductores de temperatura, la utilización de baterías de ion-litio y su carga, la comunicación SPI y la programación en Arduino. Pero sobre todo me ha servido para mejorar mi destreza en el diseño de placas de circuito impreso, y ganar experiencia a la hora de trabajar con hardware, ya que la implementación de los componentes físicos presenta problemas que no se tenían contemplados cuando se hacia el estudio teórico de los mismos, lo que supone una carga de tiempo adicional en muchas ocasiones.

6. Bibliografía

[1] FLUKE. Calibradores de Procesos. https://eu.flukecal.com/es/products/process-calibration-tools

[2] FLUKE. Calibrador de Procesos Multifunción Fluke 726 https://eu.flukecal.com/es/products/process-calibration-tools/multifunction-calibrators/calibrador-de-procesos-multifunci%C3%B3n-flu

[3] PCE. Calibrador. https://www.pce-instruments.com/espanol/instrumento-medida/medidor/calibrador-kat_71009.htm

[4] PCE. Calibrador de procesos PCE-123. https://www.pce-instruments.com/espanol/instrumento-medida/medidor/calibrador-pce-instruments-calibrador-de-procesos-pce-123-det_95170.htm?_list=qr.art&_listpos=9

[5] BEAMEX. Calibradores de Proceso. https://www.beamex.com/es/calibradores/calibradores-de-proceso/

[6] BEAMEX. Beamex MC6. https://www.beamex.com/es/calibradores/beamex-mc6/

[7] FemtoCal. Calibradores de Procesos. https://www.femtocalibracion.es/calibradores-de-procesos/

[8] Crear Mi Empresa. Precisión Y Calidad: Como Influye La Correcta Calibración De Los Equipos En El Entorno Industrial. https://crearmiempresa.es/calibracion-equipos-industriales.html

[9] Cuaderno Técnico: Calibración De Equipos De Medida Industriales Según ISO 9000. https://intranet.ceautomatica.es/old/actividades/jornadas/XXI/documentos/ja00_012/ja00_012.pdf

[10] ARDUINO. What is Arduino? https://www.arduino.cc/en/Guide/Introduction

[11] ARDUINO. Getting Started with Arduino and Genuino products. https://www.arduino.cc/en/Guide/HomePage

[12] ARDUINO. Arduino Software (IDE). https://www.arduino.cc/en/Guide/Environment

[13] ARDUINO. Languaje Reference. https://www.arduino.cc/reference/en/

[14] Wiring. http://wiring.org.co/

[15] Processing. https://processing.org/

[16] ARDUINO. Arduino Mega 2560 Rev3. https://store.arduino.cc/mega-2560-r3

[17] Arduino Mega Pinout. https://www.luisllamas.es/wp-content/uploads/2015/11/aduino-pinout-mega.png

[18] Prometec. ¿Cuánto consume Arduino? https://www.prometec.net/consumos-arduino/

[19] ARDUINO. SPI Library. https://www.arduino.cc/en/Reference/SPI

[20] Stack Exchange. How do you use SPI on an Arduino? https://arduino.stackexchange.com/questions/16348/how-do-you-use-spi-on-an-arduino

[21] Texas Instruments. DAC1220. http://www.ti.com/lit/ds/symlink/dac1220.pdf

[22] Texas Instruments. DAC1220EVM. http://www.farnell.com/datasheets/1813467.pdf

[23] Texas Instruments. E2E Community. DAC1220 20 bit DAC. https://e2e.ti.com/support/data-converters/f/73/t/560304?DAC1220-20-bit-DAC

[24] Analog Devices. AD421. http://www.farnell.com/datasheets/2285599.pdf

[25] Engineer Zone. Can the AD421 function correctly with standard SPI. https://ez.analog.com/data_converters/precision_dacs/f/q-a/26608/can-the-ad421-function-correctly-with-standard-spi

[26] Supertex. DN2535.
http://ww1.microchip.com/downloads/en/DeviceDoc/DN2535%20B062813.pdf

[27] Microchip. DN2530.
http://ww1.microchip.com/downloads/en/devicedoc/20005451a.pdf

[28] Analog Devices. AD9833.
https://www.analog.com/media/en/technical-documentation/data-sheets/AD9833.pdf

[29] Aliexpress. Programmable AD9833 Module.
https://fr.aliexpress.com/item/Programmable-Microprocessors-AD9833-Sine-Square-Wave-DDS-Signal-Generator-Module-Drop-shipping/32687211805.html

[30] AD9833 Waveform Generator.
http://www.vwlowen.co.uk/arduino/AD9833-waveform-generator/AD9833-waveform-generator.htm

[31] Stompville. Bit-banging the AD9833 DDS module.
http://stompville.co.uk/?p=853

[32] Wikipedia. Modulación Sigma- Delta.
https://es.wikipedia.org/wiki/Modulaci%C3%B3n_Sigma-Delta

[33] ARDUINO. LiquidCrystal Library.
https://www.arduino.cc/en/Reference/LiquidCrystal

[34] MIDAS MC21605J6W-SPR-V2.
http://www.farnell.com/datasheets/2175686.pdf

[35] MIDAS MC21605B6WD-SPTLY.
http://www.farnell.com/datasheets/2021788.pdf

[36] Li-ion Rechargeable Battery.
http://www.farnell.com/datasheets/1806913.pdf?_ga=2.38359139.1559076775.1525961159-609036267.1520336764&_gac=1.12142150.1525804256.CjwKCAjwlcXXBRBhEiwApfHGTSbEVNNDIEsEDE8QrlE4CH7-yXYZfXIhHuzhs8o0xvOQMf-RGtZxxRoCTAAQAvD_BwE

[37] MASCOT Exchangeable AC plug models. http://www.farnell.com/datasheets/16261.pdf?_ga=2.214102231.1371724442.1526378711-609036267.1520336764&_gac=1.193320287.1526476462.Cj0KCQjwre_XBRDVARIsAPf7zZjeZqntZf47IksfHLgHy7gRoPsoANyfcZb1HyXwtmPqCWyRnR-B9QsaAvG9EALw_wcB

[38] MASCOT. 2240LI chargers. http://www.farnell.com/datasheets/16316.pdf?_ga=2.108777538.1371724442.1526378711-609036267.1520336764&_gac=1.115800308.1526476462.Cj0KCQjwre_XBRDVARIsAPf7zZjeZqntZf47IksfHLgHy7gRoPsoANyfcZb1HyXwtmPqCWyRnR-B9QsaAvG9EALw_wcB

[39] Nomadas Electrónicos. Fuente/Cargador con carga compartida. https://nomadaselectronicos.wordpress.com/2015/05/22/fuentecargador-con-carga-compartida-load-sharing/

[40] Battery University. BU-409: Charging Lithium-ion. https://batteryuniversity.com/learn/article/charging_lithium_ion_batteries

[41] Prometec. Rotary Encoders. https://www.prometec.net/rotary-encoders/

[42] Arduino Library List. Encoder. https://www.arduinolibraries.info/libraries/encoder

[43] ARDUINO. attachInterrupt(). https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

[44] Prometec. Arduino y las interrupciones. https://www.prometec.net/interrupciones/

[45] Luis Llamas. Qué son y cömo usar interrupciones en Arduino. https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/

[46] TRACO POWER. TMR 1. http://www.farnell.com/datasheets/2200579.pdf?_ga=2.265156879.1454402126.1527527383-609036267.1520336764&_gac=1.122016889.1527248118.Cj0KCQjw6J7YBRC4ARIsAJMXXsf-

oJMDHbWU0JjGTYfjxtYYXElOLDHoxi0EsDzh5vm3hgsKH5Kat34aAqxaEALw_wcB

[47] OMRON. G5V-2 Low Signal Relay. http://www.farnell.com/datasheets/2182432.pdf?_ga=2.111125250.620454931.1527010501-609036267.1520336764&_gac=1.61275870.1527240191.Cj0KCQjw6J7YBRC4ARIsAJMXXsfQwICBAr0bBffY3FbqpMadRWCRywKF3qjU7OHfhURRIuloxDY4J5AaAuC5EALw_wcB

[48] RS Components. Regulador de tensión lineal. https://es.rs-online.com/web/p/reguladores-de-tension-lineal/1891295/

[49] RS Components. Conector jack macho. https://es.rs-online.com/web/p/conectores-jack/8104591/

[50] RS Components. Conector jack hembra. https://es.rs-online.com/web/p/conectores-jack/8051696/

[51] RS Components. Interruptor actuador deslizante. https://es.rs-online.com/web/p/interruptores-de-actuador-deslizante/7023546/

[52] RS Components. Cabezal de pines. https://es.rs-online.com/web/p/conectores-macho-para-pcb/6812991/

[53] RS Components. Conector Hembra Faston negro. https://es.rs-online.com/web/p/conectores-banana/2306293/

[54] RS Components. Conector Hembra Faston rojo. https://es.rs-online.com/web/p/conectores-banana/2306287/

[55] RS Components. Bloque terminal PCB macho. https://es.rs-online.com/web/p/bloques-terminales-para-pcb/8971219/

[56] RS Components. Bloque terminal PCB hembra. https://es.rs-online.com/web/p/bloques-terminales-para-pcb/8971262/

[57] RS Components. Conector termopar. https://es.rs-online.com/web/p/accesorios-para-sensores-de-temperatura/4559679/

[58] RS Components. Termopar J. https://es.rs-online.com/web/p/termopares/1365868/

[59] KiCad EDA. http://kicad-pcb.org/

[60] KiCad User Manual. http://docs.kicad-pcb.org/5.0.2/en/kicad/kicad.pdf

[61] KiCad Libraries. https://kicad.github.io/symbols/

[62] KiCad Electronic CAD libraries. http://smisioto.no-ip.org/elettronica/kicad/kicad-en.htm

[63] Tabla Termopar tipo J. https://www.unirioja.es/cu/lzorzano/Emfj_c.pdf

[64] Tabla Termopar tipo K. https://www.unirioja.es/cu/lzorzano/Emfk_c.pdf

[65] Tabla Termopar tipo E. https://www.unirioja.es/cu/lzorzano/Emfe_c.pdf

[66] Tabla Termopar tipo T. https://www.unirioja.es/cu/lzorzano/Emft_c.pdf

[67] FLUKE. Serie Fluke 170. https://dam-assets.fluke.com/s3fs-public/6011663a-es-17x-ds-w.pdf?ice5u5CVHEFPunhsO5IJVpA00_CxV4I0

[68] TME. MM2000. https://www.tmethermometers.com/mwdownloads/download/link/id/2/

6.1. Fuentes de las imágenes

Figura 1: https://media.rs-online.com/t_large/R6699662-01.jpg

Figura 2: https://www.beamex.com/wp-content/uploads/2016/06/MC6-facing-left-v1-300x224.jpg

Figura 3: https://www.pce-instruments.com/espanol/slot/4/artimg/normal/pce-instruments-calibrador-de-procesos-pce-123-95170_525713.jpg

Figura 4: https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/500x375/f8876a31b63532bbba4e781c30024a0a/a/0/a000067_iso_1_.jpg

Figura 5: http://www.ti.com/lit/ds/symlink/dac1220.pdf

Figura 6: https://media.digikey.com/photos/Texas%20Instr%20Photos/DAC1220EVM.jpg

Figura 7: http://www.ti.com/lit/ds/symlink/dac1220.pdf

Figura 8: http://www.farnell.com/datasheets/1813467.pdf

Figura 9: http://www.farnell.com/datasheets/1813467.pdf

Figura 10: http://www.farnell.com/datasheets/1813467.pdf

Figura 14: http://www.farnell.com/datasheets/2285599.pdf

Figura 15: http://www.farnell.com/datasheets/2285599.pdf

Figura 17: http://www.farnell.com/datasheets/2285599.pdf

Figura 19: http://www.farnell.com/datasheets/2182432.pdf?_ga=2.111125250.620454931.1527010501-609036267.1520336764&_gac=1.61275870.1527240191.Cj0KCQjw6J7YBRC4ARIsAJMXXsfQwlCBAr0bBffY3FbqpMadRWCRywKF3qjU7OHfhURRIuloxDY4J5AaAuC5EALw_wcB

Figura 21: https://media.rs-online.com/t_large/F2306287-01.jpg

Figura 22: https://www.analog.com/media/en/technical-documentation/data-sheets/AD9833.pdf

Figura 23 (izquierda): https://ae01.alicdn.com/kf/HTB1WzAyQpXXXXXZXpXXq6xXFXXXM/220343492/HTB1WzAyQpXXXXXZXpXXq6xXFXXXM.jpg

Figura 23 (derecha):
https://ae01.alicdn.com/kf/HTB1nlQHQpXXXXaKXXXXq6xXFXXX2/220343492/HTB1nlQHQpXXXXaKXXXXq6xXFXXX2.jpg

Figura 24: https://uk.farnell.com/productimages/standard/en_GB/2063240-40.jpg

Figura 28: https://es.farnell.com/productimages/standard/en_GB/2401854-40.jpg

Figura 29:
http://www.portatilmovil.com/a/Carga%20de%20iones%20de%20litio/03.jpg

Anexo A: Código de Programa

```
/*    Programa de implementación de un Calibrador de Procesos
      Autor: Samuel Pérez Trigueros
      Proyecto TFG, Escuela de Ingenierías Industriales,
Universidad de Valladolid
      Febrero, 2019
*/

// ****************** Declaración de variables
*********************

// ------------------------- Librerías ---------------------------
--
#include <LiquidCrystal.h>
#include <Encoder.h>
#include <SPI.h>


// -------------------------- Botones ----------------------------
--
const int timeThreshold = 500; // Espera en la pulsación para
evitar rebotes
long timeCounter = 0;

const int PinInterrF1 = 2;    // Pines correspondientes a las
entradas
const int PinInterrF2 = 3;    // de las interrupciones
const int PinInterrF3 = 19;
const int PinInterrF4 = 18;

volatile int ISRCounterF1 = 0;    // Variables que cambian las
interrupciones
volatile int ISRCounterF2 = -1;   // para cambiar de modo de
funcionamiento
volatile int ISRCounterF3 = -1;
volatile int ISRCounterF4 = -1;

// ---------------------- Pines Auxiliares ----------------------
--
const int PinRele = 46;
const int PinBateria = A2;
const int BateriaBaja = 717;    // Aviso cuando la señal de la
batería baja de 3.5V


// -------------------------- Pantalla ---------------------------
--
const int rs = 42, en = 40, d4 = 36, d5 = 34, d6 = 30, d7 = 32;
const int numRows = 2;    // Tamaño de la pantalla: 16x2
const int numCols = 16;
int cifrasAnt = 0;        // Variable utilizada después para
contar cifras
```

```
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); // Función de
inicialización
                                           // de la librería
LiquidCrystal.h

// Los siguientes son los caracteres especialmente creados para el
display y mostrar la
// información de forma más intuitiva. Corresponden a los símbolos
de los tipos de onda
// (seno, cuadrada y triangular), la flecha que indica la cifra
que se cambia con el
// encoder, y la alerta que indica batería baja
byte sen1[8] = {B00000, B00000, B00110, B01001, B10000, B10000,
B00000, B00000};
byte sen2[8] = {B00000, B00000, B00001, B00001, B10010, B01100,
B00000, B00000};
byte cuad[8] = {B00000, B00000, B01111, B01001, B01001, B11001,
B00000, B00000};
byte tri1[8] = {B00000, B00000, B00100, B01010, B10001, B00000,
B00000, B00000};
byte tri2[8] = {B00000, B00000, B00010, B00101, B01000, B10000,
B00000, B00000};
byte flecha[8] = {B00000, B00000, B00000, B00000, B00000, B11111,
B01110, B00100};
byte alerta1[8] = {B00000, B11111, B10000, B10100, B10100, B10000,
B11111, B00000};
byte alerta2[8] = {B00000, B11110, B00010, B00011, B00011, B00010,
B11110, B00000};


// --------------------------- Encoder ---------------------------
--
int PinAmarillo = 11;  // Los colores se refieren al color de los
cables
int PinAzul = 12;      // del encoder, pines A y B respectivamente

unsigned long startMillis;        // Utilizamos la función
millis() para contar el tiempo
unsigned long currentMillis;      // entre interacciones, de forma
que se evitan rebotes
const unsigned long period = 50;

int old_Pos_Enc  = -1;  // La librería encoder funciona de forma
que se almacena un valor
                        // y el cambio de posición lo actualiza, y
asignamos -1 al almacenado
                        // antes de comenzar a moverlo
Encoder miEnc(PinAzul, PinAmarillo);  //Función de inicialización
de la librería Encoder.h


// ----------------------------- SPI -----------------------------
--
const int CLKPin = 52;          // Pin de reloj de la com serie
const int DATAPin = 51;         // Pin de datos de la com serie


// --------------------------- DAC1220 ---------------------------
--
```

```
int SCKPin = 48;     // pin 3, pin solo para el reset
int CSPin = 25;      // pin 7, activo a LOW
int SDIENPin = 27;   // pin 9, entrada 13 activa a HIGH


const uint8_t command_dato = 0x40; // Escribimos 3 bytes de datos

float datoTension = 0;

// Las variables que comienzan con T_ son específicas de las
funciones de termopares,
// siendo necesario diferenciarlas por ser el mismo componente que
proporciona tensión
int T_tipo = 1;     // J:1 K:2 E:3 T:4
int T_grados = 0;   // Grados a simular
float T_tension = 0;  // Tensión teórica correspondiente a los
grados a simular
float T_salida = 0;   // T_salida incorpora un factor de
corrección respecto a T_tension
const float T_refTension = 2.538450;    // Hallada empíricamente
float T_incTension = 0; // La interpolación funciona con
incrementos de tensión, no con
                        // los valores absolutos

// ---------------------------- AD421 ----------------------------
--
const int LATCHPin = 23;        // Pin latch del ad421

float datoCorriente = 4.00;


// ---------------------------- AD9833 ----------------------------
--
const int FSYNCPin = 13;        // Pin FSYNC del AD9833

const int SINE = 0x2000;        // Códigoos para decidir el tipo
de onda a generar
const int SQUARE = 0x2028;
const int TRIANGLE = 0x2002;

int waveType = SINE;            // Variable que almacena el tipo
de onda

const float refFreq = 25000000.0;     // Referencia del cristal de
reloj utilizado

unsigned long freq = 1000;


// *************************** Setup
*****************************

void setup()    // En el setup inicializamos los pines, las
funciones de las
{               // interrupciones y creamos los carácteres
especiales

  lcd.begin(numCols, numRows);
```

```
  pinMode(PinInterrF1, INPUT);
  pinMode(PinInterrF2, INPUT);
  pinMode(PinInterrF3, INPUT);
  pinMode(PinInterrF4, INPUT);

  attachInterrupt(digitalPinToInterrupt(PinInterrF1),
interruptCountF1, RISING);
  attachInterrupt(digitalPinToInterrupt(PinInterrF2),
interruptCountF2, RISING);
  attachInterrupt(digitalPinToInterrupt(PinInterrF3),
interruptCountF3, RISING);
  attachInterrupt(digitalPinToInterrupt(PinInterrF4),
interruptCountF4, RISING);

  pinMode(SS, OUTPUT);
  pinMode(CSPin, OUTPUT);
  pinMode(SDIENPin, OUTPUT);
  pinMode(LATCHPin, OUTPUT);
  pinMode(FSYNCPin, OUTPUT);
  pinMode(PinRele, OUTPUT);
  pinMode(PinBateria, INPUT);

  lcd.createChar(0, sen1);
  lcd.createChar(1, sen2);
  lcd.createChar(2, cuad);
  lcd.createChar(3, tri1);
  lcd.createChar(4, tri2);
  lcd.createChar(5, flecha);
  lcd.createChar(6, alerta1);
  lcd.createChar(7, alerta2);

}

// ***************************** Loop
*******************************

void loop()
{
  // ------------------------ Tensión ---------------------------
--
  if (ISRCounterF1 >= 0)    //Setup de F1
  {
    lcd.clear();
    lcd.setCursor(4, 1);
    lcd.print("Cargando...");  // Mostramos un mensaje por
pantalla mientras de configura
                              // el DAC1220
    datoTension = 0.9999;
    float multiplicadorTension = 0.0001;

    digitalWrite(PinRele, LOW);
    digitalWrite(SDIENPin, HIGH);

    Comienzo_DAC1220();       // Configuración inicial del DAC1220
    DAC1220_Normal();

    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("V");
```

```
    lcd.setCursor(11, 1);
    lcd.print(datoTension, 3);

    EscribeTension(datoTension);

    while (ISRCounterF1 >= 0) // Loop de F1
    {
      ComprobarBateria();  // Comprobamos que la batería no está
en un nivel bajo de carga

      switch (ISRCounterF1) // Cada vez que se pulsa F1 se cambia
de submodo, en este
      {                     // caso las cifras que cambiamos con
el encoder
        case 0:
          multiplicadorTension = 0.0001;
          lcd.setCursor(10, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 1:
          multiplicadorTension = 0.001;
          lcd.setCursor(15, 0);
          lcd.print(" ");
          lcd.setCursor(14, 0);
          lcd.write(byte(5));
          break;

        case 2:
          multiplicadorTension = 0.01;
          lcd.setCursor(14, 0);
          lcd.print(" ");
          lcd.setCursor(13, 0);
          lcd.write(byte(5));
          break;

        case 3:
          multiplicadorTension = 0.1;
          lcd.setCursor(13, 0);
          lcd.print(" ");
          lcd.setCursor(12, 0);
          lcd.write(byte(5));
          break;

        case 4:
          multiplicadorTension = 1;
          lcd.setCursor(12, 0);
          lcd.print(" ");
          lcd.setCursor(10, 0);
          lcd.write(byte(5));
          break;

      }

      int new_Pos_Enc = miEnc.read();
      int incrementoTension = 0;
```

```
      new_Pos_Enc = new_Pos_Enc / 4;    // Cada cambio de posición
en el encoder cambia en 4 el valor
      incrementoTension = new_Pos_Enc - old_Pos_Enc;
      if (new_Pos_Enc != old_Pos_Enc) // Si cambiamos el valor del
encoder
      {
        old_Pos_Enc = new_Pos_Enc;
        currentMillis = millis();
        if (currentMillis - startMillis >= period) // Actualizar
la salida cada 50 milisegundos
        {
          datoTension = datoTension + incrementoTension *
multiplicadorTension;

          if (datoTension < 0.004) datoTension = 0.004; //
Acotamos la tensión a un rango
          if (datoTension > 10) datoTension = 10;

          if (datoTension < 10)   // La posición donde se escribe
en la pantalla depende
          {                       // del número de cifras que se
escriban
            lcd.setCursor(9, 1);
            lcd.print(" ");
            lcd.print(datoTension, 4);
          }
          else if (datoTension >= 10)
          {
            lcd.setCursor(9, 1);
            lcd.print(datoTension, 4);
          }

          EscribeTension(datoTension);    // Transmitimos el valor
de salida
          startMillis = currentMillis;
        }
      }
    }
  }

  // ----------------------- Corriente --------------------------
--
  if (ISRCounterF2 >= 0)  // Setup del F2
  {
    DAC1220_Sleep(); // Cambiamos el DAC1220 a modo Sleep para
ahorrar consumo

    datoCorriente = 4.00;
    float multiplicadorCorriente = 0.01;

    digitalWrite (LATCHPin, LOW);
    digitalWrite (PinRele, HIGH);
    Comienzo_AD421();               // Configuración inicial del
AD421

    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("mA");
    lcd.setCursor(12, 1);
```

```
    lcd.print(datoCorriente);

    while (ISRCounterF2 >= 0)    // Loop del F2
    {
      ComprobarBateria();        // Comprobamos que la batería no
está en un nivel bajo de carga

      switch (ISRCounterF2)      // Cada vez que se pulsa F2 se
cambia de submodo, en este
      {                          // caso las cifras que cambiamos
con el encoder
        case 0:
          multiplicadorCorriente = 0.01;
          lcd.setCursor(12, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 1:
          multiplicadorCorriente = 0.1;
          lcd.setCursor(15, 0);
          lcd.print(" ");
          lcd.setCursor(14, 0);
          lcd.write(byte(5));
          break;

        case 2:
          multiplicadorCorriente = 1;
          lcd.setCursor(14, 0);
          lcd.print(" ");
          lcd.setCursor(12, 0);
          lcd.write(byte(5));
          break;
      }

      int new_Pos_Enc = miEnc.read();
      int incrementoCorriente = 0;
      new_Pos_Enc = new_Pos_Enc / 4;    // Cada cambio de posición
en el encoder cambia en 4 el valor
      incrementoCorriente = new_Pos_Enc - old_Pos_Enc;
      if (new_Pos_Enc != old_Pos_Enc) // Si cambiamos el valor del
encoder
      {
        old_Pos_Enc = new_Pos_Enc;
        currentMillis = millis();
        if (currentMillis - startMillis >= period) // Actualizar
la salida cada 50 milisegundos
        {
          datoCorriente = datoCorriente + incrementoCorriente *
multiplicadorCorriente;

          if (datoCorriente < 4) datoCorriente = 4;   // Acotamos
la corriente a un rango
          if (datoCorriente > 20) datoCorriente = 20;

          if (datoCorriente < 10)   // La posición donde se
escribe en la pantalla depende
```

```
          {                          // del número de cifras que se
escriban
            lcd.setCursor(11, 1);
            lcd.print(" ");
            lcd.print(datoCorriente);
          }
          else if (datoCorriente >= 10)
          {
            lcd.setCursor(11, 1);
            lcd.print(datoCorriente);
          }

          EscribeCorriente(datoCorriente);    // Transmitimos el
valor de salida
          startMillis = currentMillis;
        }
      }

      currentMillis = millis();                  // Con el AD421 es
necesario refrescar la salida
      if (currentMillis - startMillis >= 2000)  // porque si se
abre el lazo, cuando se vuelve a
      {                                          // cerrar la
corriente cambia a 4mA
        EscribeCorriente(datoCorriente);
        startMillis = currentMillis;
      }
    }
  }

  // ------------------------ Termopar -------------------------
---
  if (ISRCounterF3 >= 0)     // Setup de F3
  {
    lcd.clear();
    lcd.setCursor(4, 1);
    lcd.print("Cargando...");  // Mostramos un mensaje por
pantalla mientras de configura
                              // el DAC1220
    T_tipo = 1;       // Comienzo en tipo J
    T_grados = 0;
    T_tension = T_refTension;
    T_salida = 0;
    T_incTension = 0;

    digitalWrite(PinRele, LOW);
    digitalWrite(SDIENPin, HIGH);

    Comienzo_DAC1220();
    DAC1220_Normal();

    // T_salida incluye un factor de corrección sobre T_tensión,
hallado por prueba y error
    T_salida = T_refTension + (0.00008 + T_grados * 0.0000002);
    EscribeTensionTermopar(T_salida);

    cifrasAnt = 0; // Variable para almacenar las cifras del valor
mostrado anterior por pantalla
```

```
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print((char)223); // Símbolo de grado °
    lcd.print("C");
    lcd.setCursor(15, 1);
    lcd.print(T_grados);

    while (ISRCounterF3 >= 0)
    {
      ComprobarBateria();  // Comprobamos que la batería no está
en un nivel bajo de carga

      switch (ISRCounterF3) // Cada vez que se pulsa F3 se cambia
de submodo, en este
      {                     // caso el tipo de termopar
        case 0:
          if (T_tipo != 1)
          {
            T_grados = 0;               //Si cambiamos de tipo de
termopar,
            T_tension = T_refTension;  //ponemos los grados y
tension a 0
          }
          T_tipo = 1;
          lcd.setCursor(0, 0);
          lcd.print("J");
          lcd.setCursor(13, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 1:
          if (T_tipo != 2)
          {
            T_grados = 0;               //Si cambiamos de tipo de
termopar,
            T_tension = T_refTension;  //ponemos los grados y
tension a 0
          }
          T_tipo = 2;
          lcd.setCursor(0, 0);
          lcd.print("K");
          lcd.setCursor(13, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;


        case 2:
          if (T_tipo != 3)
          {
            T_grados = 0;               //Si cambiamos de tipo de
termopar,
            T_tension = T_refTension;  //ponemos los grados y
tension a 0
          }
          T_tipo = 3;
```

```
          lcd.setCursor(0, 0);
          lcd.print("E");
          lcd.setCursor(13, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 3:
          if (T_tipo != 4)
          {
            T_grados = 0;               //Si cambiamos de tipo de
termopar,
            T_tension = T_refTension;  //ponemos los grados y
tension a 0
          }
          T_tipo = 4;
          lcd.setCursor(0, 0);
          lcd.print("T");
          lcd.setCursor(13, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

      }

      int new_Pos_Enc = miEnc.read();
      int incrementoGrados = 0;
      int Col = 15, Row = 1;
      new_Pos_Enc = new_Pos_Enc / 4;    // Cada cambio de posición
en el encoder cambia en 4 el valor
      incrementoGrados = new_Pos_Enc - old_Pos_Enc;
      if (new_Pos_Enc != old_Pos_Enc) // Si cambiamos el valor del
encoder
      {
        old_Pos_Enc = new_Pos_Enc;
        currentMillis = millis();
        if (currentMillis - startMillis >= period) // Actualizar
la salida cada 50 milisegundos
        {
          T_grados = T_grados + incrementoGrados;

          switch (T_tipo) // Acotamos los grados a un rango, que
depende del tipo de termopar
          {               // y realizamos la interpolación de
grados a tensión
            case 1:
              if (T_grados < -210) T_grados = -210;
              if (T_grados > 1200) T_grados = 1200;
              T_incTension = InterpolacionJ (T_grados);   //
Interpolación según las ecuaciones teóricas

              // T_salida incluye un factor de corrección sobre
T_tensión, hallado por prueba y error
              // y que cambia ligeramente según el tipo de
termopar simulado
              if (incrementoGrados < 0) T_incTension = -
T_incTension;
```

```
              T_tension = T_tension + T_incTension * 0.001;
              T_salida = T_tension + (0.00008 + T_grados *
0.0000002);
              break;

            case 2:
              if (T_grados < -270) T_grados = -270;
              if (T_grados > 1372) T_grados = 1372;
              T_incTension = InterpolacionK (T_grados);   //
Interpolación según las ecuaciones teóricas

              // T_salida incluye un factor de corrección sobre
T_tensión, hallado por prueba y error
              // y que cambia ligeramente según el tipo de
termopar simulado
              if (incrementoGrados < 0) T_incTension = -
T_incTension;
              T_tension = T_tension + T_incTension * 0.001;
              T_salida = T_tension + (0.00028 - abs(T_grados) *
0.0000002);
              break;

            case 3:
              if (T_grados < -270) T_grados = -270;
              if (T_grados > 1000) T_grados = 1000;
              T_incTension = InterpolacionE (T_grados);   //
Interpolación según las ecuaciones teóricas

              // T_salida incluye un factor de corrección sobre
T_tensión, hallado por prueba y error
              // y que cambia ligeramente según el tipo de
termopar simulado
              if (incrementoGrados < 0) T_incTension = -
T_incTension;
              T_tension = T_tension + T_incTension * 0.001;
              T_salida = T_tension - 0.00011;
              break;

            case 4:
              if (T_grados < -270) T_grados = -270;
              if (T_grados > 400) T_grados = 400;
              T_incTension = InterpolacionT (T_grados);   //
Interpolación según las ecuaciones teóricas

              // T_salida incluye un factor de corrección sobre
T_tensión, hallado por prueba y error
              // y que cambia ligeramente según el tipo de
termopar simulado
              if (incrementoGrados < 0) T_incTension = -
T_incTension;
              T_tension = T_tension + T_incTension * 0.001;
              T_salida = T_tension + (0.00030 + T_grados *
0.0000001);
              break;
          }

          if (T_grados >= 0)    // La posición donde se escribe en
la pantalla depende
```

```
          {                    // del número de cifras que se
escriban y del signo
            if (cifrasAnt > cifrasTer(T_grados))
            {
              Col = Col - cifrasAnt;
              lcd.setCursor(Col, Row);
              lcd.print(" ");
              Col = 15;
            }
            Col = Col - cifrasTer(T_grados) - 1;
            lcd.setCursor(Col, Row);
            lcd.print(" ");
            lcd.print(T_grados);
            cifrasAnt = cifrasTer(T_grados);
          }
          else if (T_grados < 0)
          {
            if (cifrasAnt > cifrasTer(T_grados))
            {
              Col = Col - cifrasAnt - 1;
              lcd.setCursor(Col, Row);
              lcd.print(" ");
              Col = 15;
            }
            Col = Col - cifrasTer(T_grados) - 1;
            lcd.setCursor(Col, Row);
            lcd.print(T_grados);
            cifrasAnt = cifrasTer(T_grados);
          }

          EscribeTensionTermopar(T_salida);    // Transmitimos el
valor de salida
          startMillis = currentMillis;
        }
      }
    }
  }


  // ----------------------- Frecuencia ------------------------
--
  if (ISRCounterF4 >= 0)    // Setup de F4
  {
    DAC1220_Sleep(); // Cambiamos el DAC1220 a modo Sleep para
ahorrar consumo

    int multiplicadorFrecuencia = 1;
    digitalWrite(PinRele, LOW);
    freq = 1000;
    cifrasAnt = 0;
    AD9833_Reset();       // Inicialización del AD9833
    delay(50);

    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("Hz");
    lcd.setCursor(12, 1);
    lcd.print(freq);
```

```
    AD9833_SetFrecuencia(freq, waveType);

    while (ISRCounterF4 >= 0)    // Loop de F4
    {
      ComprobarBateria();   // Comprobamos que la batería no está
en un nivel bajo de carga

      switch (ISRCounterF4) // Cada vez que se pulsa F4 se cambia
de submodo, en este
      {                     // caso el tipo de onda y las cifras
que cambiamos con el encoder
        case 0:
          waveType = SINE;
          multiplicadorFrecuencia = 1;
          lcd.setCursor(0, 0);
          lcd.write(byte(0));
          lcd.write(byte(1));
          lcd.setCursor(12, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 1:
          waveType = SINE;
          multiplicadorFrecuencia = 1000;
          lcd.setCursor(15, 0);
          lcd.print(" ");
          lcd.setCursor(12, 0);
          lcd.write(byte(5));
          break;

        case 2:
          waveType = SQUARE;
          multiplicadorFrecuencia = 1;
          lcd.setCursor(0, 0);
          lcd.write(byte(2));
          lcd.write(byte(2));
          lcd.setCursor(12, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 3:
          waveType = SQUARE;
          multiplicadorFrecuencia = 1000;
          lcd.setCursor(15, 0);
          lcd.print(" ");
          lcd.setCursor(12, 0);
          lcd.write(byte(5));
          break;

        case 4:
          waveType = TRIANGLE;
          multiplicadorFrecuencia = 1;
          lcd.setCursor(0, 0);
          lcd.write(byte(3));
          lcd.write(byte(4));
```

```
          lcd.setCursor(12, 0);
          lcd.print(" ");
          lcd.setCursor(15, 0);
          lcd.write(byte(5));
          break;

        case 5:
          waveType = TRIANGLE;
          multiplicadorFrecuencia = 1000;
          lcd.setCursor(15, 0);
          lcd.print(" ");
          lcd.setCursor(12, 0);
          lcd.write(byte(5));
          break;
      }

      int new_Pos_Enc = miEnc.read();
      int incrementoFrecuencia = 0;
      int Col = 15, Row = 1;
      new_Pos_Enc = new_Pos_Enc / 4;    // Cada cambio de posición
en el encoder cambia en 4 el valor
      incrementoFrecuencia = new_Pos_Enc - old_Pos_Enc;
      if (new_Pos_Enc != old_Pos_Enc) // Si cambiamos el valor del
encoder
      {
        old_Pos_Enc = new_Pos_Enc;
        currentMillis = millis();
        if (currentMillis - startMillis >= period) // Actualizar
la salida cada 50 milisegundos
        {
          freq = freq + incrementoFrecuencia *
multiplicadorFrecuencia ;

          if (freq < 3) freq = 3;           // Acotamos la
frecuencia a un rango
          if (freq > 50000) freq = 50000;

          if (cifrasAnt > cifrasFrec(freq)) // La posición donde
se escribe en la pantalla depende
          {                                 // del número de
cifras que se escriban
            Col = Col - cifrasAnt;
            lcd.setCursor(Col, Row);
            lcd.print(" ");
            Col = 15;
          }
          Col = Col - cifrasFrec(freq);
          lcd.setCursor(Col, Row);
          lcd.print(freq);
          cifrasAnt = cifrasFrec(freq);

          AD9833_SetFrecuencia(freq, waveType);     //
Transmitimos el valor de salida
          startMillis = currentMillis;
        }
      }
    }
  }
}
```

```
// ************************** Funciones
**************************

// ----------------------------- DAC1220 -------------------------
--

void Comienzo_DAC1220() // Incluye el reset
{
  SPI.end();
  delay(25); // Esperamos para asegurarnos de que el reloj de 2.5
MHz alcance la frec correcta
  digitalWrite(CSPin, LOW);     // pin 7 a LOW
  delayMicroseconds(5);

  //Patrón de reset del reloj
  pinMode(SCKPin, OUTPUT);
  digitalWrite(SCKPin, LOW);
  delay(1);

  digitalWrite(SCKPin, HIGH);
  delayMicroseconds(240); // Primer periodo alto (600 clocks),
txin = 0.4 ms
  digitalWrite(SCKPin, LOW);
  delayMicroseconds(5);
  digitalWrite(SCKPin, HIGH);
  delayMicroseconds(480); // Segundo periodo alto (1200 clocks)
  digitalWrite(SCKPin, LOW);
  delayMicroseconds(5);
  digitalWrite(SCKPin, HIGH);
  delayMicroseconds(960); // Tercer periodo alto (2400 clocks)
  digitalWrite(SCKPin, LOW);
  delay(1);

  pinMode(SCKPin, INPUT);

  // Inicio comunicación
  delay(1000);
  SPI.begin();
  SPI.setDataMode(SPI_MODE1);
  SPI.setClockDivider(SPI_CLOCK_DIV128); // 125 KHz
  SPI.setBitOrder(MSBFIRST);
  delay(500);
}

void EscribeTension(float valor) {
  uint32_t code;

  if (valor < 0) {
    valor = 0;
  } else if (valor > 10) {
    valor = 10;
  }

  //code = (uint32_t)(valor / 5 * 0xFFFFF); // Esta seria la recta
teórica de regresión
```

```
  code = (uint32_t)((valor + 0.0833) / (5 * 2.5 * 0.9984) *
0xFFFFF); // Calculada experimentalmente
  if (code > 0xFFFFF)
  {
    code = 0xFFFFF;
  }
  EscribeSalida(code << 4);
}

void EscribeTensionTermopar(float valor) {
  uint32_t code;

  if (valor < 0) {
    valor = 0;
  } else if (valor > 5) {
    valor = 5;
  }

  code = (uint32_t)(valor / 5 * 0xFFFFF); // Esta seria la recta
teórica de regresión

  if (code > 0xFFFFF)
  {
    code = 0xFFFFF;
  }
  EscribeSalida(code << 4);
}

void EscribeSalida(uint32_t code)
{
  digitalWrite(CSPin, LOW);
  delayMicroseconds(5);

  SPI.transfer(command_dato);
  delayMicroseconds(7); // Necesitamos esperar un tiempo min de
5.2 us
  SPI.transfer((code & 0x00FF0000) >> 16);
  SPI.transfer((code & 0x0000FF00) >> 8);
  SPI.transfer((code & 0x000000FF));

  delayMicroseconds(5);
  digitalWrite(CSPin, HIGH);
  delayMicroseconds(10); // Necesitamos esperar un tiempo min de
8.8 us
}

void DAC1220_Normal() // Cambiamos al modo Normal para utilizarlo
{
  digitalWrite(CSPin, LOW);
  delayMicroseconds(5);

  SPI.transfer(0b00100100); // Escribir 2 bytes en el reg de
comandos
  delayMicroseconds(7);     // Necesitamos esperar un tiempo min
de 5.2 us
  SPI.transfer(0b00100000); // Iniciamos correctamente le primer
byte del reg de comandos
  SPI.transfer(0b10110000); // Entramos en modo Normal
```

```
  delayMicroseconds(5);
  digitalWrite(CSPin, HIGH);
}

void DAC1220_Sleep() // Cambiamos al modo Sleep cuando no lo
utilizemos
{
  SPI.end();
  SPI.begin();
  SPI.setDataMode(SPI_MODE1);
  delay(50);

  digitalWrite(CSPin, LOW);
  delayMicroseconds(5);

  SPI.transfer(0b00100100); // Escribir 2 bytes en el reg de
comandos
  delayMicroseconds(7);     // Necesitamos esperar un tiempo min
de 5.2 us
  SPI.transfer(0b00100000); // Iniciamos correctamente le primer
byte del reg de comandos
  SPI.transfer(0b10110010); // Entramos en modo Sleep

  delayMicroseconds(5);
  digitalWrite(CSPin, HIGH);
}

// ------------------------- Termopares -------------------------
--

float InterpolacionJ (int temp) // Ecuaciones de interpolación
para el tipo J
{
  float valor;

  if (temp >= -200 && temp <= -100)
    valor = -0.0000007 * temp * temp - 0.00001 * temp + 0.0475;

  else if (temp >= -99 && temp <= 120)
    valor = 0.0000000005 * temp * temp * temp - 0.0000003 * temp *
temp + 0.00006 * temp + 0.0504;

  else if (temp >= 121 && temp <= 485)
    valor = 0.0000005 * temp + 0.0552;

  else if (temp >= 486 && temp <= 750)
    valor = 0.00000003 * temp * temp - 0.000006 * temp + 0.0510;

  else if (temp >= 751 && temp <= 840)
    valor = 0.000005 * temp + 0.0603;

  else if (temp >= 841 && temp <= 1049)
    valor = -0.00003 * temp + 0.0894;

  else if (temp >= 1050 && temp <= 1200)
    valor = -0.000005 * temp + 0.0636;

  return valor;
}
```

```
float InterpolacionK (int temp) // Ecuaciones de interpolación
para el tipo K
{
  float valor;

  if (temp >= -270 && temp <= 0)
    valor = 0.00000000008 * temp * temp * temp - 0.0000003 * temp
* temp + 0.00006 * temp + 0.0396;

  else if (temp >= 1 && temp <= 120)
    valor = 0.000000002 * temp * temp * temp - 0.0000007 * temp *
temp + 0.00007 * temp + 0.0393;

  else if (temp >= 160 && temp <= 690)
    valor = - 0.00000003 * temp * temp + 0.00003 * temp + 0.0356;

  else if (temp >= 691 && temp <= 1372)
    valor = - 0.000000007 * temp * temp + 0.000002 * temp +
0.0438;

  return valor;
}

float InterpolacionE (int temp) // Ecuaciones de interpolación
para el tipo E
{
  float valor;

  if (temp >= -270 && temp <= -71)
    valor = - 0.0000006 * temp * temp + 0.00002 * temp + 0.0535;

  else if (temp >= -70 && temp <= 220)
    valor =  - 0.0000001 * temp * temp + 0.0001 * temp + 0.0584;

  else if (temp >= 221 && temp <= 599)
    valor = - 0.00000007 * temp * temp + 0.00007 * temp + 0.0634;

  else if (temp >= 600 && temp <= 1000)
    valor = - 0.000000000006 * temp * temp * temp - 0.0000000008 *
temp * temp - 0.0000008 * temp + 0.0828;

  return valor;
}

float InterpolacionT (int temp) // Ecuaciones de interpolación
para el tipo T
{
  float valor;

  if (temp >= -270 && temp <= -230)
    valor = - 0.0000007 * temp * temp - 0.0001 * temp + 0.0257;

  else if (temp >= -229 && temp <= -1)
    valor = 0.0000000001 * temp * temp * temp - 0.00000007 * temp
* temp + 0.0001 * temp + 0.0389;

  else if (temp >= 0 && temp <= 400)
```

```
    valor = 0.00000000006 * temp * temp * temp - 0.0000001 * temp
* temp + 0.00009 * temp + 0.0386;

  return valor;
}

// ---------------------------- AD421 ----------------------------
--

void Comienzo_AD421()
{
  SPI.end();
  SPI.begin();
  SPI.setDataMode(SPI_MODE0);
  //SPI.setClockDivider(SPI_CLOCK_DIV128); // 125 KHz  // Ya se
definen estos parámetros
  //SPI.setBitOrder(MSBFIRST);                         // en el
DAC1220
  delay(50);
  digitalWrite (LATCHPin, HIGH);
  delay(10);
  digitalWrite (LATCHPin, LOW);
  delay(10);
}

void EscribeCorriente(float valor)
{
  long code;

  if (valor < 4) {
    valor = 4;
  } else if (valor > 20) {
    valor = 20;
  }

  //code = (long)((valor - 4) * 0xFFFF / 16); // Esta seria la
recta teórica de regresión
  code = (long)((valor - 4) * 0xFFA8 / 16 - 15); // Calculada
experimentalmente

  if (code < 0x0000)
  {
    code = 0x0000;
  }
  else if (code > 0xFFE0) // Se ha hallado que a más de este valor
la salida supera los 20mA
  {
    code = 0xFFE0;
  }

  Escribe_AD421(code);
}

void Escribe_AD421(long dato)
{
  SPI.transfer(highByte(dato));
  SPI.transfer(lowByte(dato));

  digitalWrite (LATCHPin, HIGH);
```

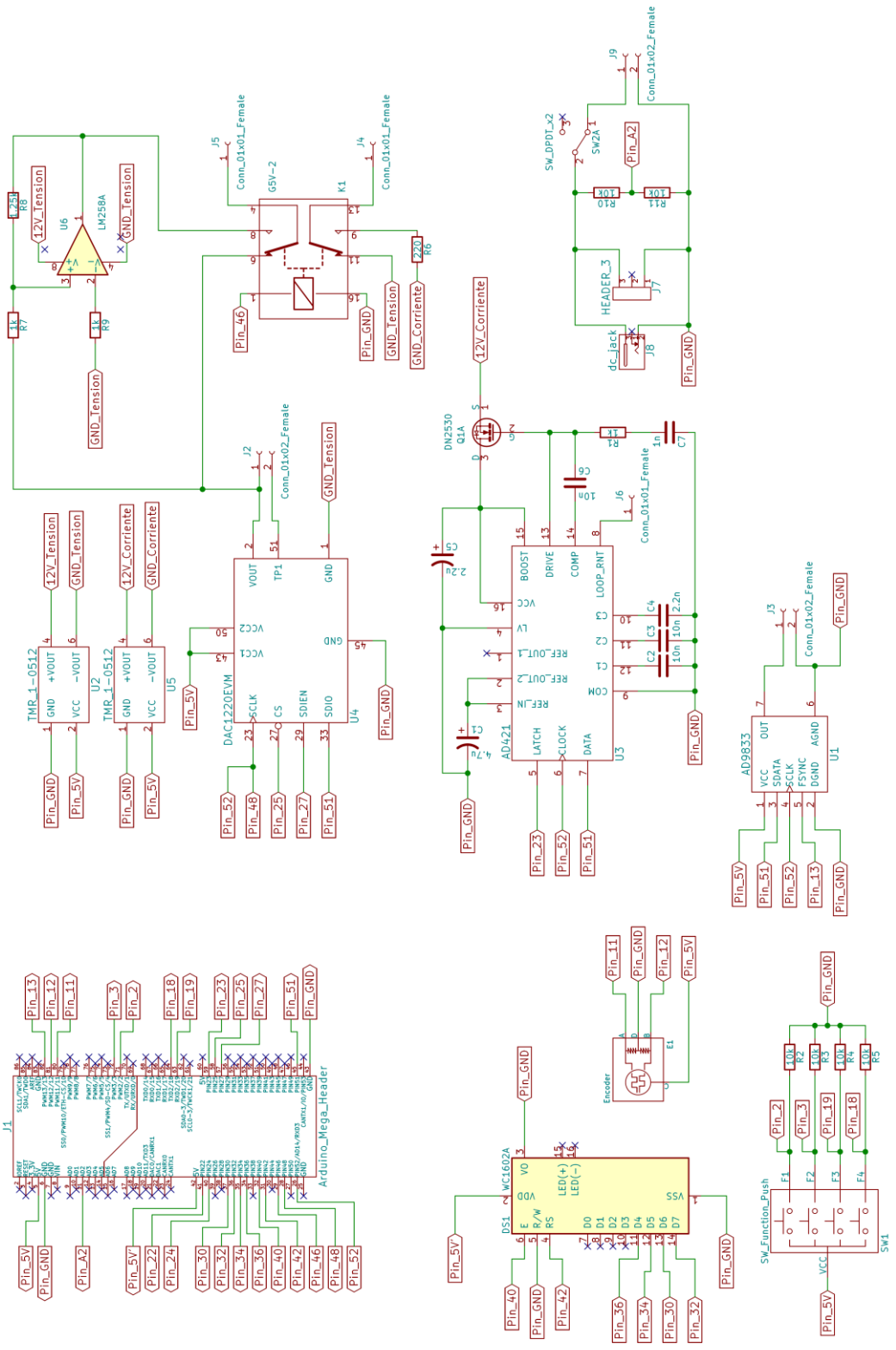
```
  delay(10);
  digitalWrite (LATCHPin, LOW);
  delay(10);
}

// --------------------------- AD9833 ----------------------------
--

void AD9833_Reset()
{
  SPI.end();
  SPI.begin();
  SPI.setDataMode(SPI_MODE2);
  delay(50);

  AD9833_EscribeRegistro(0x100);   // Escribimos '1' en el bit D8
del registro de control
  delay(10);
}

void AD9833_SetFrecuencia(long frequency, int Waveform)
{
  long FreqWord = (frequency * pow(2, 28)) / refFreq; // Recta
teórica de regresión

  int MSB = (int)((FreqWord & 0xFFFC000) >> 14);    //Solo se usan
los 14 bits mas bajos para el dato
  int LSB = (int)(FreqWord & 0x3FFF);

  // Establecemos los bits 15 y 14 a 0 y 1 respectivamente para
que el registro de frecuencia sea 0
  LSB |= 0x4000;
  MSB |= 0x4000;

  AD9833_EscribeRegistro(0x2100);
  AD9833_EscribeRegistro(LSB);          // Escribimos los 16 bits
mas bajos en los registros
  AD9833_EscribeRegistro(MSB);          // Escribimos los 16 bits
mas altos en los registros
  AD9833_EscribeRegistro(0xC000);       // Registro de fase
  AD9833_EscribeRegistro(Waveform);     // Salimos y cambiamos a
la forma SINE, SQUARE o TRIANGLE
}

void AD9833_EscribeRegistro(int dat)
{
  digitalWrite(FSYNCPin, LOW);      // Ponemos la señal FSYNC a
LOW antes de escribir en los registros
  delayMicroseconds(10);            // Esperamos a que el AD9833
este listo para recibir datos

  SPI.transfer(highByte(dat));      // Cada registro del AD9833 es
de 32 bits, pero la transferencia
  SPI.transfer(lowByte(dat));       // de 16 bits tiene que
hacerse en 2 escrituras de 8 bits

  digitalWrite(FSYNCPin, HIGH);     //Una vez completada la
escritura, cambiamos FSYNC a HIGH
}
```

```
// ---------------------- Interrupciones ----------------------
--

void interruptCountF1()
{
  if (millis() > timeCounter + timeThreshold)
  {
    ISRCounterF1++;
    ISRCounterF2 = -1;
    ISRCounterF3 = -1;
    ISRCounterF4 = -1;
    ISRCounterF1 = ISRCounterF1 % 5;    // F1 tiene 5 submodos
    timeCounter = millis();
  }
}

void interruptCountF2()
{
  if (millis() > timeCounter + timeThreshold)
  {
    ISRCounterF2++;
    ISRCounterF1 = -1;
    ISRCounterF3 = -1;
    ISRCounterF4 = -1;
    ISRCounterF2 = ISRCounterF2 % 3;    // F1 tiene 3 submodos
    timeCounter = millis();
  }
}

void interruptCountF3()
{
  if (millis() > timeCounter + timeThreshold)
  {
    ISRCounterF3++;
    ISRCounterF1 = -1;
    ISRCounterF2 = -1;
    ISRCounterF4 = -1;
    ISRCounterF3 = ISRCounterF3 % 4;    // F1 tiene 4 submodos
    timeCounter = millis();
  }
}

void interruptCountF4()
{
  if (millis() > timeCounter + timeThreshold)
  {
    ISRCounterF4++;
    ISRCounterF1 = -1;
    ISRCounterF2 = -1;
    ISRCounterF3 = -1;
    ISRCounterF4 = ISRCounterF4 % 6;    // F1 tiene 6 submodos
    timeCounter = millis();
  }
}

// ----------------------- Display LCD -------------------------
--
```

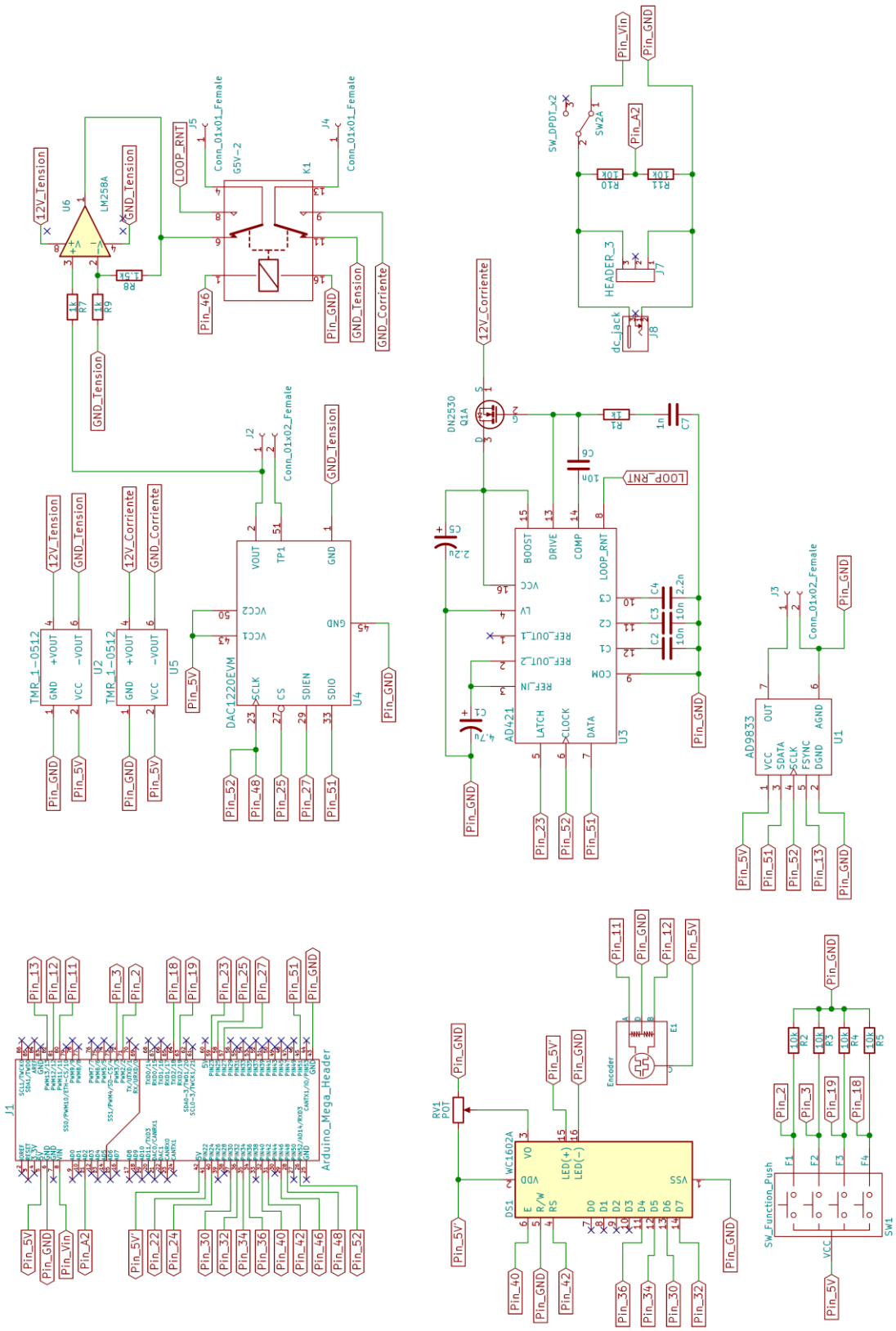
```
int cifrasFrec(unsigned long variable)     // Cuenta las cifras de
un dato
{                                          // unsigned long
  unsigned long num = 0;
  int contador = 0;

  num = variable;
  while (num / 10 > 0)
  {
    num /= 10;
    contador++;
  }
  return contador;
}

int cifrasTer(int variable)    // Cuenta las cifras de un dato int
{
  int num = 0, contador = 0;

  if (variable >= 0) num = variable;
  else num = -variable;

  while (num / 10 > 0)
  {
    num /= 10;
    contador++;
  }
  return contador;
}

// ------------------------- Batería ----------------------------
--

void ComprobarBateria()    // Si el nivel de la batería se
encuentra
{                          // por debajo de 3.5V, se muestra un
aviso
  int alerta = 0;          // en el display

  if ((analogRead(PinBateria) < BateriaBaja) && (alerta == 0))
  {
    lcd.setCursor(7, 0);
    lcd.write(byte(6));
    lcd.write(byte(7));

    alerta = 1;
  }

  if ((analogRead(PinBateria) > BateriaBaja) && (alerta = 1))
  {
    lcd.setCursor(7, 0);
    lcd.print("  ");

    alerta = 0;
  }
}
```

Anexo B: Esquemáticos del circuito de la PCB

Primera versión del circuito



Segunda versión del circuito



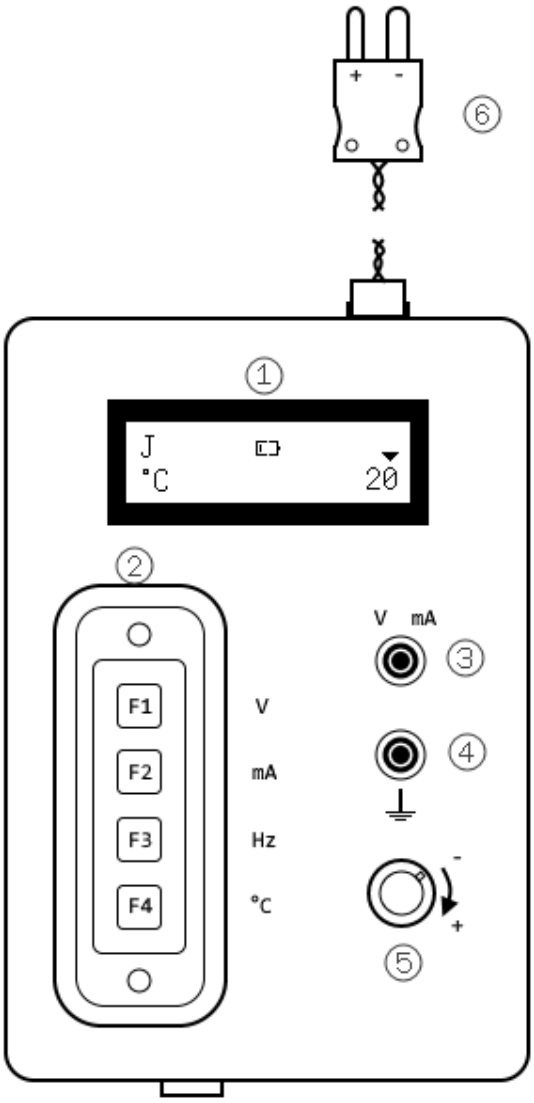
Anexo C: Manual de uso

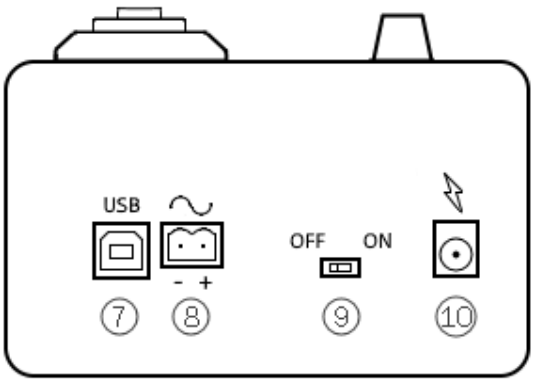
El calibrador de procesos permite generar múltiples señales eléctricas, de forma que en la salida se puede mantener una tensión, intensidad o frecuencia de gran precisión, así como simular el comportamiento de varios tipos de termopares, como son J, K, E y T.

Tabla de especificaciones

Especificaciones	
Tensión	
Rango	4.0mV, 10.0000V
Resolución	0.1mV
Precisión	±(0.09% + 0.1mV)
Corriente	
Rango	4.00mA, 20.00mA
Resolución	0.01mA
Precisión	±1%
Carga máxima	250Ω
Termopares	
Tipos	J, K, E, T
Rango	J: -210ºC, +1200ºC
	K: -270ºC, +1372ºC
	E: -270ºC, +1000ºC
	T: -270ºC, +400ºC
Resolución	1ºC (todos los tipos)
Precisión	±2ºC (todos los tipos)
Frecuencia	
Rango	3Hz, 50000Hz
Resolución	1Hz
Precisión	±(0.1% + 0.1Hz)
Temperatura de funcionamiento	0ºC, 45ºC
Alimentación	Batería ion-litio, 2250mAh
Consumo	188mA – 261mA (en función de la salida)
Duración estimada en funcionamiento	8h 30m, 12h (en función de la salida)
Duración estimada de carga	< 2h 30m

Diagrama de partes del calibrador





1. Pantalla LCD
2. Botones de función
3. Salida positiva de tensión y corriente
4. Masa en las salidas de tensión y corriente
5. Encoder
6. Conector de termopares
7. Conector USB del Arduino
8. Salida en frecuencia
9. Interruptor ON/OFF
10. Puerto de carga

Instrucciones de funcionamiento

El calibrador de procesos se enciende mediante el interruptor ON/OFF, de forma que en la posición izquierda permanece apagado, y en la derecha encendido. Una vez encendido, el calibrador entrará directamente en el modo de generación de tensión, correspondiente a la tecla de función F1. Para cambiar entre los cuatro modos de funcionamiento basta con pulsar una vez la tecla de función correspondiente a ese modo en concreto, siempre y cuando no sea ya el seleccionado.

La pantalla LCD muestra en la esquina inferior izquierda la unidad de medida correspondiente al modo actual de funcionamiento, de forma que en el modo de generación de tensión aparecerán “V”, en corriente “mA”, en termopares “ºC” y en frecuencia “Hz”. En los modos de simulación de termopar y frecuencia figurarán además en la esquina superior izquierda el submodo actual, de forma que en los termopares se encontrarán las letras “J”, “K”, “E” o “T”, y en frecuencia los símbolos de onda senoidal, cuadrada o triangular. El valor de la salida que se esté generando en cada instante se mostrará en la esquina inferior derecha, figurando encima de la cifra que se modifica al rotar el encoder una flecha. Esta flecha puede moverse cambiando de submodo en los modos de funcionamiento correspondientes.

Pantallas de carga y función F1

El modo F1 corresponde a la generación de tensión entre las salidas 3 y 4 del diagrama anterior. La tensión a generar puede cambiarse entre 0.0004V y 10.000V. El valor de la salida se cambia mediante la rotación del encoder,

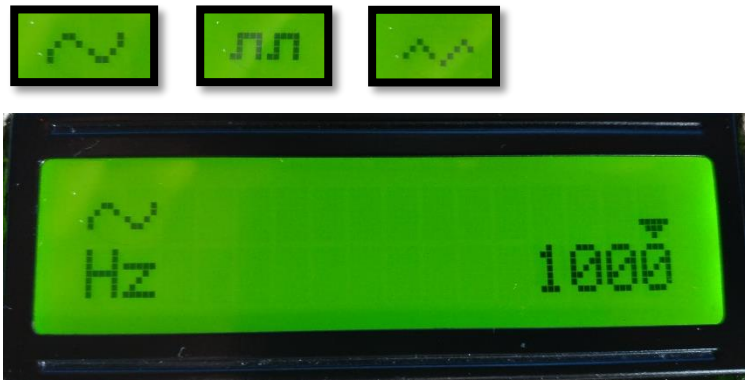
sumando en el sentido de las agujas del reloj y restando en el contrario. La cifra que se modifica con el encoder viene indicada por una flecha encima de dicha cifra, pudiendo cambiarse en una cifra a la izquierda pulsando una vez en la tecla de función F1. De esta forma, cuando se inicia el modo la flecha indica la cifra más a la derecha (resolución de 0.1mV). Al pulsar una vez F1, la flecha cambia a la posición inmediatamente a la izquierda (resolución de 1mV), y al pulsar otra vez más se mueve otra posición (resolución de 0.01V). Cuando la resolución se encuentre en 1V, al pulsar F1 la resolución vuelve a la inicial, 0.1mV. Cuando se entra a este modo de funcionamiento desde otro o por primera vez, la pantalla muestra un mensaje de "Cargando..." durante unos segundos, ya que los componentes involucrados en la generación de tensión necesitan un tiempo para inicializarse correctamente.

Pantalla de función F2

El modo F2 sirve para generar una corriente controlada. Los terminales de salida 3 y 4 se deben conectar a un circuito de forma que se forme un lazo de auto-alimentación, de forma que la corriente indicada en el calibrador será la que circule por ese lazo. La carga máxima a conectar es de 250Ω. Se puede alimentar el lazo con una tensión de hasta 293V, con el polo positivo en el borne negro de la salida, aunque esta tensión no aumenta la carga máxima que se puede conectar. El rango de corriente que se puede controlar es de entre 4mA y 20mA, el estándar de comunicaciones. Al igual que en modo de generación de tensión, para cambiar la cifra de la salida que se modifica con el encoder basta con pulsar la tecla de función de este modo, es decir F2. Las resoluciones permitidas son de 0.01mA, 0.1mA y 1mA.

Pantalla de función F3, termopar tipo J

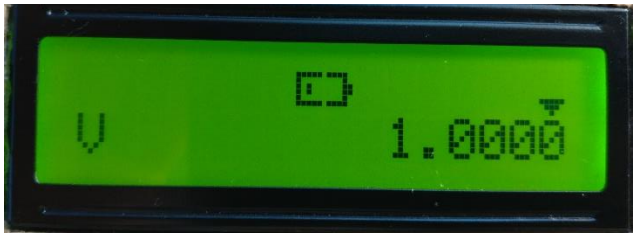
El modo F3 es el de simulación de termopares. En los polos del conector específico para esta función se genera una tensión correspondiente a la que generaría un termopar (con la unión fría a 0ºC) si se encontrase a la temperatura que se indique en la pantalla del calibrador. Esta temperatura solo puede incrementarse o disminuirse una unidad con cada posición del encoder. El pulsar la tecla F3 cuando se encuentra en este modo cambia el tipo de termopar simulado, mostrándose en la esquina superior izquierda de cual se trata en cada caso. Al cambiar de termopar la temperatura simulada cambia a 0ºC. Al igual que en el caso de generación de tensión, cuando entramos en este modo se muestra un mensaje de "Cargando..." mientras se inicializan los componentes pertinentes.



Pantalla de función F4, detalles de los símbolos de onda senoidal (izquierda), cuadrada (centro), triangular (derecha)

El último modo, el F4, corresponde a la generación de frecuencia por el conector 8 específico para este fin. El pulsar la tecla de función F4 cuando ya se encuentra el calibrador en este modo puede cambiar tanto la cifra que se modifica de la salida como el tipo de onda generado. De esta forma, cuando se entra en la función se genera una onda senoidal y la resolución es de 1Hz. Al pulsar F4, se cambia a una resolución de 1KHz. Al pulsar una segunda vez, retornamos a la resolución de 1Hz, pero la forma de la onda ahora es cuadrada.

De ahí se vuelve a cambiar la resolución a 1KHz manteniendo la forma de onda. Las dos últimas pulsaciones antes de volver al primer submodo corresponden a la onda triangular, con las resoluciones de 1Hz y 1KHz. Cambiar de formas de onda no cambia el valor de la frecuencia de salida.



Pantalla de función F1, con el aviso de batería baja

El último elemento que puede aparecer en la pantalla es un aviso de batería baja, que se muestra en la parte central superior cuando la carga de la batería desciende por debajo del mínimo de tensión de alimentación recomendable de los componentes. Cuando se muestra este mensaje, conviene conectar el calibrador al cargador si no se quiere que el instrumento se apague.

Advertencias:

- No conectar dos polos a tensiones diferentes, puede generar un cortocircuito y dañar el instrumento.
- El calibrador no cuenta con ningún tipo de certificado de protección IP, por lo que no es conveniente utilizarlos en ambientes con mucho polvo, ni tampoco mojarlo, ya que se podría dañar el instrumento.
- No conectar el conector de termopares a un termómetro con conexión a la red eléctrica mientras el calibrador está siendo cargado, porque podría generarse un cortocircuito y dañar el instrumento.
- Mantener el interruptor en la posición de apagado si se conecta el Arduino interno del calibrador mediante el puerto USB, porque podría generarse un cortocircuito en el circuito de alimentación y dañar el instrumento.