Francisco Guerrero López

Diseño e Implementación de mejoras en un Ionómetro Automático

TRABAJO DE FIN DE GRADO

Dirigido Eduard Llobet Valero

Grado en Ingeniería Electrónica Industrial y Automática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2015

A la empresa NT Sensor SL, en especial a Cristina Cid y Josep Maria Cid por toda la ayuda recibida durante la realización del proyecto.

ÍNDICE GENERAL

1. OBJETIVOS Y PRELIMINARES

1.1. Resumen

Este Proyecto de final de Grado ha sido realizado en la empresa NT Sensors, SL, una empresa que se dedica a poner en el mercado un nuevo tipo de sensores analíticos basados en la tecnología de nanotubos de carbono. Entre sus productos se encuentra el AMI C7, con el que nosotros trabajaremos, un instrumento totalmente automatizado para la determinación simultánea de varios iones en aguas, aguas residuales, medios de cultivo, hidropónicos, reactores de algas…

Nuestro trabajo es dotar este producto de total independencia ya que el AMI C7 necesita de un software de ordenador para funcionar, por lo tanto en primer lugar lo que debemos hacer es eliminar la dependencia de un ordenador personal para que nuestro producto funcione. Por otro lado, debemos introducir en nuestro AMI C7 una salida analógica 4-20 mA, ya que en la empresa reciben muchas demandas del sector industrial preguntando por la posibilidad de obtener esta señal en nuestro producto. I por último, incorporaremos a nuestro AMI C7 un Arduino UNO, que en este primer prototipo simplemente servirá como apoyo para el microcontrolador PIC18F4550, que utiliza nuestro ionómetro, y nos permitirá implementar una gran variedad de mejoras en nuestro diseño, en esta caso incorporaremos una tarjeta SD donde se guardaran todos los datos registrados por el AMI C7.

En esta memoria veremos una descripción de la empresa con la que hemos realizado este proyecto, nos introduciremos en los electrodos selectivos de iones (ISE), que es el tipo de sensor que utilizamos para detectar las concentraciones de iones, y en el mundo de Arduino, explicaremos que es Arduino y porque hemos escogido esta herramienta para nuestro proyecto.

Veremos también el diseño realizado para generar nuestra salida 4-20 mA, en el cual destaca el uso del chip XTR110, un convertidor de tensión a corriente con entrada de 0 a 5 voltios y salida de 4 a 20 mA, veremos los componentes utilizados durante la realización del proyecto, y todo el software que nuestro AMI C7 lleva introducido tanto en el microcontrolador PIC18F4550 como en el Arduino, que consigue que ambos microcontroladores se comuniquen entre ellos mediante una comunicación digital USART y nos permite interpretar las lecturas del electrodo selectivo de iones y controlar las bombas, con las cuales llenamos una celda, donde se introducen los líquidos de las dos muestras P1 y P2 para el calibrado del sensor y la muestra a medir y se leen las concentraciones de iones.

1.2. Objetivos

El Objetivo principal, en el cual se basa el proyecto, es la construcción de un prototipo de ionómetro para un sensor ISE (electrodo selectivo de iones) con salida 4-20 mA, con la condición de usar un Arduino para futuras mejoras en el equipo. A partir de este objetivo principal se extraen otros principalmente académicos:

- Comprensión del funcionamiento de un electrodo selectivo de iones y en concreto del sensor utilizado en nuestro proyecto.

- Entender el software (del microcontrolador y del ordenador) y el hardware del producto anterior.

- Comprensión y realización de un nuevo software para el PIC18F4550 y para Arduino además de para la comunicación entre ambos.

- Diseño y realización de una salida 4-20 mA para el uso de nuestro ionómetro en un ámbito industrial.

- Iniciarnos en la plataforma Arduino, valorando la variedad de herramientas que nos ofrece.

- Realizar el montaje correctamente y finalmente hacer las medidas experimentales que se crean pertinentes.

Es necesario especificar que partimos de un ionómetro ya acabado con salida mediante comunicación RS-232 a un ordenador a través del cual interactúa con el usuario mediante un software, el cual también hemos participado en su elaboración.

Como se menciona en los objetivos, para comprobar que el circuito funciona correctamente y que el diseño es el más idóneo, se realizarán tanto durante las diferentes fases de realización del circuito como una vez obtenido el resultado final, las diferentes pruebas, simulaciones y cálculos correspondientes y necesarios.

Finalmente se realizaran las pruebas que se estimen oportunas para demostrar que se han cumplido todos los objetivos y el prototipo funciona correctamente.

2. INTRODUCCIÓN

2.1. NT SENSORS

NT Sensors, SL se creó en noviembre de 2008 con una finalidad muy concreta: poner en el mercado un nuevo tipo de sensores analíticos basados en la tecnología de nanotubos de carbono inicialmente desarrollada en el grupo de Quimiometría, Cualimetria y Nanosensores de la URV.

En NT Sensors, SL se producen estos sensores electroquímicos para la detección de iones en muestras acuosas y se desarrollan soluciones concretas para cada aplicación (multielectrodos, pequeños volúmenes de muestra, detección de patógenos).

Este tipo de sensores potenciométricos tienen una serie de características que los distingue de los ISEs clásicos: no contienen solución interna y, por tanto, se pueden miniaturizar más fácilmente y son más robustos, además de operar en cualquier posición, no solamente la vertical. Además, pueden utilizarse para detectar iones y otras muchas especies químicas como proteínas o incluso microorganismos. Este hecho, el poner en el mercado una serie de productos basados en un nuevo desarrollo científico-tecnológico, hace que NT Sensors sea una empresa de base tecnológica.

Los sensores presentan, además de la misma fiabilidad y calidad que tienen los que actualmente se pueden encontrar en el mercado, una mejor robustez y durabilidad con un coste más reducido. También se abren nuevos segmentos de mercado, con la determinación de nuevos iones y moléculas, con electrodos miniaturizados adaptados a pequeños volúmenes de muestra, y con la agrupación de electrodos para la medida simultánea de diferentes iones presentes en una muestra.

2.1.1. Productos y aplicaciones

En NT Sensor trabajan para intentar adaptar las cualidades de sus productos a las necesidades de cada uno de sus posibles clientes, albergando así aplicaciones muy diferentes, por lo que tanto ionómetros, como sensores son idóneos para una gran variedad de campos: Agricultura, medio ambiente, hidroponía, enología, laboratorio, industrias, control de calidad, depuradoras, golf, dureza del agua, desalinizadoras, piscifactorías…

A continuación veremos solamente algunos de sus productos más destacados:

Combi ION (Electrodo combinado):

Formato clásico de electrodo combinado. Integra un electrodo selectivo de iones y un electrodo de referencia, ambos de estado sólido. El formato ideal para el análisis de una especie iónica de forma rápida, sencilla y eficaz.

La combinación entre las propiedades ofrecidas por la potenciometría y las características diferenciales de los ISEs basados en nanotubos de carbono, hacen que el

Combi ION sea la solución perfecta para el análisis del contenido de un determinado ión en una muestra de base acuosa.

Características: Estado completamente sólido, mantenimiento cero, robusto, fácil manejo, insensibles a especies redox o a oxígenos, medidas en cualquier posición/orientación, mínimo consumo de muestras/reactivos y conector tipo BNC estándar.

Electrodos miniaturizados:

La miniaturización real de los electrodos abre las puertas a analizar volúmenes muy reducidos de muestra. Ésta propiedad es muy valorada en distintas aplicaciones en el mundo de la investigación. Los pequeños volúmenes necesarios requeridos combinados al carácter no destructivo de la técnica, suponen un valor añadido a éste tipo de sensores.

Electrodo de media celda de estado sólido y dimensiones reducidas. De cuerpo plástico y con conexión bañada en oro. Las propiedades que los nanotubos de carbono confieren a nuestros sensores permiten a su vez la miniaturización de estos, conservando las buenas prestaciones en cuanto a sencillez de uso y robustez de funcionamiento.

Características: Estado completamente sólido, mantenimiento cero, robusto, fácil manejo, compatible con otros ionómetros/phmetros, insensibles a cambios de intensidad de luz, insensibles a especies redox o a oxígenos, medidas en cualquier posición/orientación, trabaja junto a un electrodo de referencia.

Sonda modular Multi ION:

NT Sensors presenta a nivel mundial el sistema Multi Ion, un sistema modular que permite cuantificar simultáneamente la concentración de diferentes iones en una misma muestra. Este innovador concepto permite hacer realidad la medida multiparamétrica de 6 iones simultáneamente. Un producto que ofrece una solución simple, eficaz y rápida que permite ahorrar en reactivos y tiempo de análisis. La sonda Multi Ion puede configurarse para analizar desde 2 hasta 6 diferentes iones, en un mismo diámetro, de tan sólo 25 mm.

Características: estado completamente sólido, sin mantenimiento, robusto, fácil manejo, configurable hasta seis iones, sustitución de electrodos desgastados, medidas simultaneas de los seis iones, sin pretratamiento de la muestra, conector M12 multipin.

Ionómetros para computadora:

Ionómetros diseñados para la simplificación de los análisis rutinarios mediante electrodos selectivos de iones, electrodos de pH, con la última tecnología existente en el mercado de plataformas y comunicaciones.

Los modelos compactos y económicos, ofrecen gran versatilidad, permiten monitorización en continuo y personalización de hasta cinco puntos de calibrado etc.

Los ionómetros están diseñados para facilitar el análisis al usuario, sacando el máximo rendimiento a los electrodos selectivos.

Ligeros, de dimensiones reducidas y totalmente portátiles gracias a su alimentación vía USB. Funcionan junto a un completo e intuitivo software para PC. Sus características y prestaciones lo convierten en ideal tanto para usuarios inexpertos como para los más exigentes.

AMI C7:

Instrumento de proceso de alta precisión para determinación simultanea de varios iones en aguas, aguas residuales, medios de cultivo, hidropónicos, reactores de algas…

Configuración disponible hasta un máximo de 7 parámetros (Ca2+, Cl-, K+, Na+, NH4+, NO3- y pH).

Hasta 4 puntos de toma de muestra, no requiere preparación de la misma ya que la medida no se ve afectada por la coloración o turbidez de la muestra.

La medición se realiza mediante un portasondas configurado con electrodos sensibles y selectivos a la especie iónica.

El analizador está equipado con un armario resistente a la intemperie para instalación en interiores o exteriores. Se configura en función del tipo de punto de toma de muestra, ya sea a presión atmosférica o con presión positiva.

Totalmente automatizado: Calibración, medidas y limpieza.

Extensas funciones de autodiagnóstico, Manejo y uso sencillo, montaje en pared, en rail o en soporte en interiores o exteriores y consta de configuraciones adicionales bajo demanda.

2.2. Electrodo Selectivo de Iones (ISE)

Básicamente existen dos tipos de conducción eléctrica: conductividad en metales, en los cuales la electricidad es transportada por electrones, y conductividad en líquidos con iones disueltos (electrolitos), en donde la electricidad es transportada por los iones.

En cualquier proceso electroquímico (celda galvánica, celda electrolítica, etc.), podemos observar ambos tipos de transporte eléctrico. El lugar en donde el metal y el electrolito interactúan para transferir la carga eléctrica de uno a otro, es llama interface metal-líquido. Este tipo de interface es la que sucede en los electrodos.

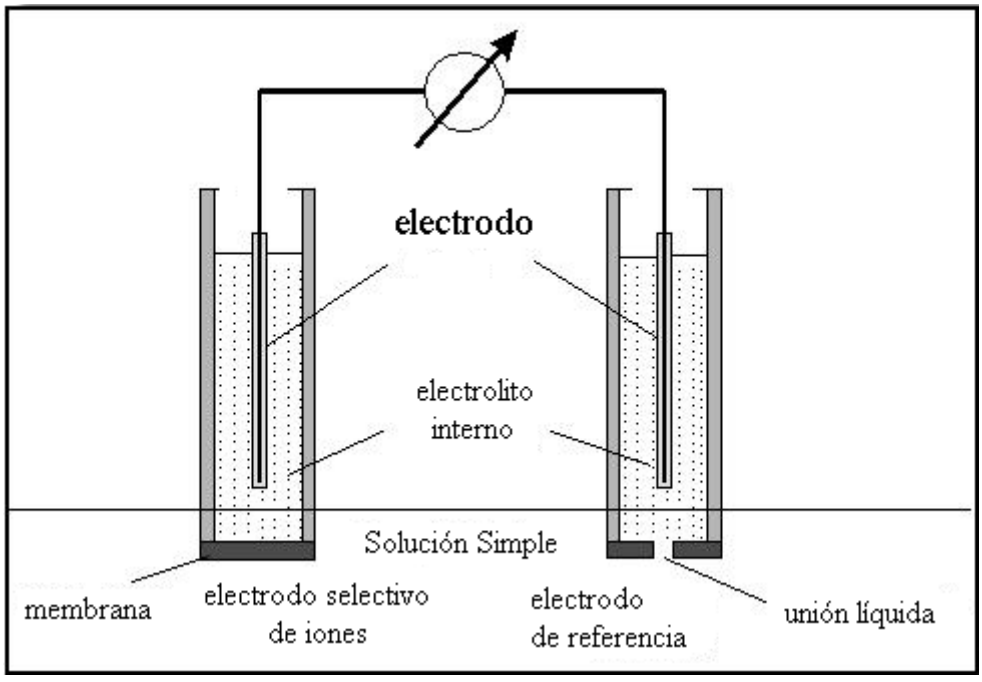
Por ejemplo, en una celda galvánica cobre-plata, en uno de los electrodos ocurre un proceso de oxidación: el electrodo de cobre se oxida, liberando dos electrones y formando iones de cobre, mientras que en el otro electrodo, sucede un proceso de reducción, los iones de plata disueltos toman esos electrones, se reducen y se depositan átomos de plata metálica sobre el electrodo.

Los electrodos selectivos de iones (ISE), son electrodos que se utilizan para medir la concentración de un determinado ion en un electrolito. Para hacerlo, mide la diferencia de potencial causada por el contacto del electrodo con el ion en cuestión, respecto de la diferencia de potencial en el electrodo de referencia.

Estos electrodos selectivos de iones poseen una membrana selectiva, que sólo responde al contacto con un determinado ion disuelto en la solución, y la diferencia de potencial generada a cada lado de la membrana se utiliza para medir la concentración del ion en la solución estudiada.

El electrodo selectivo de iones debe estar inmerso en la solución acuosa que contiene el ion que se desea medir, y en la misma solución estará también inmerso el electrodo de referencia. Para completar el circuito electroquímico, ambos electrodos se conectan a un mini voltímetro, muy sensible, usando cables especiales de baja interferencia. Cuando el ion a medir atraviesa la membrana selectiva del electrodo, debido al gradiente de concentración, genera una diferencia de potencial que es medida. A mayor diferencia de potencial generada, mayor es la concentración del ion en la solución.

En la figura siguiente observamos un esquema del circuito que acabamos de describir:



El electrolito interno en el cual está inmerso el electrodo, tiene una concentración conocida del ion que se pretende medir. En la membrana selectiva, que está en contacto con el ion a medir disuelto en la solución del problema y con el electrolito interno de "referencia", se genera una diferencia de potencial, que tiene relación directa con la concentración de dicho ion en la solución medida, de acuerdo con la ecuación de Nernst.

2.2.1. Ecuación general de potenciometría directa

En el método de potenciometría directa se establece la relación que existe entre el potencial de un electrodo selectivo en función de la concentración de el ion para el cual él es activo. Es decir que se trata de un método que permite obtener "directamente" el valor de concentración a partir de una lectura de potencial.

Desde el punto de vista analítico, para un anión se cumple que:

$$E = E^{o'} + \frac{0{,}059}{n} p[anión]$$

Donde Eº' contiene una serie de constantes que están relacionadas a la construcción de los electrodos y no corresponde a un potencial estándar de reducción, puesto que no ocurre este proceso en este tipo de electrodos. El parámetro n corresponde a la carga del anión sensado.

De igual manera para un catión se cumple:

$$E = E^{o'} - \frac{0{,}059}{n} p[catión]$$

Como se aprecia, las ecuaciones tienen un aspecto similar a la ecuación general de la potenciometría (de hecho se derivan del mismo modo), esta es, la ecuación de Nernst. Por lo que se espera que obtenga una relación lineal entre el potencial del electrodo y la función "p" para el anión o el catión.

Ecuación de Nernst:

$$E = E^0 - \frac{RT}{nF} \ln(Q)$$

Donde:

- *E* es el potencial corregido del electrodo.
- *E*0 el potencial en condiciones estándar (los potenciales se encuentran tabulados para diferentes reacciones de reducción).
- *R* la constante de los gases.
- *T* la temperatura absoluta (escala Kelvin).
- *n* la cantidad de mol de electrones que participan en la reacción.
- *F* la constante de Faraday (aproximadamente 96500 C/mol).
- *Ln(Q)* es el logaritmo neperiano de *Q* que es el cociente de reacción.

2.2.2. Comportamiento "Nernstiano"

De acuerdo a la ecuación general obtenida para los electrodos selectivos:

$$E = E^{o'} \pm \frac{0{,}059}{n} p[analito]$$

Se espera que la respuesta en potencial de estos electrodos produzca una línea recta si se la representa en función del p [analito]. Existirá un rango de concentraciones (normalmente de 0,1M a 10^{-5}M) donde se cumple lo anterior, se dice que en esta zona el sistema tiene un comportamiento "tipo Nernstiano".

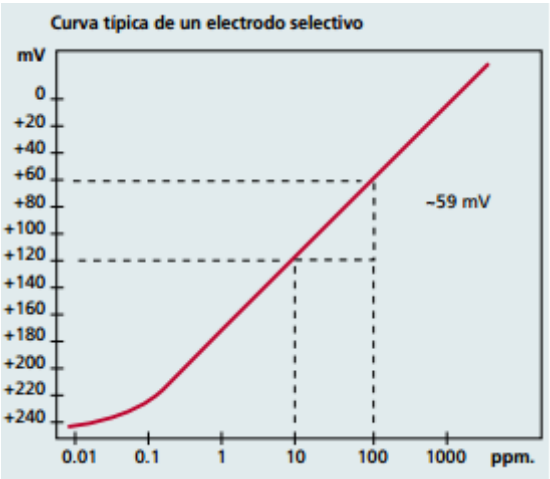
Este comportamiento varía con el tiempo de uso del electrodo, por lo que es recomendable verificar esta relación, especialmente si el electrodo no ha sido utilizado recientemente.

2.2.3. Variables a considerar en una medida con ISE

2.2.3.1. Escala de medida

Es la escala de concentración del ion a medir, en la cual el electrodo es "sensible" a una variación de la misma. Puede dividirse en dos zonas: Zona lineal y zona no lineal.

Siempre es recomendable trabajar en la zona lineal, ya que la zona no lineal exige efectuar una calibración con al menos 5 patrones y una interpolación manual.



2.2.3.2. Electrodo de referencia

La función del electrodo de referencia es proporcionar un potencial constante frente al que poder medir las variaciones debidas al electrodo indicador.

Es muy importante escoger el electrolito de referencia adecuado, para ello es necesario tener en cuenta:

- La fuerza iónica del electrolito debe ser muy superior a la de la muestra.
- Debe ser una solución equitrasferente. Las velocidades del catión y del anión deben ser lo más parecidas posibles.

- No debe reaccionar con la muestra.
- No debe contaminar la muestra. Nunca debe contener el ion a medir.

2.2.3.3. Temperatura

De todos es conocido el efecto de la temperatura sobre las medidas de potencial. El comportamiento de los electrodos selectivos frente a la temperatura no es tan conocido como el de los de pH, por esta razón no se habla de compensación de temperatura en las medidas con ISE.

Es necesario que la temperatura permanezca constante durante un análisis con electrodo selectivo, tanto durante la calibración con patrones como durante la medida de las muestras.

2.2.3.4. Fuerza iónica

El potencial de un electrodo selectivo responde a la actividad de los iones, no a su concentración.

Para que la actividad de un ion y su concentración se parezcan, se añade tanto a los patrones como a las muestras un ajustador de fuerza iónica, ISA.

Un ajustador de fuerza iónica, ISA, es una disolución de fuerza iónica elevada que no interfiere con la muestra y que iguala la fuerza iónica de patrones y muestras.

2.2.3.5. Interferencias del electrodo

Si en la muestra hay presentes iones que afecten al electrodo selectivo, éste responderá tanto a los iones de interés como a los interferentes. Por lo tanto es necesario garantizar que no hay especies interferentes antes de iniciar un análisis.

2.2.3.6. Tipos de membranas selectivas de iones

Existen cuatro tipos de membranas selectivas de iones que se utilizan en electrodos selectivos de iones (ISEs): vidrio, estado sólido, líquidas, y de electrodos compuestos.

Membranas de vidrio:

Las membranas de vidrio se fabrican con un tipo de vidrio para intercambio de iones (silicato o anfígeno). Este tipo de ISE tiene una buena selectividad, pero solo para algunos cationes con cargas simples; principalmente H^+, Na^+, y Ag^+. El vidrio de anfígeno también posee selectividad para iones de metal con cargas dobles, tales como Pb^{2+}, y Cd^{2+}. La membrana de vidrio posee una excelente durabilidad química y puede funcionar en medios muy agresivos. Un ejemplo muy común de este tipo de electrodo es el electrodo de pH de vidrio.

Membranas cristalinas:

Las membranas cristalinas se fabrican como mono- o poli cristalinas de una única substancia. Poseen una buena selectividad, porque solo iones que logran penetrar en la estructura cristalina pueden interferir con la respuesta del electrodo. La selectividad de membranas cristalinas puede ser tanto para catión y anión de la substancia que forma la membrana. Un ejemplo es el electrodo selectivo de fluoruro basado en cristales de LaF_3.

Membranas de resinas de intercambio de iones:

Las resinas de intercambio iónico se basan en membranas orgánicas especiales de polímero que contienen una substancia específica de intercambio de iones (resina). Este es el tipo más difundido de electrodo selectivo de iones. El uso de resinas específicas permite preparar electrodos selectivos para decenas de iones diferentes, tanto con un solo átomo como con varios átomos. Además son los electrodos más populares y difundidos con selectividad aniónica. Sin embargo, este tipo de electrodos poseen una baja durabilidad química y física lo que resulta en una vida útil corta. Por ejemplo el electrodo selectivo de potasio basado en valinomicina como agente de intercambio iónico.

2.2.4. Ventajas y desventajas de los electrodos selectivos

Ventajas:

- Respuesta rápida

- Ensayo no destructivo

- Posibilidad de diseños adaptables

- No importa color de la muestra

Desventajas:

- Contaminación del electrodo

- Interferencias

- Vida útil limitada

2.2.5. Aplicaciones habituales

Una de las aplicaciones de la técnica de potenciometría directa, es en el área de química clínica, en donde se logra medir la concentración de iones y gases en sangre de pacientes, gracias a aparatos en los cuales están instalados varios electrodos selectivos de iones, uno para cada parámetro que se desea medir. Los iones más frecuentemente medidos en el ionograma sérico, son sodio, potasio, calcio y cloro, ya que son los más importantes en el equilibrio electrolítico del organismo.

Los electrodos selectivos de iones se utilizan también en campos como:

Agricultura: Nitrato, potasio, calcio y cloruro en suelo, además de otros iones.

Ambiente: Control analítico de cianuro, fluoruro sulfuro en efluentes, aguas naturales, etc.

Alimentos: Nitrato y nitrito en alimentos base de carnes; determinación de cloruros; fluoruros en agua, bebidas etc.; calcio en la leche; potasio en jugo de frutas

Análisis clínicos: Sodio, Calcio, potasio, cloruro en suero, Plasma, Fluoruro en estudios dentales…

Se usan también en investigaciones en el ámbito de la bioquímica y biofísica, donde es preciso conocer las concentraciones iónicas en una solución acuosa, por lo general en tiempo real.

2.3. Arduino

2.3.1. ¿Qué es Arduino?

Arduino es una herramienta para hacer que ordenadores u otros instrumentos puedan detectar y controlar más del mundo físico que un simple equipo de escritorio. Es una plataforma de computación física de código abierto basado en una simple placa electrónica con microcontrolador, y un entorno de desarrollo para escribir el software del microcontrolador.

Arduino se puede utilizar para desarrollar objetos interactivos, tiene gran variedad de entradas para diferentes interruptores o sensores, y nos permite el control de multitud de luces, motores, y otras salidas físicas. Los proyectos con Arduino pueden ser independientes, o pueden comunicarse con un software que se ejecuta desde el ordenador. Las tablas se pueden montar a mano o comprarse preensambladas.

El lenguaje de programación Arduino es una implementación de Wiring, una plataforma computación física similar, que se basa en el entorno de programación multimedia Processing.

2.3.2. ¿Por qué Arduino?

Hay muchos otros microcontroladores y plataformas de microcontroladores disponibles para computación física: Parallax Basic Stamp, de Netmedia BX-24, Phidgets, Handyboard del MIT, y muchos otros ofrecen una funcionalidad similar. Todas estas herramientas toman los detalles de la programación de microcontroladores y lo envuelven en un paquete fácil de usar. Arduino también simplifica el proceso de trabajar con microcontroladores, pero ofrece algunas ventajas para los profesores, estudiantes y aficionados interesados sobre otros sistemas:

- Asequible: Las placas Arduino son relativamente baratas en comparación con otras plataformas de microcontroladores. La versión menos costosa del módulo Arduino puede ser montada a mano, e incluso los módulos de Arduino premontados cuestan menos de 30 euros.

- Multiplataforma: El software de Arduino se ejecuta en los sistemas operativos Windows, Macintosh OSX y Linux. La mayoría de los sistemas de microcontrolador se limitan a Windows.
- Entorno de programación simple y claro: El entorno de programación de Arduino es fácil de usar para los principiantes, pero lo suficientemente flexible para los usuarios avanzados que también pueden disfrutar de muchas ventajas. Se basa convenientemente en el entorno de programación Processing, por lo que los estudiantes que estén aprendiendo a programar en ese entorno estarán familiarizados con la apariencia de Arduino.
- Código software abierto y extensible: El software de Arduino está publicado como herramientas de código abierto, disponible para la extensión por programadores experimentados. El idioma se puede ampliar a través de bibliotecas de C ++, y la gente con ganas de entender los detalles técnicos pueden dar el salto de Arduino a el lenguaje de programación AVR C, en el que se basa. Del mismo modo, puede agregar código AVR-C directamente en sus programas de Arduino.
- Hardware extensible: El Arduino se basa en los microcontroladores Atmel's ATmega8 y ATmega168. Los planos de los módulos están publicados bajo una licencia de Creative Commons, por lo que los diseñadores de circuitos experimentados pueden hacer su propia versión del módulo, ampliándolo y mejorándolo. Incluso los usuarios con poca experiencia pueden construir la versión tablero del módulo con el fin de entender cómo funciona y ahorrar dinero.

2.3.3. Productos Arduino

Arduino tiene una amplia gama de productos oficiales, entre los cuales están los productos principales, módulos (elementos se pueden conectar a otros productos para darles características extra), kits…

Estos productos se pueden diferenciar por las siguientes características:

Nivel Principiante:

Son los más adecuados para empezar con Arduino, fáciles de usar y ya están preparados para cualquier uso que queramos darles. Estas tarjetas y módulos son los mejores para comenzar a aprender y juguetear con la electrónica y la codificación.

ARDUINO UNO

ARDUINO PRO

ARDUINO MICRO

ARDUINO NANO

ARDUINO Starter Kit

ARDUINO SHIELD MOTOR

Funciones mejoradas:

Estos son Arduinos más complejos que permiten funcionalidades avanzadas o actuaciones rápidas.

ARDUINO MEGA 2560

ARDUINO CERO

ARDUINO DUE

ARDUINO PROTO SHIELD

Relacionados con internet:

Con cualquiera de estos Arduinos se pueden crear dispositivos conectados fácilmente a la red de internet, lo cual ofrece una gran variedad de oportunidades creativas y de funcionalidad.

ARDUINO Yun

ARDUINO ETHERNET SHIELD

ARDUINO SHIELD GSM

ARDUINO SHIELD 101 WIFI

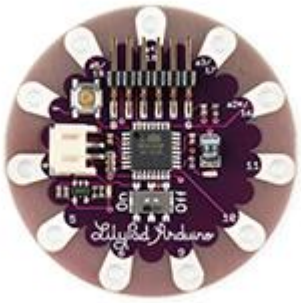
Para textiles:

Estos arduinos son ideales para proyectos en los que sea necesario unir la electrónica junto a productos textiles. Aunque por su tamaño también pueden ser ideales para otros proyectos personales en los que se requiera de un arduino de dimensiones algo más reducidas.

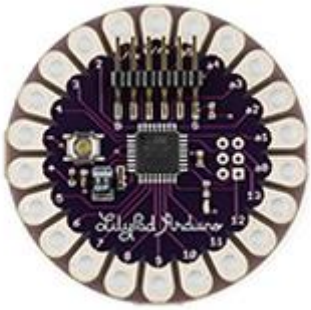
ARDUINO GEMMA

LILYPAD SIMPLE ARDUINO

LILYPAD ARDUINO



LILYPAD USB ARDUINO



Impresión 3D:

El enfoque de Arduino para la impresión 3D está representado por Materia 101, una impresora que le permite comenzar a experimentar con esta tecnología increíble de la forma más fácil.

MATERIA 101

Otros:

La historia de Arduino empieza sobre el 2006, desde entonces además de todos los productos que ya hemos visto han ido surgiendo también gran cantidad de placas, accesorios, kits, etc. que alcanza desde proyectos muy específicos y no tan comunes,

como características que pueden ser necesarias según el proyecto que estemos realizando o el entorno para el que esté pensado. Algunos de estos Arduinos son:

Arduino Mega ADK, Arduino Ethernet, Arduino Robot, Arduino Leonardo, Arduino Esplora, Arduino Mini, Arduino Fio, Arduino Wifi Shield, Arduino USB Host Shield, Arduino Wireless SD Shield, Arduino Wireless Proto Shield, Arduino LCD Screen, Arduino USB Serial Light Adapter, Arduino Mini USB Serial Adapter, Arduino ISP…

3. MEMORIA DESCRIPTIVA

3.1. Descripción de la solución adoptada

En la parte práctica de este proyecto hemos diseñado, montado y probado una salida 4-20 mA para un ionómetro en concreto, al cual hemos incorporado un Arduino y una tarjeta SD.

Nuestro diseño consta de dos partes muy diferenciadas. La primera parte englobaría lo que es todo el software tanto del microcontrolador que utilizamos en el potenciómetro, PIC18F4550, como del propio Arduino, consiguiendo que estos interactúen entre ellos, para lo cual se ha utilizado una comunicación digital UART. La segunda parte comprende todo lo referente al diseño y la realización del hardware para la salida 4-20 mA, del cual habría que destacar el chip XTR110 que convierte un voltaje de entre 0 a 5 voltios a nuestra salida de 4 a 20 mA.

Cabe mencionar que previamente se ha estudiado y entendido todo el funcionamiento del ionómetro del cual partíamos, y se ha modificado gran parte del firmware de este, inclusive hardware, para lograr el producto final que pretendíamos conseguir.

3.2. Descripción de los elementos

3.2.1. Alimentación

En nuestro prototipo hemos incorporado un convertidor de corriente alterna a 24 V de corriente continua para poder alimentar nuestro potenciómetro desde la red eléctrica.

La alimentación de gran parte de los componentes electrónicos de nuestro ionómetro es de 5 V, las bombas se alimentan con una tensión de 3.3 V, la placa Arduino con una tensión de almenos entre 7 a 12 V y para alimentar el lazo de corriente del señal de salida 4-20 mA utilizamos la tensión de 24 V que obtenemos del convertidor AC/DC.

Para conseguir la tensión de alimentación de 5 V de gran parte de los componentes del ionómetro se ha incorporado a la placa un LDO con una tensión de entrada de entre 7 a 35 V y una salida de 5 V, este es el NCP7805TG, con lo cual pasamos de los 24 V en la salida del convertidor AC/DC a 5V.

Con otro LDO, el MCP1825S-3002E/AB, conseguimos convertir estos 5 V en los 3.3 V necesarios para alimentar las bombas.

I por último utilizamos otro convertidor para obtener los 12 V del Arduino.

3.2.2. Filtrado de señal y Multiplexado

Para cada uno de los 8 canales por los que recibimos el valor de la diferencia de potencial en los electrodos selectivos correspondientes a cada uno de los iones a medir, se realiza un filtrado de la señal. Después se multiplexaran con tal de obtener como única salida el canal seleccionado. Esto se consigue en diferentes etapas:

Para conseguir que la lectura del electrodo sea los más fiable posible, en primer lugar, se debe garantizar una impedancia de entrada mínima y en segundo lugar se debe filtrar

el máximo de ruido posible. Esto se consigue a través de un seguidor de tensión mediante el integrado LMC6484.

Posteriormente, se realiza un filtrado con un filtro RC de R = 10 kohms y C = 0.22 µF, y un nuevo seguidor de tensión mediante el integrado OPA4277.

A continuación, cada canal se conecta por una de las diferentes entradas del multiplexor DG408DJ.

3.2.3. Conversión AD (Analógico→Digital) y Microcontrolador

Este módulo tiene la función de convertir la señal eléctrica dada por el sensor y filtrada, en un valor digital de 24 bits, que recibirá el microcontrolador. Está basada en el chip AD7712.

Utilizamos el microcontrolador PIC18F4550-I/P como el principal cerebro de nuestro ionómetro, a través del software que se muestra más adelante, este se encarga de recibir el valor digital del convertidor A/D correspondiente a las medidas realizadas por los electrodos selectivos, realiza los cálculos pertinentes para la realización de la calibración del propio ionómetro y la obtención del valor exacto de las concentraciones de iones, dirige la secuencia de activación de las bombas y envía la información final al Arduino a través de una comunicación por el puerto serie, USART asíncrona.

3.3. Comunicación USART

El USART (universal synchronous asynchronous receiver transmitter) es uno de los dos puertos series de los que disponen algunos microcontroladores. Puede funcionar de forma síncrona (half duplex) o asíncrona (full duplex).

3.3.1. Modo asíncrono:

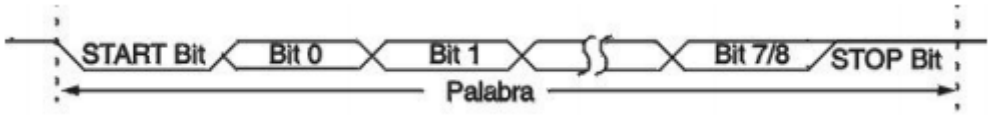
No se necesita una conexión para la señal de reloj, los relojes del transmisor y del receptor son independientes, aunque deben de tener la misma frecuencia, la sincronización entre ambos se hace añadiendo unos bits adicionales (bit de inicio y bit de parada) al byte de datos, que puede estar formado por 8 ó 9 bits. La comunicación puede llegar a ser hasta dúplex completo (bidireccional simultanea). Este tipo de conexión es la que se utiliza normalmente para comunicar un PIC con un Ordenador o para comunicar dos PIC'S entre sí.

Modo full-duplex (bidireccional). Utiliza los pines:

• RC6/TX/CK: transmisión (salida).

• RC7/RX/CK: recepción (entrada).

Los datos enviados tienen tamaño de byte. En el formato de la trama se añade un bit de Start = 0 y un bit de Stop = 1, y puede añadirse un noveno bit de datos (ejemplo bit de paridad) a los 8 bits del dato:



Esta forma de comunicar serie usa la norma RS-232 / RS-485. Los bits se transmiten a una frecuencia fija y normalizada.

Los bloques que configuran la USART en modo asíncrono son:

• Circuito de muestreo.

• Generador de baudios.

• Transmisor asíncrono.

• Receptor asíncrono.

La USART no soporta la generación de paridad por hardware. En modo asíncrono la USART se para al entrar el micro en modo SLEEP.

3.3.1.1. Generador de Baudios:

Para generar la velocidad de transmisión existe un temporizador dedicado de 8-bits (BRG), con funcionamiento dedicado para la USART. La velocidad en baudios se controla mediante el registro SPBRG y las siguientes fórmulas:

TABLA: Fórmulas de cálculo de los baudios.

SYNC	BRGH = 0 (Baja velocidad)	BRGH = 1 (Alta velocidad)
0	(Asíncrono) Baudios = Fosc/(64(X+1))	Baudios = Fosc/(16(X+1))
1	(Síncrono) Baudios = Fosc/(4(X+1))	N/A

X = valor en SPBRG (0 a 255)

TABLA: Registros asociados con la generación de los baudios.

Direcc.	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor en: POR, BOR	Valor en otros RESETS
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
99h	SPBRG	*Baud Rate Generator Register*								0000 0000	0000 0000

TABLA: Baudios modo asíncrono con (BRGH = 1)

BAUD (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG valor (decimal)	KBAUD	% ERROR	SPBRG valor (decimal)	KBAUD	% ERROR	SPBRG valor (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG valor (decimal)	KBAUD	% ERROR	SPBRG valor (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

TABLA: Buaudios modo asíncrono con (BRGH = 0)

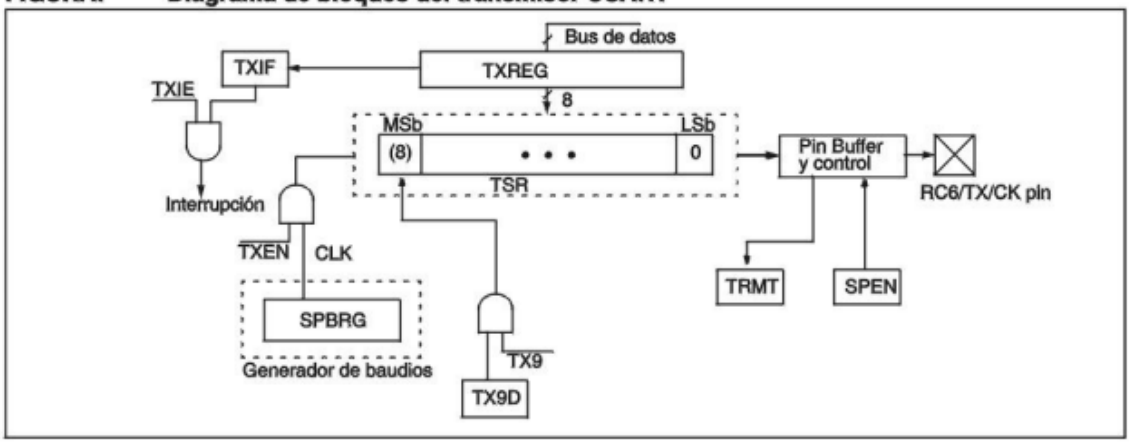
BAUD (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG valor (decimal)	KBAUD	% ERROR	SPBRG valor (decimal)	KBAUD	% ERROR	SPBRG valor (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG valor (decimal)	KBAUD	% ERROR	SPBRG valor (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

3.3.1.2. Transmisor asíncrono

La transmisión se habilita mediante el bit TXEN, TXSTA(5). El registro de transmisión es el TXREG. Para transmitir un dato el software lo escribe en este registro. Después de haber escrito el TXREG el dato pasa al registro de desplazamiento TSR, este registro no se carga hasta que el bit de STOP del dato anterior no se ha transmitido. Al quedar vacío el TXREG se activa el bit de interrupción TXIF (PIR1(4)), habilitado por el bit TXIE (PIE(4)). (TXIF no se desactiva por software, se desactiva sólo cuando se cargan nuevos datos). Hay otro bit el TRMT, TXSTA(1) que muestra el estado del TSR, no produce ninguna interrupción. (Cuando activa TRMT está vacío). Para enviar un dato con 9 bits hay que habilitar el bit TX9, (TXSTA(6)) y poner el que se quiere enviar en TX9D (TXSTA(0)).

FIGURA: Diagrama de bloques del transmisor USART



Registro: TXSTA: Transmitir estado y registro de control (dirección 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Bit 7 → **CSRC**: Selección de la fuente de reloj
Modo asíncrono:
No importa
Modo síncrono:
1 = Modo maestro (reloj generado internamente por BRG)
0 = Modo esclavo (reloj de fuente externa)

Bit 6 → **TX9**: Habilitación transmisión del 9-bit
1 = Selecciona transmisión 9-bit
0 = Selecciona transmisión 8-bit

Bit 5 → **TXEN**: Habilita transmisión
1 = Transmisión habilitada
0 = Transmisión deshabilitada

Registro: TXSTA: Transmitir estado y registro de control (dirección 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Bit 4 → **SYNC**: Selección del modo USART
1 = Modo síncrono
0 = Modo asíncrono
Bit 3 → **No implementado**: Se lee como '0'
Bit 2 → **BRGH**: Selección de velocidad
Modo asíncrono:
1 = Velocidad alta
0 = Velocidad baja
Modo síncrono:
No se utiliza en este modo
Bit 1 → **TRMT**: Bit de estado del registro de desplazamiento de transmisión
1 = TSR vacío
0 = TSR lleno
Bit 0 → **TX9D**: 9th bit de transmisión, puede ser de paridad

Pasos a seguir para implementar la transmisión:

1. Configurar RC6/TX/CK como salida y RC7/RX/DT como entrada.

2. Poner SYNC=0 y SPEN=1, USART en modo asíncrono.

3. Si se desea activar interrupciones activar TXIE=1.

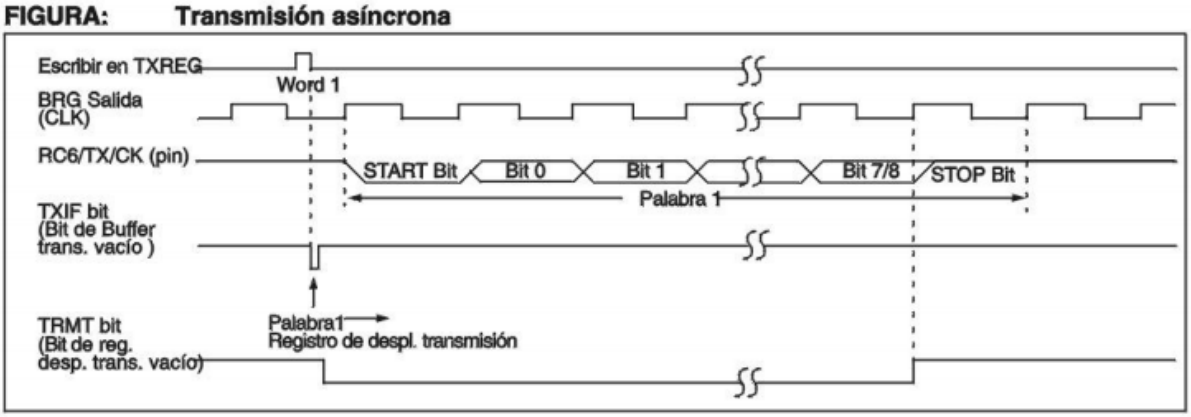
4. Si el dato es de 9 bits TX9=1 y cargar TX9D

5. Cargar X en SPBRG, y elegir BRGH para controlar la frecuencia de trabajo.

6. Activar la transmisión TXEN=1.

7. Cargar en TXREG el dato a transmitir.

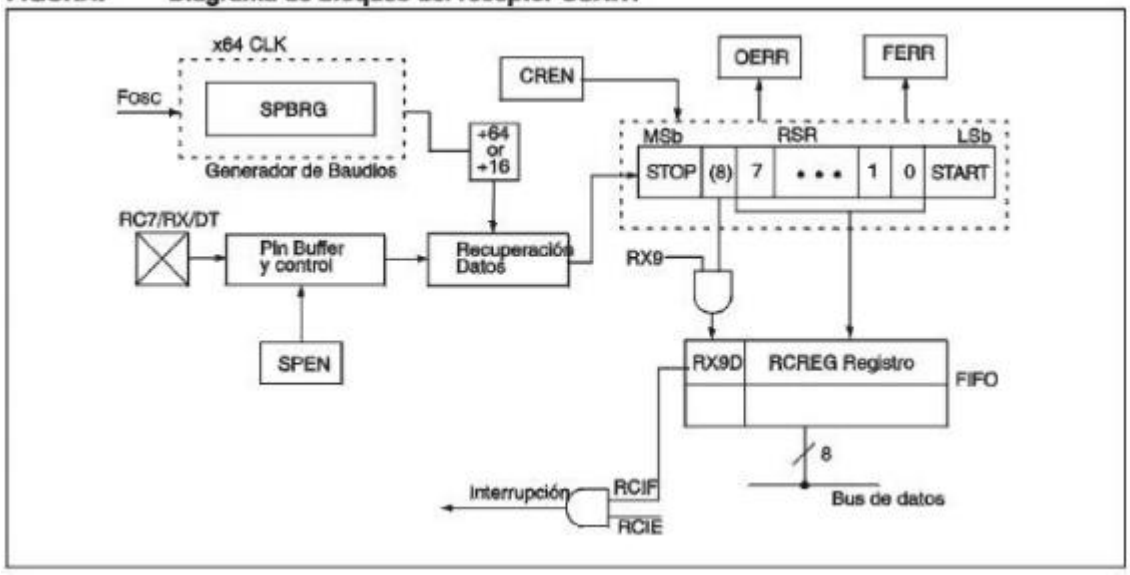
FIGURA: Transmisión asíncrona



Receptor asíncrono

La recepción se habilita mediante el bit CREN, (RCTA(4)). Los datos entran por el pin RC7/RX/DT, llegan hasta el muestreador y se cargan en el registro de desplazamiento RSR de forma serie. Al recibir el bit de STOP, el dato contenido en RSR pasa al registro RCREG si está vacío, y se activa el bit de interrupción RCIF, (PIR1(5)). Habilitada mediante el bit RCIE (PIE1(5)). (RCIF es de sólo lectura y se desactiva por hardware al leer RCREG). El registro RCREG admite dos datos a la espera de ser leídos. Formando un FIFO de dos niveles. Si se reciben tres datos sin que RCREG se lea, el último se pierde. Se produce un error de sobreescritura y hay que reiniciar el receptor. El bit de sobre escritura OERR(RCSTA(1)), se desactiva reseteando el receptor. (CREN=0). El error de encuadre FERR, (RCSTA(2)) se produce si el bit de STOP es un cero. El 9th bit y FERR se cargan a la vez que RCREG, al leer el último dato de RCREG por lo tanto siempre hay que leer el 9th bit y FERR antes de leer RCREG.

FIGURA: Diagrama de bloques del receptor USART



Pasos a seguir para programar la recepción:

1. Configurar RC6/TX/CK como salida y RC7/RX/DT como entrada.

2. Cargar X en SPBRG, y elegir BRGH para controlar la frecuencia de trabajo.

3. Poner SYNC=0 y SPEN=1, USART en modo asíncrono.

4. SI se desea activar interrupciones activar RCIE=1.

5. Si el dato es de 9 bits RX9=1.

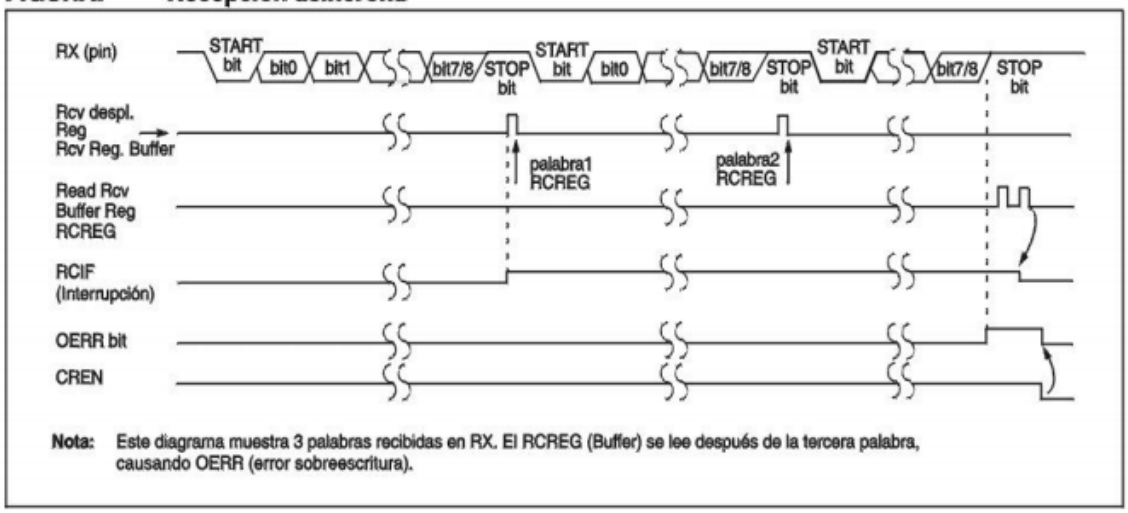
6. Habilitar la recepción con CREN=1.

7. Al completarse la recepción RCIF=1 y produce interrupción si se ha habilitado.

8. Se lee el registro RCSTA y se averigua si se ha producido algún error.

9. Leer el dato de RDREG.

FIGURA: Recepción asíncrona



Nota: Este diagrama muestra 3 palabras recibidas en RX. El RCREG (Buffer) se lee después de la tercera palabra, causando OERR (error sobreescritura).

Registro: RCSTA: Recibir estado y registro de control (dirección 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Bit 7 → **SPEN**: Habilita el puerto serie
1 = Habilita el puerto serie (configura los pines RC7/RX/DT y RC/TX/CK como pines del puerto serie)
0 = Deshabilitado

Bit 6 → **RX9**: Habilita la recepción del 9-bit
1 = Selecciona recepción con 9-bit
0 = Selecciona recepción con 8-bit

Bit 5 → **SREN**: Habilita la recepción sencilla
Modo asíncrono:
No influye
Modo síncrono-maestro:
1 = Habilita la recepción sencilla
0 = Deshabilita la recepción sencilla
Este bit se desactiva después de la recepción
Modo síncrono-esclavo:
No influye

Registro: RCSTA: Recibir estado y registro de control (dirección 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Bit 4 → **CREN**: Habilita la recepción continua
Modo asíncrono:
1 = Habilita la recepción continua
0 = Deshabilita la recepción continua
Modo síncrono:
1 = Habilita la recepción continua
0 = Deshabilita la recepción continua

Bit 3 → **ADDEN**: Habilita la detección de la dirección
Modo asíncrono con 9-bit (RX9 = 1):
1 = Habilita la detección de la dirección, sólo recibe el dato y produce una interrupción de recepción cuando RSR<8> está activo
0 = Deshabilita la detección de la dirección, se reciben todos los bytes, y 9-bit puede usarse para paridad

Bit 2 → **FERR**: Error de encuadre
1 = Error de encuadre (puede actualizarse leyendo RCREG y recibiendo el próximo byte válido)
0 = No hay error de encuadre

Bit 1 → **OERR**: Error de sobreescritura
1 = Error de sobreescritura (puede ser borrado escribiendo un cero en CREN)
0 = No hay error de sobreescritura

Bit 0 → **RX9D**: 9-bit del dato recibido (la paridad debe ser calculada por el software de usuario)

TABLA: Registros asociados con la recepción asíncrona

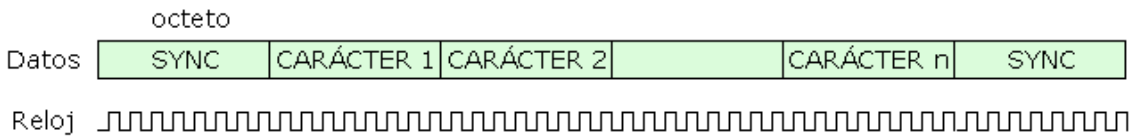
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor en: POR, BOR	Valor en otros RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Recepción								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baudios								0000 0000	0000 0000

- **PIR1** –> Registro para señalizaciones o banderas.
- **RCSTA** –> Registro para el control de la parte receptora de la USART
- **TXREG** –> Registro de datos de transmisión
- **SPBRG** –> Registro para el control del generador de frecuencia, encargado de generar la frecuencia en baudios para realizar la transmisión.
- **TXSTA** –> Registro para el control de la parte de transmisión de la USART
- **PIE1** –> Habilitación de interrupciones
- **RCREG** --> Registro de datos de recepción

3.3.2. Modo síncrono

Necesita una conexión adicional para la señal de reloj. Una USART hace de Master y la otra de esclava. La comunicación es del tipo halfduplex (bidireccional por turnos). Se emplea cuando se quiere comunicar un PIC con otro dispositivo electrónico, como una memoria EEPROM externa.

Es un método más eficiente de comunicación en cuanto a velocidad de transmisión. Ello viene dado porque no existe ningún tipo de información adicional entre los caracteres a ser transmitidos.



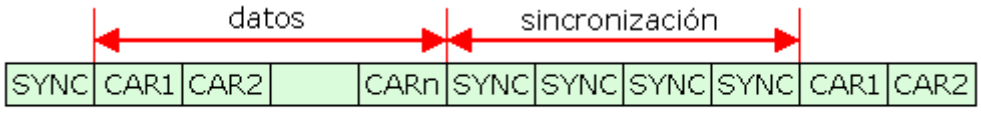
Transmision sincrónica

Cuando se transmite de manera síncrona lo primero que se envía es un octeto de sincronismo ("sync"). El octeto de sincronismo realiza la misma función que el bit de inicio en la transmisión asíncrona, indicando al receptor que va a ser enviado un mensaje. Este carácter, además, utiliza la señal local de reloj para determinar cuándo y con qué frecuencia será muestreada la señal, es decir, permite sincronizar los relojes de los dispositivos transmisor y receptor. La mayoría de los dispositivos de comunicación llevan a cabo una resincronización contra posibles desviaciones del reloj, cada uno o dos segundos, insertando para ello caracteres del tipo "sync" periódicamente dentro del mensaje.

Los caracteres de sincronismo deben diferenciarse de los datos del usuario para permitir al receptor detectar los caracteres "sync". Por ejemplo, el código ASCII utiliza el octeto 10010110.

Existen ocasiones en que son definidos dos caracteres de sincronismo, ello puede ser necesario si, por cualquier motivo el carácter "sync" original se desvirtuara, el siguiente permitirá la reinicialización del receptor. En segundo lugar, puede ocurrir que el equipo receptor necesite un tiempo adicional para adaptarse a la señal entrante.

Cuando se transmite de forma síncrona, es necesario mantener el sincronismo entre el transmisor y el receptor cuando no se envían caracteres, para ello son insertados caracteres de sincronismo de manera automática por el dispositivo que realiza la comunicación.



Inserción automática de caracteres de sincronismo

El receptor/transmisor síncrono debe indicar además cuándo el sincronismo ha sido logrado por parte del receptor.

3.3.2.1. Transmisión y recepción:

La transmisión se activa al escribir en el bit TXEN del registro USARTn_CMD. Se transmitirá cualquier dato grabado en el buffer de TX. Para ello el USART mueve primero los datos al registro de desplazamiento, desde el cual los datos se desplazan hacia el pin TX. Cuando los datos se han movido del buffer de TX, el bit TXBL del registro USARTn_STATUS se activa, lo cual indica que un nuevo byte de transmisión puede ser escrito. Cuando se han transmitido todos los datos de transmisión disponibles (tanto en el registro de desplazamiento como en el buffer de TX), se activa el bit TXC

en el registro USARTn_STATUS. También TXC y TXBL pueden usarse como interrupción.

Cuando el bit RXEN del Registro USARTn_CMD se ha escrito, se aceptarán los datos recibidos en el pin RX. Cuando hay nuevos datos disponibles en el buffer de RX, se activa el bit RXDATAV del registro USARTn_STATUS. Este bit se borrará cuando se lee el búfer RX. El RXDATAV también está disponible como una interrupción.

3.3.3. Diferencias entre el modo asíncrono y el modo síncrono

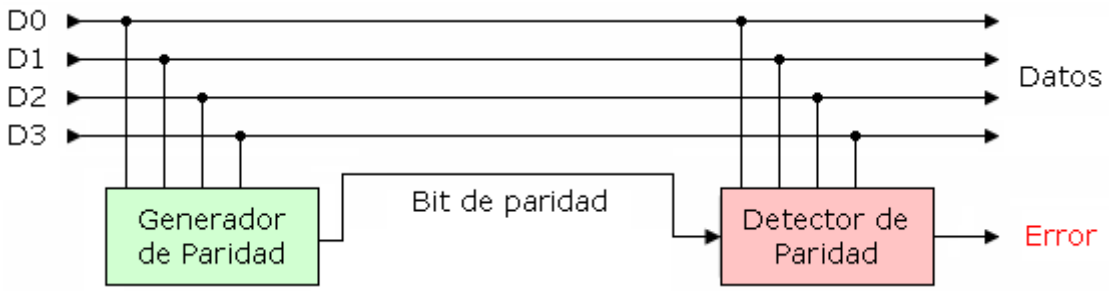
Las diferencias prácticas entre el modo síncrono (que sólo es posible con un USART) y el modo asíncrono (lo cual es posible ya sea con un UART o una USART) pueden resumirse como sigue:

- El modo síncrono requiere tanto datos como un reloj. El modo asíncrono requiere sólo datos.
- En modo síncrono, los datos se transmiten a una velocidad fija. En el modo asíncrono, los datos no tienen que ser transmitidos a una tasa fija.
- Los datos síncronos se transmiten normalmente en forma de bloques , mientras que los datos asíncronos se transmiten normalmente un byte a la vez.
- El modo síncrono permite mayor DTR (velocidad de transferencia de datos) que el modo asíncrono, si todos los demás factores se mantienen constantes.

3.3.4. Generadores y detectores de paridad

Como un error en una transmisión serie solamente suele afectar a un bit, uno de los métodos más comunes para detectar errores es el control de la paridad.

El control de paridad consiste en añadir un bit, denominado de paridad, a los datos que se envían o escriben.



La paridad puede ser par o impar.

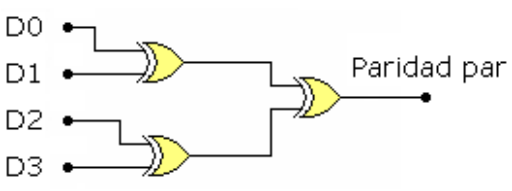
Paridad par:

El bit de paridad será cero, cuando el número de bit "unos" que contienen los datos a transmitir sea un número par, y el bit de paridad será uno cuando los datos que se mandan contienen un número impar de unos.

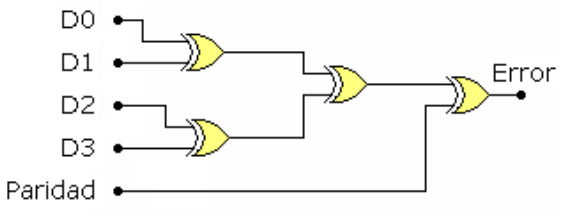
Dato	Paridad
0000 0001	1
0101 0001	1
0101 0101	0
0000 0000	0

La suma de los bits que son unos, contando datos y bit de paridad dará siempre como resultado un número par de unos.

En las siguientes figuras se muestra cómo se puede realizar un generador de paridad y un detector de paridad con puertas lógicas or-exclusivas (EXOR).



Generador de Paridad



Detector de Paridad

Paridad impar:

Dato	Paridad
0000 0001	0
0101 0001	0
0101 0101	1
0000 0000	1

En el sistema de paridad impar, el número de unos (datos + paridad) siempre debe ser impar.

Ejemplo:

Se quieren transmitir los datos C3H y 43H con paridad impar.

- C3H = 1100 0011
- 43H = 0100 0011

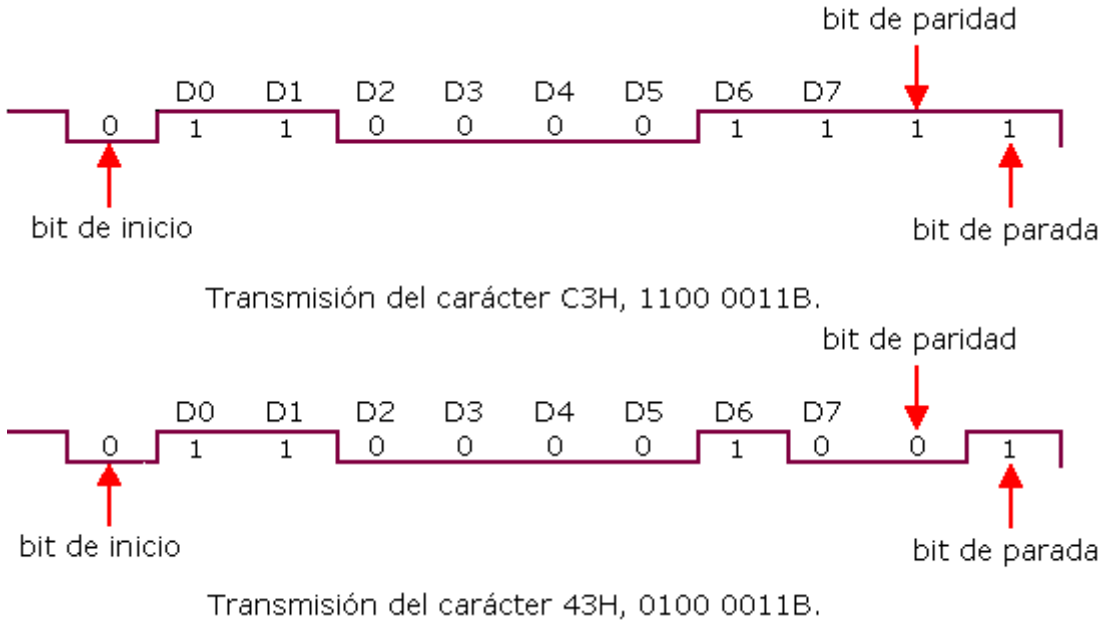
C3H tiene un número par de unos, por lo que el bit de paridad a insertar debe ser 1 para que se cumpla que el número de unos (datos + paridad) siempre debe ser impar:

D0	D1	D2	D3	D4	D5	D6	D7	BIT DE PARIDAD	
1	**1**	0	0	0	0	**1**	**1**	**1**	**5 "unos"**

43H tiene un número impar de unos, por lo que el bit de paridad a insertar debe ser 0 para que se cumpla que el número de unos (datos + paridad) siempre debe ser impar:

D0	D1	D2	D3	D4	D5	D6	D7	BIT DE PARIDAD	
1	**1**	0	0	0	0	**1**	**0**	**0**	**3 "unos"**

La secuencia de transmisión se muestra en la figura siguiente.



El método de detección de errores mediante paridad sólo es válido cuando falla un bit, si por ejemplo fallan dos, no se detectará el error.

3.4. Nuestro Arduino: Arduino UNO

3.4.1. Información General

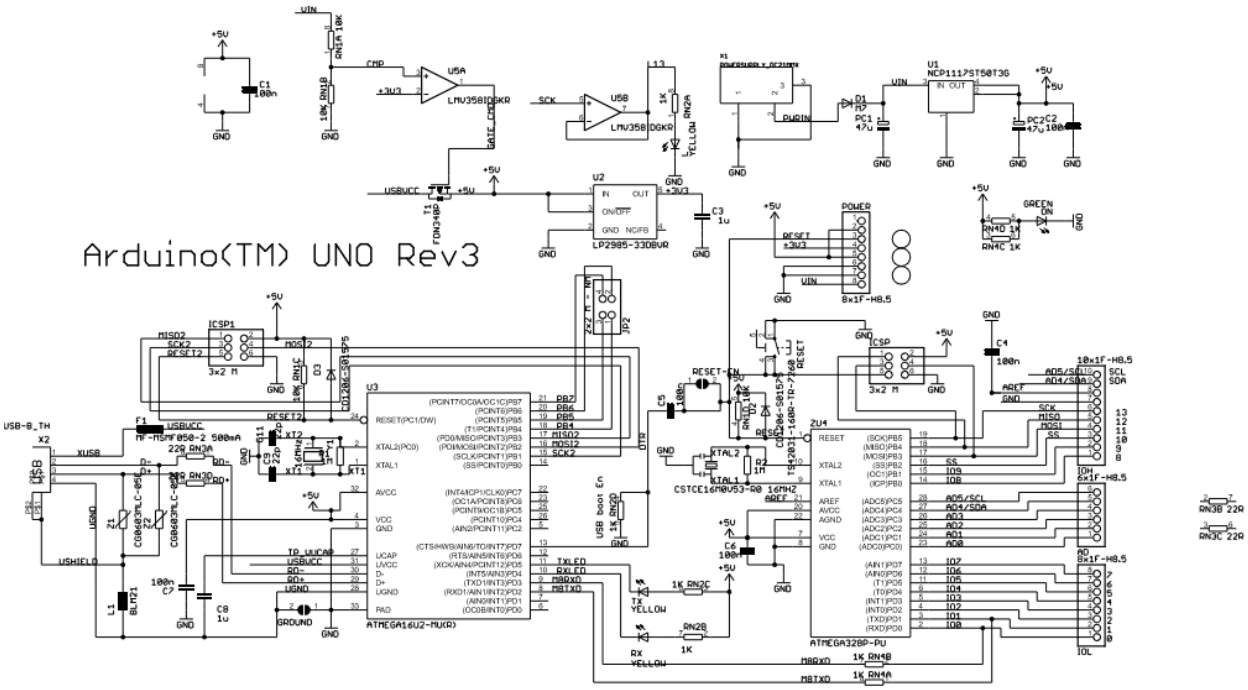
El Arduino Uno es una placa electrónica basada en el microcontrolador ATmega328. Cuenta con 14 pines de entrada / salida digitales (de los cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas, un 16 MHz resonador cerámico, una conexión USB, un conector de alimentación, un header ICSP, y un botón de reinicio. Contiene todo lo necesario para apoyar al microcontrolador; simplemente se debe conectar a un ordenador con un cable USB, a la corriente con un adaptador de CA a CC o a una batería CC para empezar.

El Arduino Uno es el último de una serie de placas Arduino USB y el modelo de referencia para la plataforma Arduino.

3.4.2. Resumen de las especificaciones

Microcontroladores	ATmega328
Tensión de funcionamiento	5 V
Voltaje de entrada (recomendado)	7-12 V
Voltaje de entrada (límites)	6-20 V
Pines digitales E/S	14 (de las cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC en pines E/S	40 mA
Corriente DC del pin 3.3 V	50 mA
Memoria Flash	32 KB (ATmega328) de los cuales 0,5 utilizado por el gestor de arranque
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de reloj	16 MHz
Longitud	68.6 mm
Anchura	53.4 mm
Peso	25 g

3.4.3. Esquemático Arduino UNO Rev 3



3.5. Señal 4-20 mA

Es un estándar analógico, en el cual se establece el valor de 0 (cero) y fondo de escala. La magnitud medida se convierte en una corriente continua proporcional, que se envía por la línea y es detectada en el extremo receptor midiendo la caída de tensión en una resistencia conocida, normalmente de 250 Ω, obteniéndose así tensiones de 1 a 5 V.

Por ejemplo, en nuestro caso, si tuviéramos que medir la concentración de un ion en un rango entre 0 ppm y 100 ppm, se puede establecer el valor de 4 mA para 0 ppm y 20 mA para 100

ppm, esto se hace en la parametrización del equipo, siempre y cuando estos parámetros sean configurables por el usuario.

Otros valores de este estándar, son los valores o señales de error, por ejemplo si la medición esta fuera de rango por defecto algunos equipos generan una señal de 3.6 mA (- 10%) y por el contrario si el valor medido excede el rango, algunos equipos generan una señal de 20.5 mA, otros 21 mA y otros 22 mA, en cualquiera de estos tres casos, son valores que cualquier sistema de medición de proceso interpreta como una señal de error, por estar fuera de rango.

Este tipo de sistema de control se llama bucle de alimentación proporcional. Las señales de corriente análoga también se usan para registrar datos.

Existen varios valores de corriente normalizados: 4-20 mA, 0-5 mA, 0-20 mA, 10-50 mA, 1-5 mA y 2-10 mA; pero de todos ellos el valor de corriente más usado habitualmente es 4-20 mA, adoptado como norma en 1975. Se utiliza un rango 4-20 mA ya que de este modo, una rotura de cable o caída de tensión se detecta al bajar la corriente por debajo de 4 mA y un cortocircuito provoca una subida por encima de 20 mA, o de los valores establecidos como errores de rango. Además un rango de 16mA es fácilmente digitalizable y manejable en binario, octal o hexadecimal.

Una señal de corriente ofrece una mayor resistencia contra efectos electromagnéticos que señales de tensión. Las perturbaciones electromagnéticas se manifiestan en variaciones de tensión y provocan pocas variaciones de corriente. Por este motivo, los dispositivos eléctricos que proporcionan una señal de 4-20 mA (miliamperios) son herramientas importantes para proporcionar una comunicación fiable entre sensores y dispositivos de control en largas distancias.

Otra ventaja importante es que a veces es posible realizar el enlace entre emisor y receptor con sólo dos hilos compartidos por la alimentación y la señal. En la figura *a* se presenta el circuito general con cuatro hilos, dos para la alimentación y dos para la señal. Normalmente es posible compartir un hilo de retorno, tal como se indica en la figura *b*. En el caso de la figura *c*, se conecta la fuente de alimentación en serie con el dispositivo o dispositivos de lectura y cualquier resistencia de bucle que haya hasta el sensor. La tensión de alimentación usual es de 24 V: 12 V de caída en el transmisor, 5 V de caída en el receptor (250 Ω), 2 V de caída en la línea (100 Ω) y 5 V de caída en una resistencia dispuesta para tener seguridad intrínseca en caso de corto circuito. Esta tensión de alimentación de aumentar al hacerlo la resistencia de carga, a razón de 1 V por cada 50 Ω.

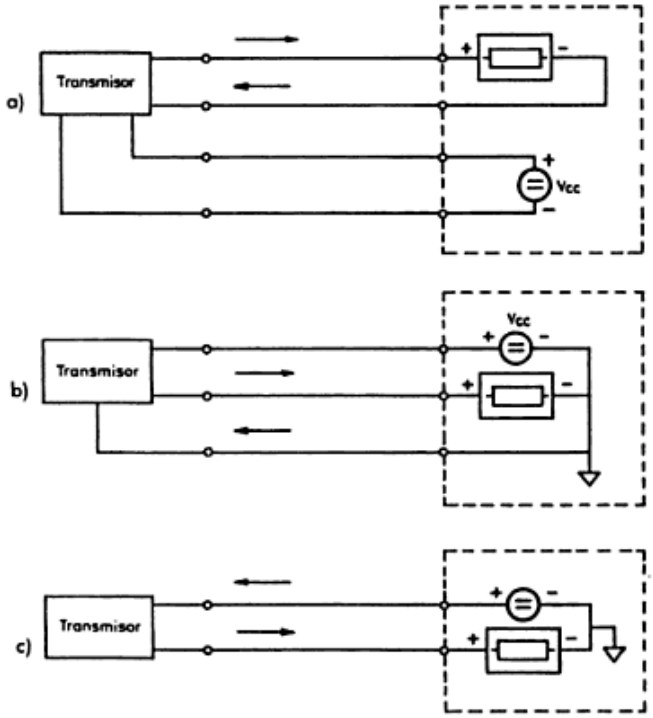


Figura 10.16 Telemedida por corriente empleando: *a*) cuatro hilos; *b*) tres hilos; *c*) dos hilos.

El conexionado habitual por ser el más económico es el de dos hilos en lugar del conexionado de tres hilos. Las variantes de 4…20 mA con tres hilos son poco habituales y se aplican únicamente para instrumentos que requieren un elevado nivel de energía auxiliar.

Otras señales muy frecuentes en la automatización industrial son 0...10 V, 1…5V y 1…10 V. Las ventajas residen en el fácil manejo y en la detección fácil de errores mediante un multímetro.

Sin embargo este tipo de señalización de tensión es muy susceptible a perturbaciones electromagnéticas, provocando errores de medición. Por lo tanto se deben utilizar cables apantallados para la protección. Las señales de 0-10 V, 1-5 V y 1-10 V son muy frecuentes como valores nominales de motores, pero también se aplican en contados casos en sensores de temperatura y presión.

4. MEMORIA DE CÁLCULO

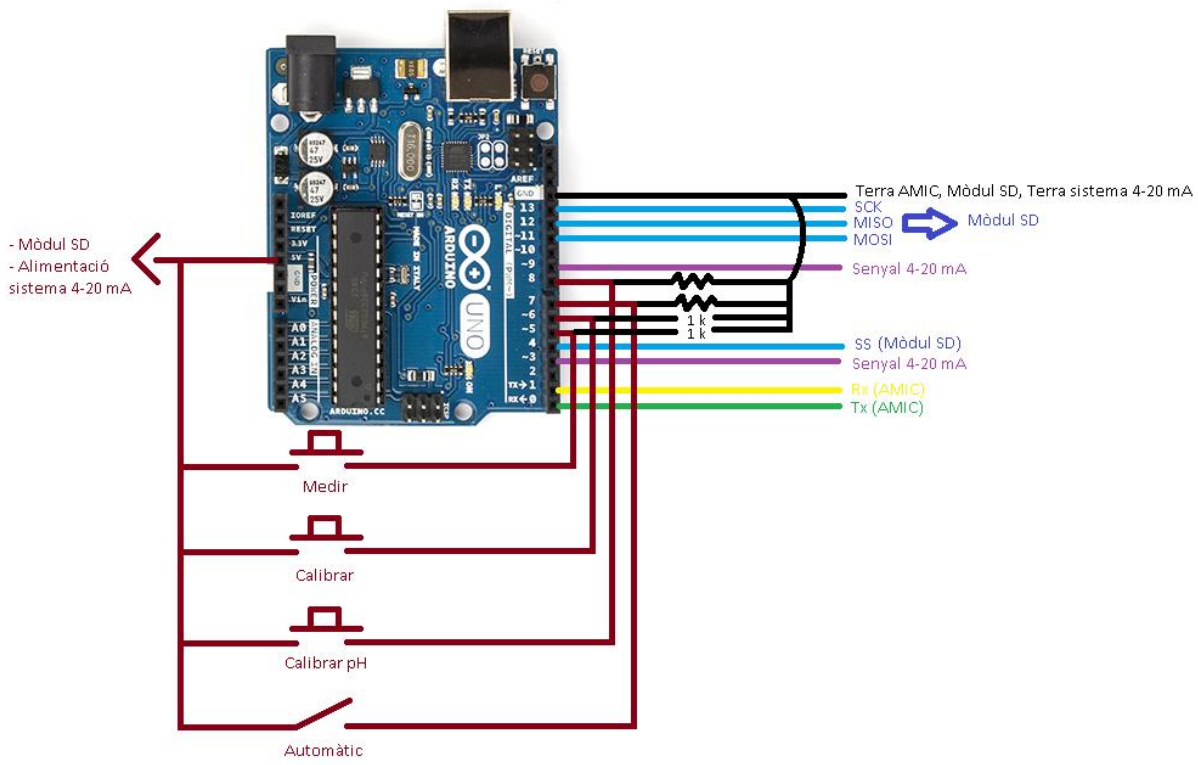
4.1. Hardware

En este apartado explicaremos principalmente el hardware correspondiente a la parte del arduino y a la generación de la señal 4-20 mA, ya que es la parte del hardware de nuestro ionómetro con la que principalmente hemos trabajado. El resto de componentes electrónicos que configuran nuestro ionómetro, que hemos descrito brevemente en la memoria descriptiva, siguen el mismo esquema electrónico que la versión anterior de nuestro producto.

4.1.1. Arduino

En cuanto a hardware es muy importante la presencia del arduino, ya que nos proporciona un gran abanico de posibilidades a la hora de mejorar e incrementar las funciones de nuestro producto.

En la figura que presentamos a continuación se pueden apreciar cada una de las conexiones que establecemos con nuestro arduino:



En primer lugar, utilizaremos tres pulsadores y un interruptor que conectaran los pines 5, 6, 7 y 8, configurados como entradas, a la tensión de referencia de 5 V que sale del propio arduino, como se muestra en la figura estos tres pulsadores y el interruptor nos servirán para indicarle al software del arduino que instrucciones queremos que este envíe al software del PIC18F4550 para que este realice. El motivo de que utilicemos un interruptor en lugar de un pulsador para la instrucción de 'automático' es que esta se realice repetidamente mientras mantengamos el interruptor conectado, sin embargo el resto de instrucciones solo deben realizarse una vez cada vez que apretamos los

pulsadores. Más adelante, en el apartado *"4.2. Software"* se explicará detenidamente el significado de cada una de las instrucciones y como son interpretadas tanto por el arduino como por el PIC18F4550-I/P.

El segundo paso y uno de los más importantes es establecer las conexiones de la comunicación USART y comprobar que esta se realiza correctamente, ya que para poder utilizar cualquiera de las funciones que arduino nos permite implementar en nuestro producto, debemos conseguir que los datos recogidos por el sensor ISE y el ionómetro lleguen con la máxima precisión y rapidez posibles a nuestro arduino y que este pueda guardarlos y utilizarlos del modo que nosotros le indiquemos a través del software. En el propio arduino podemos ver que los pines 0 y 1 corresponden a Rx y Tx respectivamente y por lo tanto como vemos en la figura anterior, el pin 0 (Rx) irá conectado a Tx del microcontrolador PIC18F4550-I/P, y el pin 1 (Tx) se conectará con Rx del microcontrolador. Una vez establecida la conexión, solo necesitamos comprobar que tanto el software como el hardware funcionan correctamente. Para ello, simplemente conectamos arduino con el PC y establecemos en el software que muestre por la pantalla del ordenador los datos que arduino recibe del microcontrolador principal.

Una vez nuestro arduino es capaz tanto de enviar nuestras instrucciones al potenciómetro como de recibir los datos de las calibraciones y las medidas que realiza nuestro ionómetro, y de guardarlos en variables, ya podemos guardar estos datos en la memoria SD o en enviar una señal PWM al generador de señal 4-20 mA que nosotros mismos hemos diseñado.

Para guardar los datos en una memoria SD necesitamos un módulo SD que nos permite conectar nuestra tarjeta SD con nuestro arduino, este módulo es el siguiente:

En esta figura se pueden apreciar cada uno de los pines del módulo para SD de arduino, y en la figura anterior se mostraban que pines de arduino van conectados a cada uno de los pines que necesitamos utilizar del módulo SD. Por si no queda suficientemente claro con ambas fotografías, las conexiones son las siguientes:

Arduino **módulo SD**

Pin 11 (MISO) ⟶ MOSI

Pin 12 (MOSI) ⟶ MISO

Pin 13 (SCK) ⟹ SCK

Pin 4 (CS) ⟹ CS

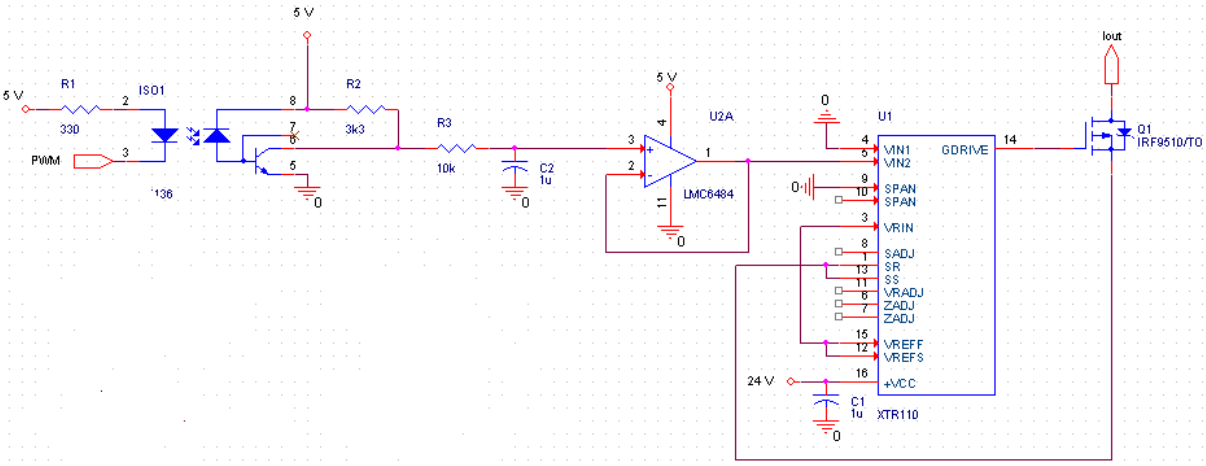
GND ⟹ GND

5 V ⟹ 5 V

Por último en el arduino que estamos utilizando se pueden usar los pines 3, 5, 6, 9, 10 y 11 como salidas PWM. Como ya tenemos ocupados muchos de estos pines, en nuestro caso utilizaremos los pines 3 y 9 como salidas PWM, por las cuales saldrá una señal PWM correspondiente con el valor de la medida que hemos obtenido de nuestro potenciómetro. Esta señal PWM será la que transformaremos en una señal 4-20 mA del modo que se explica a continuación.

4.1.2. Generación de la señal 4-20 mA

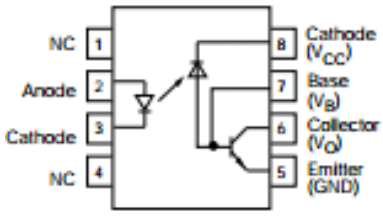
Como se explica en el apartado anterior, finalmente conseguimos obtener una señal PWM que se corresponde con el valor de la medida obtenida por el ionómetro, esta señal PWM sale del arduino a través de los pines 3 y 9, los cuales están conectados al circuito que podemos ver en la siguiente figura:



Este circuito se encarga de transformar nuestra señal PWM en una señal 4-20 mA. Está compuesto por un optoacoplador, un filtro RC, un seguidor de tensión, el chip integrado XTR110 y por último un MOSFET P-Channel. A continuación explicaremos cada uno de estos componentes.

4.1.2.1. Optoacoplador

Usamos un optoacoplador porque de esta forma podemos separar el circuito que usamos para obtener la señal 4-20 mA con el resto de circuitos de nuestro producto y evitamos que las interferencias que uno de ellos pueda generar o que en uno de ellos puedan interferir, interfiera en el resto de circuitos. En nuestro caso hemos decidido utilizar el optoacoplador 6N136:



El 6N136 es un optoacoplador con un diodo GaAIAs emisor de infrarrojos, junto ópticamente con un fotodetector integrado que consiste en un fotodiodo y un transistor de alta velocidad en un paquete de plástico DIP 8.

Las señales se pueden transmitir entre dos circuitos eléctricamente separados hasta frecuencias de 2 MHz. La diferencia de potencial entre los circuitos a ser acoplados no se le permite superar los voltajes máximos permisibles de referencia.

Características:

• Aislamiento de tensiones de prueba: V_{RMS} = 5300 V.

• Compatible con TTL

• Alta velocidad de bits: 1 Mbit / s

• Modo común con inmunidad a altas interferencias.

• Ancho de banda de 2 MHz.

• Salida del colector abierta.

• Posibilidad de cablear externamente la base.

4.1.2.2. Filtro RC

La modulación por ancho de pulso (PWM) permite que el Arduino (un dispositivo puramente digital) pueda generar una tensión analógica. La idea básica es la siguiente: Si el valor de la concentración del ion es alto, la señal PWM generada estará más tiempo en T_{ON} (voltaje = 5 V) que en T_{OFF} (voltaje = 0 V), por lo cual el voltaje medio en ese pin estará cerca de 5 V. Un valor de la concentración del ion bajo estará cerca de 0 V. Si la señal PWM varia rápidamente entre alta y baja, por ejemplo la mitad del tiempo tenemos una señal de salida alta y la otra mitad, una señal baja. Entonces el voltaje promedio estaría a medio camino entre 0 y 5 V (2,5 V). De esta forma, consiguiendo establecer el voltaje medio del PWM como un valor de tensión continua, podemos convertir el valor digital de las concentraciones de los iones en un valor analógico de entre 0 y 5 V. Los tiempos T_{ON} y T_{OFF} del PWM dependen directamente del valor de ciclo de trabajo que se especifique.

El ciclo de trabajo controla la tensión analógica de una manera muy directa; cuanto mayor sea el ciclo de trabajo más alto es el voltaje. En el caso del Arduino, el ciclo de trabajo puede variar desde 0 a 255. Por este motivo en primer lugar, debemos hacer una conversión del rango de valores que puede obtener una concentración de un ion al rango de valores del ciclo de trabajo. Una vez sabemos el ciclo de trabajo que debe tener el

PWM, podemos establecer la relación entre el ciclo de trabajo obtenido y el voltaje medio que debería resultar de la salida PWM con la siguiente fórmula:

(*Ciclo de trabajo* / 255) x 5V = Voltaje medio.

Por ejemplo, si el ciclo de trabajo es 100, (100 / 255) x 5 V = 1.96 V; PWM emite un tren de pulsos para crear (a través de una red RC) un voltaje promedio de 1.96V.

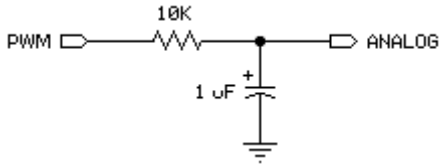
Con el fin de convertir PWM en una tensión continua tenemos que filtrar los pulsos y almacenar la tensión media. La combinación de resistencia y condensador puede hacer ese trabajo tal y como se muestra a continuación. El condensador mantendrá la tensión establecida por PWM. Por cuánto tiempo se mantendrá la tensión depende de la cantidad de corriente que se extraiga por la circuitería externa y de la fuga interna del condensador. Con el fin de mantener el voltaje relativamente constante, un programa debe repetir periódicamente el PWM para dar el condensador una carga fresca.

Del mismo modo que se necesita tiempo para descargar un condensador, también se necesita tiempo para cargarlo en primer lugar. Las instrucciones del PWM permiten especificar el tiempo de carga en términos de duración del PWM. El timing de las unidades en *Duración* es un milisegundo.

Para determinar el tiempo de carga del condensador, Utilizamos la siguiente fórmula:

Tiempo de carga = 5 x R x C

Por ejemplo, el circuito que utilizamos nosotros en la parte práctica del proyecto, utiliza una resistencia de 10 kΩ y un condensador de 1 µF:



Tiempo de carga = 5 x 10 k x 1 µ= 50 x 10^{-3} segundos, o 50 milisegundos.

Ahora solo necesitamos establecer el tiempo de carga de 50 milisegundos en las instrucciones para PWM del Arduino y conseguiremos el voltaje medio, de nuestra señal de salida PWM, en la salida del filtro RC que se muestra en la figura anterior.

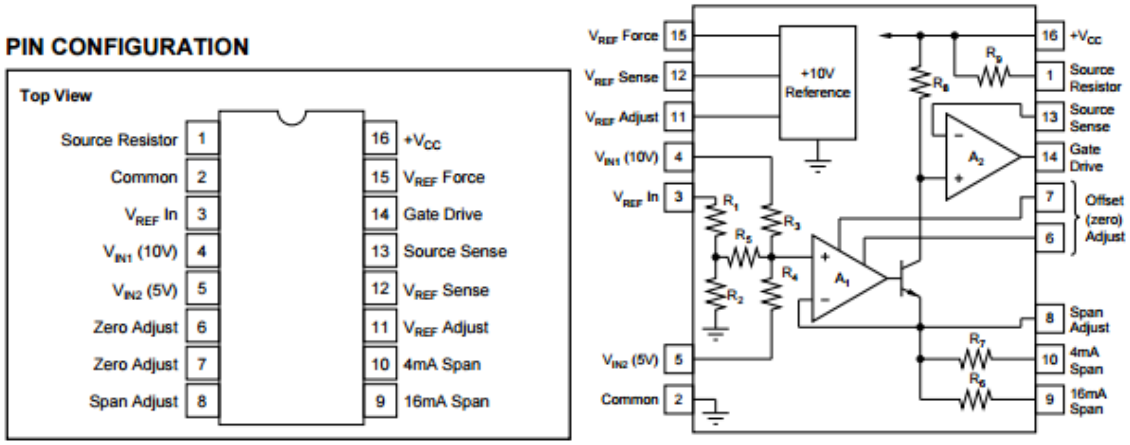
EL PWM carga el condensador y la carga presentada por el circuito externo lo descarga. Cuanto tiempo dura la carga, y por lo tanto con qué frecuencia nuestro programa debe repetir el PWM para actualizar la carga, depende de la cantidad de corriente que consuma el circuito, y de como de estable deba ser la tensión. Es posible reducir la descarga del condensador colocando un simple amplificador operacional como seguidor de tensión a la salida del filtro RC si los requerimientos de carga o estabilidad son más altos de los que el circuito pueda manejar, como es nuestro caso.

4.1.2.3. XTR110

El XTR110 es un preciso convertidor de voltaje a corriente diseñado para la transmisión de señal analógica. Acepta entradas de 0 a 5V o 0 a 10V y se puede conectar para obtener salidas de 4 a 20 mA, 0 a 20 mA, 5 a 25 mA y muchos otros rangos comúnmente usados.

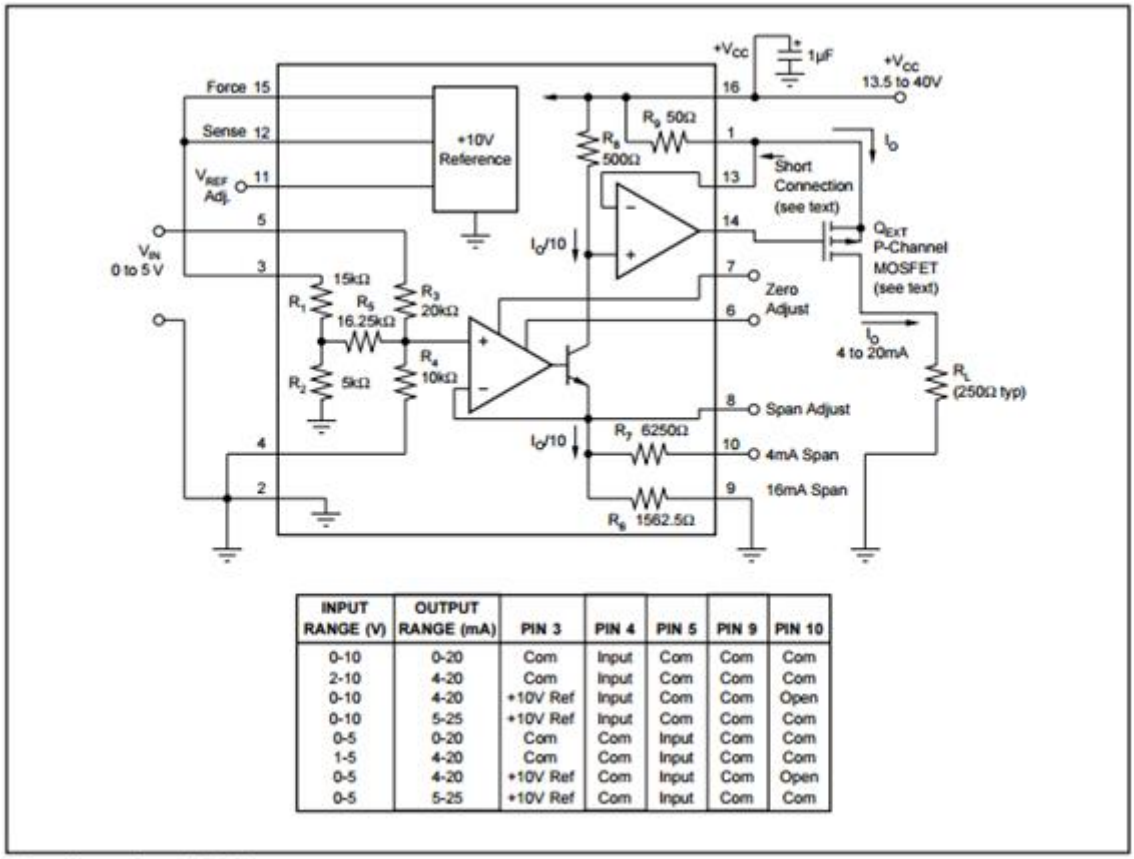
Una precisa red de resistencias en el chip proporciona una entrada a escala y una corriente de compensación. También dispone de una referencia interna de voltaje de 10V que se puede utilizar para alimentar circuitería externa.

El XTR110 está disponible en un DIP de plástico de 16 pines, DIP de cerámica y SOL-16 paquetes de montaje en superficie. También están disponibles modelos con un rango de temperaturas comerciales e industriales.



Aplicaciones:

- Control de procesos industriales.
- Transmisores de presión o temperatura.
- Excitación de puente en modo corriente
- Circuitos transductores a tierra.
- Fuente de corriente de referencia para adquisición de datos.
- Fuente de corriente programable para equipos de prueba.
- Fuente de alimentación, sistema de alimentación monitorizado.



INPUT RANGE (V)	OUTPUT RANGE (mA)	PIN 3	PIN 4	PIN 5	PIN 9	PIN 10
0-10	0-20	Com	Input	Com	Com	Com
2-10	4-20	Com	Input	Com	Com	Com
0-10	4-20	+10V Ref	Input	Com	Com	Open
0-10	5-25	+10V Ref	Input	Com	Com	Com
0-5	0-20	Com	Com	Input	Com	Com
1-5	4-20	Com	Com	Input	Com	Com
0-5	4-20	+10V Ref	Com	Input	Com	Open
0-5	5-25	+10V Ref	Com	Input	Com	Com

Conexiones circuito básico

La figura muestra las conexiones básicas requeridas para una entrada de 0 a 5 V y una salida de 4 a 20 mA. Otros rangos de tensión y corriente de entrada y salida requieren cambios en las conexiones de los pines 3, 4, 5, 9 y 10 como se muestra en la tabla de la Figura.

La función de transferencia del XTR110 es:

$$I_O = \frac{10\left[\frac{(V_{REF\ IN})}{16} + \frac{(V_{IN1})}{4} + \frac{(V_{IN2})}{2}\right]}{R_{SPAN}}$$

R_{SPAN} es la resistencia interna de 50Ω, R9, cuando se conecta como se muestra en la Figura. Puede conectarse una R_{SPAN} para diferentes rangos de corriente de salida como se describe más adelante.

4.1.2.3.1. Rango del voltaje de entrada

El amplificador operacional interno, A1, puede resultar dañado si su entrada no inversora (nodo interno del XTR110) desciende más de 0.5 V por debajo del común (0V). Esto podría ocurrir si los pines de entrada 3, 4 o 5 fueran conectados a un amplificador operacional cuya salida pueda llegar a ser negativa en condiciones anormales. El voltaje en la entrada de A1 se calcula:

$$V_{A1} = \frac{(V_{REF\ IN})}{16} + \frac{(V_{IN1})}{4} + \frac{(V_{IN2})}{2}$$

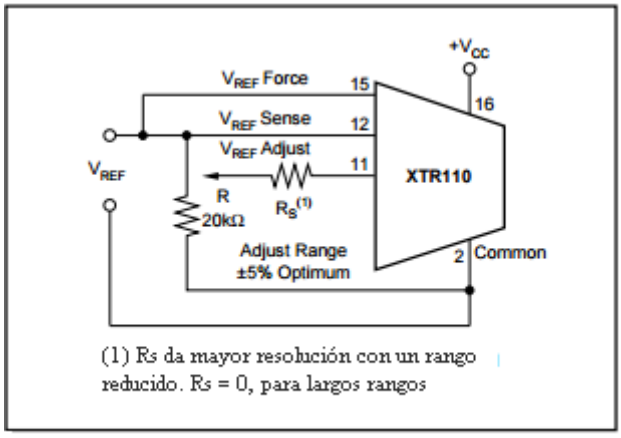
No se debe permitir que esta tensión descienda por debajo de -0.5V. Si es necesario, un diodo de sujeción se puede conectar desde la entrada negativa a tierra para sujetar la tensión de entrada.

4.1.2.3.2. Común (Tierra)

Debemos prestar especial atención a la hora de conectar los comunes (tierras). Todas las tierras deben estar unidas en un punto lo más cercano posible al pin 2 del XTR110. La única excepción es el retorno de I_{OUT}, este puede ser conectado en cualquier punto de tierra donde no interfiera sobre el pin 2 del XTR110.

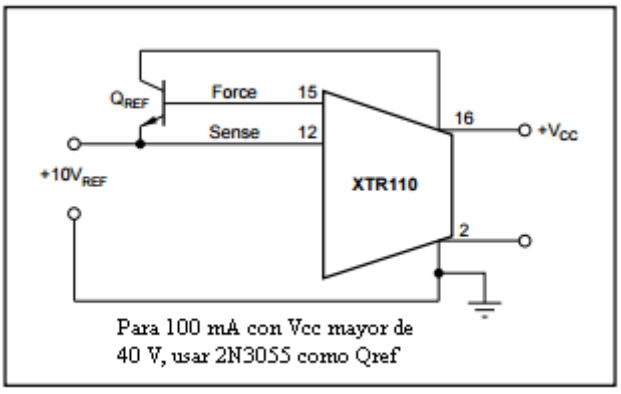
4.1.2.3.3. Voltaje de referencia

La tensión de referencia se regula con precisión en el pin 12 ($V_{REF\ SENSE}$). Para preservar la exactitud, cualquier carga, incluyendo el pin 3, debe conectarse a este punto. El circuito de la figura muestra el ajuste de la tensión de referencia:



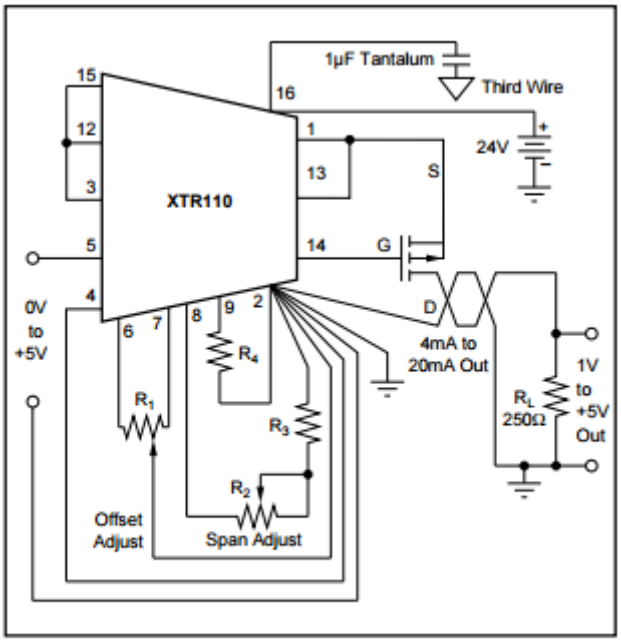
Ajuste opcional del voltaje de referencia

La capacidad de corriente de la referencia interna del XTR110 es de 10 mA. Esto se puede ampliar si se desea mediante la adición de un transistor NPN externo, como se muestra en la Figura:



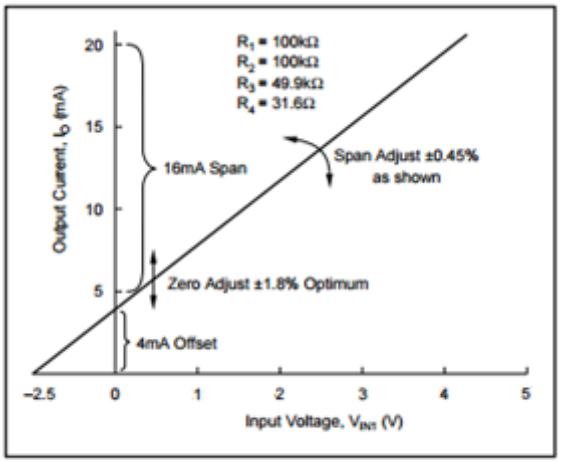
Incremento de la corriente de referencia

4.1.2.3.4. Ajuste del offset (cero)

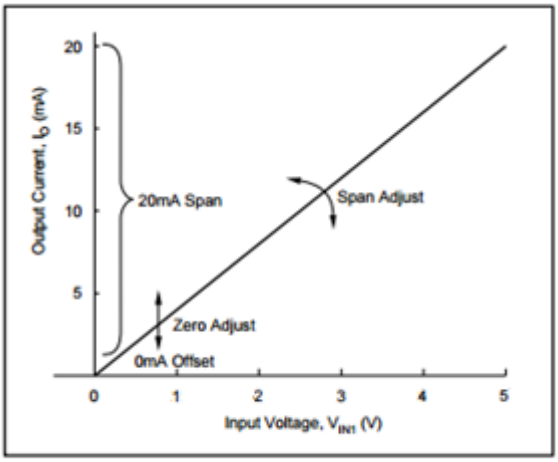


Circuito de ajuste del offset y el Span, para una entrada de 0 a 5 V y salida de 4 a 20 mA.

La corriente de offset se puede ajustar mediante el potenciómetro, R1, que se muestra en la Figura. En primer lugar se ajusta el voltaje de entrada a cero y luego ajustamos R1 para obtener 4 mA en la salida. Cuando el rango de salida empieza a partir de 0 mA, se recomienda el siguiente procedimiento especial: primero se ajusta la entrada a un pequeño distinto de cero y luego ajustamos R1 para obtener la corriente de salida correspondiente al valor de entrada previamente ajustado, de este modo cuando la entrada sea cero, la salida será cero. Las siguientes figuras, muestran gráficamente cómo ajustar el offset.



Configuración del Zero y el Span, con entrada de 0 a 5 V y salida de 4 a 20 mA.



Configuración del Zero y el Span, con entrada de 0 a 5 V y salida de 0 a 20 mA.

4.1.2.3.5. Ajuste del Span

El sobrante de la corriente de salida a gran escala se ajusta con el potenciómetro, R2, que se muestra en la primera figura del apartado anterior. Este ajuste es interactivo con

el ajuste de offset, y por tanto es necesario dar varias indicaciones. En primer lugar, con el circuito mostrado, ajustamos el voltaje de entrada a + 5 V y a continuación ajustamos R2 para obtener exactamente 20 mA en la salida. Los valores de las resistencias R2, R3 y R4 que participan también en el ajuste del span se determinan de la siguiente forma: elegimos R4 de forma que disminuya ligeramente el span; a continuación, ajustamos R2 y R3 para aumentar el span de forma que podamos obtener una salida lo más exacta posible.

4.1.2.3.6. Funcionamiento con bajo coeficiente de temperatura

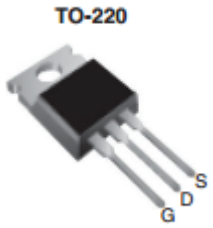
Aunque la precisión de las resistencias del XTR110 este dentro de 1 ppm / ° C, la corriente de salida depende del coeficiente de temperatura (TC) de cualquiera de las resistencias, R6, R7, R8, y R9. Dado que el coeficiente de temperatura máximo de la corriente de salida es de 20 ppm / ° C, este puede variar 20 ppm / ° C.

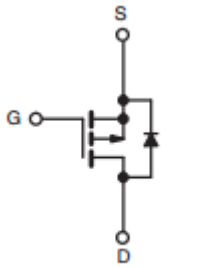
Para el funcionamiento con un bajo coeficiente de temperatura, cualquiera de las resistencias de span (R6 o R7) o la resistencia de fuente (R9), pueden ser sustituidas por resistencias con bajo coeficiente de temperatura, pero no ambas.

4.1.2.4. Mosfet IRF9510

Este MOSFET de potencia, nos proporciona una excelente combinación entre conmutación rápida, un diseño robusto, baja resistencia y una gran calidad en cuanto a coste-efectividad.

El paquete TO-220 es el más aceptado universalmente para todas las aplicaciones comerciales e industriales con niveles de disipación de potencia de aproximadamente 50 W. La baja resistencia térmica y el bajo costo del paquete TO-220 contribuyen a su amplia aceptación en la industria.





P-Channel MOSFET

Características:

• V_{DSS} = -100 V

• I_D = -4 A

• $R_{DS(ON)}$ = 1,2 Ω

• Potencia máxima disipada (P_{MAX}) = 43 W

• P-Channel

• Temperatura máxima de funcionamiento de 175 ° C

• Rápida conmutación (nanosegundos)

• Alta impedancia de entrada

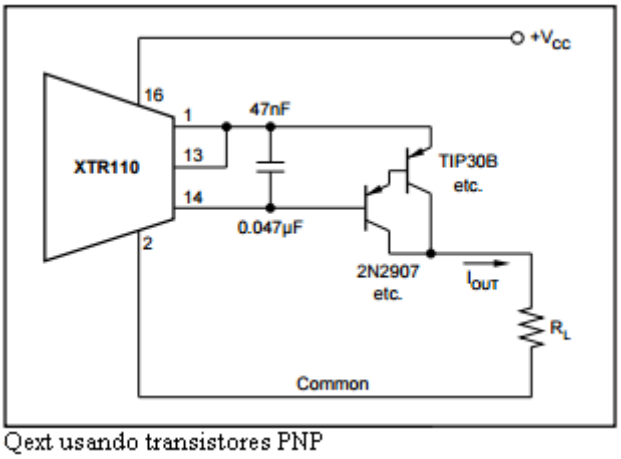
4.1.2.4.1. Transistor externo (Q_{EXT})

Como se muestra en las conexiones del chip XTR110, se requiere un transistor externo para que este conduzca la corriente de la señal de salida.

Se recomienda un transistor MOSFET P-Channel. Debe tener una tensión nominal igual o mayor que la tensión máxima de la fuente de alimentación.

Si la tensión de alimentación, + V_{CC}, sobrepasa la caída de tensión de gate a source de Q_{EXT}, y la conexión de salida (drain de Q_{EXT}) está rota, Q_{EXT} podría fallar. Si la caída de tensión de gate a source es inferior a + V_{CC}, Q_{EXT} puede ser protegido con un diodo zener de 12V conectado de gate a source.

Dos transistores PNP (conectado-Darlington) se pueden utilizar como Q_{EXT} (véase la Figura inferior), pero hay que tener en cuenta que se requiere un condensador adicional para la estabilidad. No se recomiendan los transistores integrados Darlington porque sus resistencias internas de base-emisor causan un error excesivo.



Qext usando transistores PNP

4.1.2.4.2. Disipación del transistor

La disipación de potencia máxima de Q_{EXT} depende de la tensión de alimentación y la corriente de salida a escala completa. Suponiendo que la resistencia de carga es baja, la potencia disipada por Q_{EXT} es:

$P_{MAX} = (+ V_{CC}) I_{FS}$

El tipo de transistor y el disipador de calor deben ser elegidos de acuerdo a la disipación de potencia máxima para evitar el sobrecalentamiento.

4.2. Software

En cuanto a software, debemos decir que el software principal lo encontramos en el microcontrolador PIC18F4550 ya que con él, controlamos la entrada escogida del multiplexor, recibimos e interpretamos el valor del convertidor A/D, calculamos pendientes e intersecciones para el calibrado del sistema y el valor de las concentraciones de iones, establecemos la secuencia de las bombas para cada una de las instrucciones posibles que puede recibir nuestro microcontrolador y enviamos los datos finales obtenidos al arduino.

Pero también debemos tener en cuenta el software del arduino, con el cual y a través de los pulsadores y el interruptor interactuamos con el usuario para que este pueda indicar cuál es la acción que quiere que el AMI C7 realice, recibimos los valores finales del microcontrolador PIC18F4550, escribimos en la tarjeta SD y generamos la señal PWM.

4.2.1. Microcontrolador PIC18F4550-I/P

4.2.1.1. Programa principal (MAIN)

Código:

```
#include <p18f4550.h>
#include <usart.h>
#include <stdio.h>
#include <delays.h>
#include <stdlib.h>


#include "dades.h"
#include "mV.h"
#include "parametres.h"
#include "ppm.h"


#pragma config FOSC = HS
#pragma config MCLRE = ON
#pragma config PBADEN = OFF
#pragma config LVP = OFF
#pragma config WDT = OFF

void ISRRecepcion(void);
void getsRS232(char *buffer, unsigned char len);
void sendStatus(void);
void iniRS232(void);

// variable global externa

struct dataType d;
struct mVType mV;
struct paramType parametre;
struct ppmType ppm;
```

```
// Sección de código a partir de la dirección 0x0008

#pragma code interrupcion = 0x0008
void VectorInterrupcion(void)
{

	_asm goto ISRRecepcion _endasm
}
#pragma code

// Rutina de Interrupcion

#pragma interrupt ISRRecepcion
void ISRRecepcion(void)
{
	#define C_REBUT 4
	unsigned char operacio;
	char rebut[C_REBUT];
	rebut[0]=0x00;
	rebut[1]=0x00;
	rebut[2]=0x00;
	rebut[3]=0x00;
	if(PIR1bits.RCIF==1)
	{ 				//interpretamos la trama recibida de arduino
		getsRS232(rebut,C_REBUT);
		operacio = rebut[0];
		switch(operacio)
		{
			case ST_CALIBRAR:			// Calibrado
				if(d.op_status==ST_IDLE || d.op_status==ST_PLE)
				{
					d.op_status = ST_CALIBRAR;
				}
				break;

			case ST_MEDIR:				// Medida
				if(d.op_status==ST_IDLE || d.op_status==ST_PLE)
				{
					d.op_bomba = B_M1;
					d.op_status = ST_MEDIR;
				}
				break;

			case ST_CALIBRARPH:			// Calibración de pH
				if(d.op_status==ST_IDLE || d.op_status==ST_PLE)
				{
					d.pHCal = 1;
					d.op_status = ST_CALIBRAR;
				}
				break;
```

```
                default:
                        putcUSART('Z');
                        break;
                }

                CloseUSART();
                Delay10KTCYx(255);
                iniRS232();

                PIR1bits.RCIF=0;
        }
}

// declaración de las funciones de las librerías

void Delay10TCYx(unsigned char);
void Delay100TCYx(unsigned char);
void Delay1KTCYx(unsigned char);
void Delay10KTCYx(unsigned char);

void iniDades(void);
void inimV(void);
void iniParametres(void);
void iniPPM(void);
void iniPIC(void);
void iniNTx(void);

void calibra(void);
void medeix(void);
void llegeixM(void);
void checkPoint1(void);
void checkPoint2(void);
void condicionar(void);
void func_Bomba(unsigned char);
void func_omplir(void);

void sendStatus(void);

//      VOID MAIN ( VOID )

void main()
{
        iniDades();
        inimV();
        iniParametres();
        iniPPM();
        iniPIC();
        iniNTx();

        while(1)
        {
```

```
		switch(d.op_status)		// Según la instrucción recibida iniciamos una
	{					// secuencia
			case ST_IDLE:	// descanso
				break;

			case ST_CALIBRAR:		// Calibrar
				#ifdef AMIC
					d.op_bomba = B_P1;
					d.op_status = ST_CHP1;
					sendStatus();		// enviar estado 'leyendo P1'
					checkPoint1();		// iniciar lectura 1ª muestra
				#endif
				#ifdef AMIC
					d.op_bomba = B_P2;
					d.op_status = ST_CHP2;
					sendStatus();		// enviar estado 'leyendo P2'
					checkPoint2();		// iniciar lectura 2ª muestra
				#endif
				#ifdef AMIC
					d.op_status = ST_CALIBRAR;
					sendStatus();		// enviar estado 'calibrando'
					calibra();	// calcular pendientes e intersecciones
				#endif
				d.op_status = ST_IDLE;
				sendStatus();		// enviar estado 'en reposo'
				d.pHCal = 0;
				break;

			case ST_MEDIR:			// Medir
				#ifdef AMIC
					d.op_status = ST_MEDIR;
					//d.op_bomba = B_M1;
					sendStatus();	// enviar estado 'midiendo M1'
					medeix();	//iniciar medida de concentraciones
				#endif
				d.op_status = ST_IDLE;
				sendStatus();		// enviar estado 'en reposo'

				break;

			default:
				break;
	}
}
}

void iniPIC(void)			//Inicializamos PIC18F4550
{
	Delay1KTCYx(800);
	// Por defecto todos los pines a digital.
	ADCON1 = 0x0F;
```

```
        // Configuramos los puertos como entradas (1) o salidas(0)
        TRISA = 0b00000000;
        TRISB = 0b00000000;
        TRISC = 0b00000000;
        TRISD = 0b00000000;
        TRISE = 0b00000000;
        // Vaciamos todos los puertos
        PORTA = 0b00000000;
        PORTB = 0b00000000;
        PORTC = 0b00000000;
        PORTD = 0b00000000;
        PORTE = 0b00000000;
        Delay1KTCYx(800);
        iniRS232();
        Delay1KTCYx(800);
        RCONbits.IPEN=0;        // Desabilitamos Prioridades.
        INTCONbits.PEIE=1;      // Habilitamos Interrupción de periféricos.
        INTCONbits.GIE=1;       // Habilitamos Interrupción Global.
}

void iniMux(void);
void iniADC(void);
void iniTemp(void);
void iniBombes(void);
void iniCella(void);

void iniNTx(void)           //Inicializamos el ionómetro
{
        sendStatus();               //enviar estado 'inicializando'
        Delay1KTCYx(800);
        iniMux();                   //inicializar multiplexor
        Delay1KTCYx(800);
        iniADC();                   //inicializar convertidor A/D
        Delay1KTCYx(800);
        iniBombes();                //inicializar bombas
        Delay1KTCYx(800);
        iniCella();                 //inicializar celda
        Delay1KTCYx(800);
        d.op_status=ST_IDLE;
        sendStatus();               //enviar estado 'reposo'
}
```

4.2.1.2. USART

Código:

```
#include <p18f4550.h>
#include <usart.h>
#include <delays.h>
#include <stdio.h>
#include "parametres.h"
```

```
#include "dades.h"
#include "mV.h"
#include "ppm.h"

#define EnableTxDtris TRISDbits.TRISD3
#define EnableTxDport LATDbits.LATD3


extern struct paramType parametre;
extern struct dataType d;
extern struct mVType mV;
extern struct ppmType ppm;
void iniRS232()                         //inicializamos USART
{
        //unsigned int spbrg = 260;     // baud Rate : 19200
        unsigned int spbrg = 520;       // baud Rate : 9600
        TRISCbits.TRISC6 = 0;           // Ponemos TX (RC6) como salida
        TRISCbits.TRISC7 = 1;           // RX (RC7) como entrada
        EnableTxDtris = 0;              // habilitamos como salida
        EnableTxDport = 0;
        OpenUSART(USART_TX_INT_OFF & USART_RX_INT_ON &
USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX &
USART_BRGH_HIGH, spbrg);
        BAUDCONbits.BRG16 = 1;
        Delay10KTCYx(1);
}

void getsRS232(char *buffer, unsigned char len)         // Recibimos una trama
{
        #define C_TIMEOUT 0x00001000
        char i;                                 // Contador de longitud
        unsigned char data;
        unsigned long timeout;
        for(i=0;i<len;i++)                      // Solo recuperar 'len' caracteres
        {
                timeout = 0x00000000;
                while(!DataRdyUSART())  // Esperamos a que los datos sean recibidos
                {
                        timeout++;
                        if(timeout>C_TIMEOUT)
                        {
                                timeout++;
                                break;
                        }
                }
                data = getcUSART();     // Obtenemos un carácter de la USART
                                        // y lo guardamos en el string
                *buffer = data;
                buffer++;               // Incrementamos el puntero del string.
        }
}
```

```
void putcUSART(char);
void whileBusyUsart(void);

void sendRS232New(void)                    //Enviamos una trama a Arduino
{
        unsigned char i;
        char toSend[80];

        toSend[0] = '#';
        toSend[1] = d.op_status;
        switch(d.op_status)
        {
                case ST_CALIBRAR:               //En caso de calibrado
                        toSend[2]='p';
                        for(i=0;i<8;i++)          //Guardamos las pendientes de los 8 iones
                        {
                                toSend[3+(i*4)]=(unsigned char)(parametre.m[i]>>24);
                                toSend[4+(i*4)]=(unsigned char)(parametre.m[i]>>16);
                                toSend[5+(i*4)]=(unsigned char)(parametre.m[i]>>8);
                                toSend[6+(i*4)]=(unsigned char)(parametre.m[i]);
                        }
                        toSend[35]='P';
                        for(i=0;i<8;i++)      //Guardamos las intersecciones de los 8 iones
                        {
                                toSend[36+(i*4)]=(unsigned char)(parametre.n[i]>>24);
                                toSend[37+(i*4)]=(unsigned char)(parametre.n[i]>>16);
                                toSend[38+(i*4)]=(unsigned char)(parametre.n[i]>>8);
                                toSend[39+(i*4)]=(unsigned char)(parametre.n[i]);
                        }
                        toSend[68]='@';         //Guardamos la temperatura
                        toSend[69]=(unsigned char)(parametre.tCal[0]>>24);
                        toSend[70]=(unsigned char)(parametre.tCal[0]>>16);
                        toSend[71]=(unsigned char)(parametre.tCal[0]>>8);
                        toSend[72]=(unsigned char)(parametre.tCal[0]);
                        toSend[73] = 'z';
                        for(i=0;i<74;i++)       //Enviamos la trama
                        {
                                putcUSART(toSend[i]);
                                whileBusyUsart();
                        }
                        break;

                case ST_MEDIR:                          //En caso de Medir
                        toSend[2] = d.op_bomba;
                        for(i=0;i<8;i++)                //Guardamos la concentración de iones
                        {
                                toSend[3+(i*4)]=(unsigned char)(ppm.M1[i]>>24);
                                toSend[4+(i*4)]=(unsigned char)(ppm.M1[i]>>16);
                                toSend[5+(i*4)]=(unsigned char)(ppm.M1[i]>>8);
                                toSend[6+(i*4)]=(unsigned char)(ppm.M1[i]);
                        }
```

```
                        toSend[35]='@';             //Guardamos la temperatura
                        toSend[36]=(unsigned char)(mV.tM1[0]>>24);
                        toSend[37]=(unsigned char)(mV.tM1[0]>>16);
                        toSend[38]=(unsigned char)(mV.tM1[0]>>8);
                        toSend[39]=(unsigned char)(mV.tM1[0]);
                        toSend[40] = 'z';
                        for(i=0;i<41;i++)           //Enviamos la trama
                        {
                                putcUSART(toSend[i]);
                                whileBusyUsart();
                        }
                        break;
                default:
                        toSend[2] = 'z';
                        for(i=0;i<3;i++)
                        {
                                putcUSART(toSend[i]);
                                whileBusyUsart();
                        }
                        break;
        }
}

void sendStatus(void)                   //Enviar Estado
{
        unsigned char i;
        char toSend[8];
        toSend[0] = '#';
        toSend[1] = ST_STATUS;
        toSend[2] = d.op_status;        //estado
        toSend[3] = d.op_temps;
        toSend[4] = d.op_numIons;
        toSend[5] = d.op_bomba;
        toSend[6] = d.op_metres;
        toSend[7] = 'z';
        for(i=0;i<8;i++)
        {
                putcUSART(toSend[i]);
                whileBusyUsart();
        }
}

void whileBusyUsart(void)
{
        while(BusyUSART());
}
```

4.2.1.3. CONVERTIDOR A/D AD7712

Código .h:

```
#include<p18f4550.h>
#include "dades.h"

        #define SCLK        LATDbits.LATD5
        #define SCLKtris    TRISDbits.RD5
        #define RFS         LATDbits.LATD7
        #define RFStris     TRISDbits.RD7
        #define SDATA       PORTDbits.RD0
        #define SDATAtris   TRISDbits.RD0
        #define TFS         LATDbits.LATD6
        #define TFStris     TRISDbits.RD6
        #define DRDY        PORTDbits.RD1
        #define DRDYtris    TRISDbits.RD1
        #define STDBY       LATDbits.LATD2
        #define STDBYtris   TRISDbits.RD2
        #define A0          LATDbits.LATD4
        #define A0tris      TRISDbits.RD4
```

Código .c:

```
#include "AD7712.h"
#include <delays.h>

//*****************************************************************
// Rutinas para el ADC7712
//*****************************************************************
void whileNOdataReady(void);
void delay_conf(void)
        {
                _asm nop _endasm
        SCLK = 1;
                _asm nop _endasm
        SCLK = 0;
        }

void setControlRegSelfCal(void)
{
        if(STDBY == 0)
        {
                STDBY = 1;
        }

        SDATAtris = 0;                  //pin0 puerto D salida
        A0 = 0;                         //R/W control register
        TFS = 0;                        //escribir en el AD7712
        SDATA = 0;   //MD2
        delay_conf();
```

```
SDATA = 0;   //MD1
delay_conf();

SDATA = 1;   //MD0
delay_conf();

SDATA = 0;   //G2
delay_conf();

SDATA = 0;   //G1
delay_conf();

SDATA = 0;   //G0
delay_conf();

SDATA = 0;   //CH
delay_conf();

SDATA = 0;   //PD
delay_conf();

SDATA=1;     //WL
delay_conf();

SDATA = 0;   //X
delay_conf();

SDATA = 0;   //BO
delay_conf();

SDATA = 0;   //B/U
delay_conf();

SDATA = 0;   //FS11
delay_conf();

SDATA = 0;   //FS10
delay_conf();

SDATA = 1;   //FS9
delay_conf();

SDATA = 1;   //FS8
delay_conf();

SDATA = 1;   //FS7
delay_conf();

SDATA = 1;   //FS6
delay_conf();
```

```
        SDATA = 1;   //FS5
        delay_conf();

        SDATA = 1;   //FS4
        delay_conf();

        SDATA = 1;   //FS3
        delay_conf();

        SDATA = 1;   //FS2
        delay_conf();

        SDATA = 1;   //FS1
        delay_conf();

        SDATA = 1;   //FS0
        delay_conf();

        TFS=1;          //fin escritura
        SDATAtris = 1;        //pin 0 puerto D entrada
        A0 = 1;               //R/W registros de datos y calibración.

        whileNOdataReady();//while(DRDY!=0);
}

void getValor(unsigned long* resultat)       //Obtenemos un valor del AD7712
{
        unsigned char i;
        unsigned long tmpin;

        tmpin=0;
        SDATAtris=1;                  //pin 0 puerto D entrada
        A0 = 1;                 //acceso al registro de datos y registro de calibración

        whileNOdataReady();
        RFS = 0;                              //indicamos al AD que leeremos datos
        for(i=0;i<24;i++)
        {
                SCLK=1;                       //pedimos un bit
                tmpin = tmpin|SDATA;          //cogemos el bit
                tmpin = tmpin<<1;
                SCLK=0;                       //indicamos que hemos cogido el bit
        }
        RFS=1;                                //fin de la lectura
        tmpin=tmpin>>1;
        *resultat = tmpin;
}
```

```
void getMitja(unsigned long* pMitja)            //Media de 8 valores recibido del ADC
{
        #define NmitjaADC 8
        unsigned char i;
        unsigned long tempADC;
        unsigned long pResultat[1];
        unsigned char counter;

        tempADC = 0L;
        for(i=0;i<NmitjaADC;i++)
        {
                counter=0x00;
                setControlRegSelfCal();
                getValor(pResultat);
                while((pResultat[0]==0x00FFFFFF)||(pResultat[0]==0x00000000))
                {
                        setControlRegSelfCal();
                        getValor(pResultat);
                        counter++;
                        if(counter>10)
                        {
                                break;
                        }
                }
                tempADC = tempADC + pResultat[0];
        }
        pMitja[0] = tempADC / NmitjaADC;
}

void getMitjaTemp(unsigned long* pMitja)        //Media de 5 valores de temperatura
{
        #define NmitjaT 5
        unsigned char i;
        unsigned long tempADC;
        unsigned long pResultat[1];
        unsigned char counter;


        tempADC = 0L;
        for(i=0;i<NmitjaT;i++)
        {
                counter=0x00;
                setControlRegSelfCal();
                getValor(pResultat);
                while((pResultat[0]==0x00FFFFFF)||(pResultat[0]==0x00000000))
                {
                        setControlRegSelfCal();
                        getValor(pResultat);
                        counter++;
                        if(counter>10)
                        {
```

```
                                break;
                        }
                }
                tempADC = tempADC + pResultat[0];
        }
        pMitja[0] = tempADC / NmitjaT;
}

void pauseADC(void);

void iniADC(){                  //inicializamos ADC
        //salidas
        SCLK=0;                 //ponemos un 0 al pin 5 del puerto D.
        SCLKtris=0;   //configuramos el pin 5 del puerto D como salida.
        RFS=1;                  //ponemos un 1 al pin 7 del puerto D.
        RFStris=0;      //configuramos el pin 7 del puerto D como salida.
        TFS=1;                  //ponemos un 1 al pin 6 del puerto D.
        TFStris=0;      //configuramos el pin 6 del puerto D como salida.
        STDBYtris=0;//configuramos el pin 1 del puerto D como salida.
        STDBY=1;    //AD no standby.
        A0 = 0;                 //ponemos un 0 al pin 4 del puerto D.
        A0tris=0;       //configuramos el pin 4 del puerto D como salida.
        //entradas
        DRDYtris=1;  //configuramos el pin 1 del puerto D como entrada.
        //entradas/salidas
        SDATA = 0;
        SDATAtris = 0;          //configuramos el bit 0 del puerto D como salida.
        Delay10TCYx(8);
        pauseADC();             //ponemos ad en standby
}

void pauseADC(void)
{
        STDBY = 0;
}

void resumeADC(void)
{
        setControlRegSelfCal();
}



void whileNOdataReady(void)
{
        unsigned long countNODataReady = 0L;
        unsigned char i = 0;
        while(DRDY != 0)
        {
                countNODataReady++;
```

```
                if(countNODataReady > 0x00080000)
                {
                        pauseADC();
                        Delay1KTCYx(800);
                        resumeADC();
                        i++;
                        if(i>2)
                        {
                                break;
                        }
                }
        }
        countNODataReady = 0L;
}
```

4.2.1.4. MULTIPLEXOR DG408DJZ

Código .h:

```
#include<p18f4550.h>
#include "dades.h"

        #define muxA0       PORTCbits.RC2
        #define muxA0tris   TRISCbits.RC2
        #define muxA1       PORTCbits.RC1
        #define muxA1tris   TRISCbits.RC1
        #define muxA2       PORTCbits.RC0
        #define muxA2tris   TRISCbits.RC0
```

Código .c:

```
#include "mux.h"

//*****************************************************************
// Rutinas para el MUX DG408DJZ
//*****************************************************************
void iniMux()
{                               //inicializamos el multiplexor
        muxA0=0;
        muxA0tris=0;
        muxA1=0;
        muxA1tris=0;
        muxA2=0;
        muxA2tris=0;
}
```

```
void Channel(unsigned char channel)
{
        switch(channel)                         //Elegimos el canal correspondiente
        {                                       //al ion que queremos medir
                case 0:
                        muxA0=0;
                        muxA1=0;
                        muxA2=0;
                        break;
                case 1:
                        muxA0=1;
                        muxA1=0;
                        muxA2=0;
                        break;
                case 2:
                        muxA0=0;
                        muxA1=1;
                        muxA2=0;
                        break;
                case 3:
                        muxA0=1;
                        muxA1=1;
                        muxA2=0;
                        break;
                case 4:
                        muxA0=0;
                        muxA1=0;
                        muxA2=1;
                        break;
                case 5:
                        muxA0=1;
                        muxA1=0;
                        muxA2=1;
                        break;
                case 6:
                        muxA0=0;
                        muxA1=1;
                        muxA2=1;
                        break;
                case 7:
                        muxA0=1;
                        muxA1=1;
                        muxA2=1;
                        break;
                default:
                        muxA0=0;
                        muxA1=0;
                        muxA2=0;
                        break;
        }
}
```

4.2.1.5. DATOS

Código .h:

```
#ifndef DATATYPE_H
	#define DATATYPE_H

	#define B3
	#define AMIC

	#define DEVICE_ID		(unsigned char)24
	#define FIRM_VERSION	0x32

	struct dataType{
	// ESTADOS Y OPERACIONES
		#define ST_IDLE			0x00
		#define ST_CALIBRAR		0x01
		#define ST_MEDIR		0x02
		#define ST_AUTOMATICO		0x03
		#define ST_CALIBRARPH		0x04
		#define ST_STATUS		0x05
		#define ST_INI			0x06
		//#define ST_CHP1		0x07
		//#define ST_CHP2		0x08
		//#define ST_COND		0x04
		//#define ST_RESET		0x06
		//#define ST_FIRM		0x07
		//#define ST_START		0x08
		//#define ST_CALIBRAR		0x09
		//#define ST_PARAMETROS		0x0A
		//#define ST_STOP		0x0B
		//#define ST_LIMPIEZA		0x0B
		//#define ST_PLENAR		0x0C
		//#define ST_CONFIGURAR		0x0D
		#define ST_PLE			0x0E

	// BOMBAS
		#define B_M1			0x00
		#define B_P1			0x10
		#define B_P2			0x20
		#define B_H2O			0x30
		#define B_BUIDAT		0x40
		#define B_M2			0x50
		#define B_M3			0x60
		#define B_M4			0x70

	// TIEMPOS
		#define T_CURT			0x01
		#define T_DEF			28
```

```
        //Variables que el usuario puede querer cambiar.
                unsigned char op_status;
                unsigned char op_temps;
                unsigned char op_numIons;
                unsigned char op_bomba;
                unsigned char op_metres;
                unsigned char start;
                unsigned char op_metresp1;
                unsigned char op_metresp2;

                unsigned char pHCal;
                unsigned char acond;
                unsigned char contcal;
                unsigned char valormin;
                unsigned char valorh;
        };

#endif
```

Código .c:

```
#include "dades.h"

extern struct dataType d;

void iniDades()         //Inicializamos variables que usaremos en el resto de funciones
{
        d.op_status = ST_INI;
        d.op_temps = 0x01;
        d.op_numIons = 7;
        d.op_bomba = B_BUIDAT;
        d.op_metres = 0x00;
        d.op_metresp1 = 0x00;
        d.op_metresp2 = 0x00;
        d.start = 0;
        d.contcal = 2;
        d.valormin = 10;
        d.valorh = 0;
        d.acond = 0;
        d.pHCal = 0;
}
```

4.2.1.6. PARÁMETROS

Código .h:

```
#ifndef PARAMTYPE_H
	#define PARAMTYPE_H

	struct paramType{
	// parámetros
		signed long m[8];
		signed long n[8];
		signed long tCal[1];
	};
#endif
```

Código .c:

```
#include "parametres.h"

extern struct paramType parametre;

void iniParametres()
{
	//variables para las pendientes
	parametre.m[0]=0.0f;
	parametre.m[1]=0.0f;
	parametre.m[2]=0.0f;
	parametre.m[3]=0.0f;
	parametre.m[4]=0.0f;
	parametre.m[5]=0.0f;
	parametre.m[6]=57.0f;
	parametre.m[7]=0.0f;

	//variables para intersecciones
	parametre.n[0]=0.0f;
	parametre.n[1]=0.0f;
	parametre.n[2]=0.0f;
	parametre.n[3]=0.0f;
	parametre.n[4]=0.0f;
	parametre.n[5]=0.0f;
	parametre.n[6]=404.0f;
	parametre.n[7]=0.0f;

	//variable para guardar la temperatura durante la calibración
	parametre.tCal[0]=0.0f;
}
```

4.2.1.7. BOMBAS

Código .h:

```
#include<p18F4550.h>
#include "dades.h"

#ifdef B3
	#define TRIS_B_P1		TRISBbits.RB0
	#define PORT_B_P1		PORTBbits.RB0
	#define TRIS_B_P2		TRISBbits.RB1
	#define PORT_B_P2		PORTBbits.RB1
	#define TRIS_B_B		TRISBbits.RB2
	#define PORT_B_B		PORTBbits.RB2
	#define TRIS_B_H2O		TRISBbits.RB3
	#define PORT_B_H2O		PORTBbits.RB3
	#define TRIS_B_M1		TRISBbits.RB4
	#define PORT_B_M1		PORTBbits.RB4
	#define TRIS_B_M2		TRISBbits.RB5
	#define PORT_B_M2		PORTBbits.RB5
	#define TRIS_B_M3		TRISAbits.RA0
	#define PORT_B_M3		PORTAbits.RA0
	#define TRIS_B_M4		TRISAbits.RA1
	#define PORT_B_M4		PORTAbits.RA1
#endif
```

Código .c:

```
#include "Bombes.h"

//definiciones
void EncendreBomba(unsigned char bomba);
void ApagarBomba(unsigned char bomba);
void Delay_05s(unsigned char t);

//Encender la bomba seleccionada durante un tiempo
void Bomba_05s(unsigned char nombomba,unsigned char t)
{
	unsigned char i;
	EncendreBomba(nombomba);
	Delay_05s(t);
	ApagarBomba(nombomba);
}

// Encender la bomba seleccionada
void EncendreBomba(unsigned char eBomba)
{
	switch(eBomba)
	{
		case 1:
			PORT_B_M1=1;
			break;
```

```
		case 2:
			PORT_B_B=1;
			break;
		case 3:
			PORT_B_P2=1;
			break;
		case 4:
			PORT_B_P1=1;
			break;
		default:
			PORT_B_P1=0;
			PORT_B_P1=0;
			PORT_B_B=0;
			PORT_B_M1=0;
			break;
	}
}

// Apagar la bomba seleccionada
void ApagarBomba(unsigned char aBomba)
{
	switch(aBomba)
	{
		case 1:
			PORT_B_M1=0;
			break;
		case 2:
			PORT_B_B=0;
			break;
		case 3:
			PORT_B_P2=0;
			break;
		case 4:
			PORT_B_P1=0;
			break;
		default:
			PORT_B_P1=0;
			PORT_B_P1=0;
			PORT_B_B=0;
			PORT_B_M1=0;
			break;
	}
}

void Delay10TCYx(unsigned char);
void Delay100TCYx(unsigned char);
void Delay1KTCYx(unsigned char);
void Delay10KTCYx(unsigned char);

// Delays (Retraso)
void Delay_05s(unsigned char t)
```

```
{
	unsigned char i;
	for(i=0;i<t;i++)
	{
		Delay10KTCYx(255);
	}
}

void Delay_20s(unsigned char t)
{
	unsigned char i,j;
	for(i=0;i<t;i++)
	{
		Delay_05s(0x20);
	}
}

void Delay_2m(unsigned char t)
{
	unsigned char i,j,k;
	for(i=0;i<t;i++)
	{
		Delay_20s(0x06);
	}
}

void func_encendreBomba(unsigned char eBomba);

// Inicializamos los puertos
void iniBombes()
{
	TRIS_B_P1 = 0x00;
	TRIS_B_P2 = 0x00;
	TRIS_B_B = 0x00;
	TRIS_B_H2O = 0x00;
	TRIS_B_M1 = 0x00;
	TRIS_B_M2 = 0x00;
	TRIS_B_M3 = 0x00;
	TRIS_B_M4 = 0x00;
	func_encendreBomba(0xFF);
}
```

4.2.1.8. mV

Código .h:

```
#ifndef mVTYPE_H
	#define mVTYPE_H

	struct mVType{

		signed long P1[8];
		signed long P2[8];

		signed long tM1[1];
		signed long tP1[1];
		signed long tP2[1];
	};

#endif
```

Código .c:

```
#include "mV.h"

extern struct mVType mV;

void inimV()
{	//variables para el valor de la muestra P1 en milivoltios
	mV.P1[0]=0.0f;
	mV.P1[1]=0.0f;
	mV.P1[2]=0.0f;
	mV.P1[3]=0.0f;
	mV.P1[4]=0.0f;
	mV.P1[5]=0.0f;
	mV.P1[6]=18000.0f;
	mV.P1[7]=0.0f;
	//variables para el valor de la muestra P2 en milivoltios
	mV.P2[0]=0.0f;
	mV.P2[1]=0.0f;
	mV.P2[2]=0.0f;
	mV.P2[3]=0.0f;
	mV.P2[4]=0.0f;
	mV.P2[5]=0.0f;
	mV.P2[6]=1000.0f;
	mV.P2[7]=0.0f;
	//variables para los valores de la temperatura en milivoltios
	mV.tP1[0]=0.0f;
	mV.tP2[0]=0.0f;
	mV.tM1[0]=0.0f;
}
```

4.2.1.9. PPM

Código .h:

```
#ifndef PPMTYPE_H
	#define PPMTYPE_H

	struct ppmType{
		signed long M1[8];
		float P1[8];
		float P2[8];
	};
#endif
```

Código .c:

```
#include "ppm.h"

extern struct ppmType ppm;

void iniPPM()
{	//variables para el valor de la muestra P1 en ppm.
	ppm.P1[0]=0.8f;
	ppm.P1[1]=40.0f;
	ppm.P1[2]=1.0f;
	ppm.P1[3]=1.0f;
	ppm.P1[4]=1.0f;
	ppm.P1[5]=1.0f;
	ppm.P1[6]=4.0f;
	ppm.P1[7]=1.0f;
	//variables para el valor de la muestra P2 en ppm.
	ppm.P2[0]=8.0f;
	ppm.P2[1]=400.0f;
	ppm.P2[2]=10.0f;
	ppm.P2[3]=10.0f;
	ppm.P2[4]=10.0f;
	ppm.P2[5]=10.0f;
	ppm.P2[6]=7.0f;
	ppm.P2[7]=10.0f;
	//variables para el valor de la muestra a medir en ppm.
	ppm.M1[0]=0.0f;
	ppm.M1[1]=0.0f;
	ppm.M1[2]=0.0f;
	ppm.M1[3]=0.0f;
	ppm.M1[4]=0.0f;
	ppm.M1[5]=0.0f;
	ppm.M1[6]=0.0f;
	ppm.M1[7]=0.0f;
}
```

4.2.1.10. OPERACIONES

Código:

```
#include "dades.h"
extern struct dataType d;

//Definimos variables globales
#define Bsolucio1     (unsigned char)4
#define Bsolucio2     (unsigned char)3
#define Bbuidat       (unsigned char)2
#define Bsample       (unsigned char)1

#define TEMPS_OMPLIR 22
#define TEMPS_OMPLIR_P1 (TEMPS_OMPLIR+4)
#define TEMPS_OMPLIR_P2 (TEMPS_OMPLIR)
#define TEMPS_OMPLIR_MOSTRA 33
#define TEMPS_NETEJA 17
#define TEMPS_NETEJA_MOSTRA 33
#define TEMPS_BUIDAR 30                 //(3*TEMPS_OMPLIR/2)
#define TEMPS_BUIDAR_MOSTRA 50
#define TEMPS_ESPERA 120
#define TEMPS_CURT 2

//Declaramos las funciones
void func_limpiar(void);
void Bomba_05s(unsigned char,unsigned char);
void Delay_05s(unsigned char);
void Delay_20s(unsigned char);
void Delay_2m(unsigned char);
void getMitja(unsigned long*);
void setControlRegSelfCal(void);
void Channel(unsigned char);
void llegueixTemperatura(void);
void sendRS232New(void);
void llegeixP1(void);
void llegeixP2(void);
void setParametres(void);
void tomV(unsigned long*,unsigned char);
void setPPM(unsigned long*,unsigned char);
void toCelcius(unsigned long*);
void func_Bomba(unsigned char);
void pauseADC(void);
void resumeADC(void);
void llegeixM(void);
void getMitjaTemp(unsigned long* pMitja);
void func_encendreBomba(unsigned char);
```

```
void medeix(void)              //Secuencia de bombas para la instrucción 'Medir'
{
        func_Bomba(B_M1);
        Delay_05s(TEMPS_CURT);
        Bomba_05s(Bbuidat,TEMPS_BUIDAR_MOSTRA);
        Delay_05s(TEMPS_CURT);
        func_Bomba(B_M1);
        Delay_05s(TEMPS_CURT);
        Bomba_05s(Bbuidat,TEMPS_BUIDAR_MOSTRA);
        Delay_05s(TEMPS_CURT);
        func_Bomba(B_M1);
        Delay_20s(d.op_temps);
        llegeixM();     //Leer muestra
        Bomba_05s(Bbuidat,TEMPS_BUIDAR_MOSTRA);
        sendRS232New();     //Enviar los valores de concentración de iones al arduino
        func_limpiar();
}

void llegeixM(void)            //Pedimos una lectura de la muestra a medir
{
        unsigned char i;
        unsigned long pMitja[1];

        resumeADC();

        for(i=0;i<d.op_numIons;i++)     //Para cada ion que tengamos
        {
                Channel(i);                 //Escoger canal del ion
                getMitja(pMitja);           //Recibir valor de la lectura del ADC
                tomV(pMitja,i);            //Convertir el valor del ADC en mV y ppm.
        }
        llegueixTemperatura();          //Leer temperatura durante la medida

        pauseADC();
}

void llegeixP1(void)                    //Pedimos una lectura de la muestra P1
{
        unsigned char i;
        unsigned long pMitja[1];

        resumeADC();
        if(d.pHCal == 0)              //Diferenciamos el calibrado de pH del resto
        {
                for(i=0;i<6;i++)
                {
                        Channel(i);
                        getMitja(pMitja);
                        tomV(pMitja,i);
        }
        }
```

```
        else
        {
                Channel(6);
                getMitja(pMitja);
                tomV(pMitja,6);
        }
        llegueixTemperatura();

        pauseADC();
}

void llegeixP2(void)                    //Pedimos una lectura de la muestra P2
{
        unsigned char i;
        unsigned long pMitja[1];

        resumeADC();
        if(d.pHCal == 0)
        {
                for(i=0;i<6;i++)
                {
                        Channel(i);
                        getMitja(pMitja);
                        tomV(pMitja,i);
                }
        }
        else
        {
                Channel(6);
                getMitja(pMitja);
                tomV(pMitja,6);
        }
        llegueixTemperatura();

        pauseADC();
}

void llegueixTemperatura(void)          //Pedimos una lectura de la temperatura
{
        unsigned long pMitja[1];
        Channel(7);
        getMitjaTemp(pMitja);
        toCelcius(pMitja);
}

void checkPoint1(void)                  //Secuencia de bombas para la instrucción 'Calibrar'
{                                       // (lectura de P1)
        func_Bomba(B_P1);
        Delay_05s(TEMPS_CURT);
        Bomba_05s(Bbuidat,TEMPS_BUIDAR);
        Delay_05s(TEMPS_CURT);
```

```
        func_Bomba(B_P1);
        Delay_20s(1);
        llegeixP1();                    //Lectura de P1
        Bomba_05s(Bbuidat,TEMPS_BUIDAR);
        sendRS232New();                 //Enviamos los valores de P1 en mV al arduino
}

void checkPoint2(void)                  //Secuencia de bombas para la instrucción 'Calibrar'
{                                       // (lectura de P2)
        func_Bomba(B_P2);
        Delay_05s(TEMPS_CURT);
        Bomba_05s(Bbuidat,TEMPS_BUIDAR);
        Delay_05s(TEMPS_CURT);
        func_Bomba(B_P2);
        Delay_20s(1);
        llegeixP2();                    //Lectura de P2
        sendRS232New();                 //Enviamos los valores de P2 en mV al arduino
        Bomba_05s(Bbuidat,TEMPS_BUIDAR);
}

void calibra(void)
{
        setParametres();                //Calculamos pendientes e intersecciones
        Delay_05s(40);
        sendRS232New();                 //Enviamos pendientes e intersecciones al arduino
        Delay_05s(40);
}

void func_limpiar(void)                 //Secuencia de bombas para limpiar la celda
{
        func_encendreBomba(B_H2O);
        func_Bomba(B_M1);
        Delay_05s(T_CURT);
        func_Bomba(B_BUIDAT);
}

void condicionar(void)                  //Secuencia de bombas para condicionar la celda
{                                       //En caso que el cliente lo especifique
        Bomba_05s(Bsolucio2,TEMPS_OMPLIR_P2);
}

void iniCella(void)             //Inicializamos la celda vaciando
{                               //para prevenir que inicie el programa con la celda llena
        Bomba_05s(Bbuidat,TEMPS_BUIDAR_MOSTRA);
}
```

4.2.1.11. FUNCIONES

Código:

```
#include "Bombes.h"
#include "dades.h"
extern struct dataType d;

// encender bomba seleccionada
void func_encendreBomba(unsigned char eBomba)
{
	switch(eBomba)
	{
		case B_M1:
			PORT_B_M1=1;
			break;
		case B_BUIDAT:
			PORT_B_B=1;
			break;
		case B_P2:
			PORT_B_P2=1;
			break;
		case B_P1:
			PORT_B_P1=1;
			break;
		case B_H2O:
			PORT_B_H2O=1;
			break;
		case B_M2:
			PORT_B_M2=1;
			break;
		case B_M3:
			PORT_B_M3=1;
			break;
		case B_M4:
			PORT_B_M4=1;
			break;
		default:
			PORT_B_P1=0;
			PORT_B_P2=0;
			PORT_B_B=0;
			PORT_B_H2O=0;
			PORT_B_M1=0;
			PORT_B_M2=0;
			PORT_B_M3=0;
			PORT_B_M4=0;
	}
}

//encender bomba seleccionada durante un tiempo
void Delay_05s(unsigned char t);
```

```
void func_Bomba(unsigned char nombomba)
{
	unsigned char t = T_DEF;
	switch (nombomba){
		case B_BUIDAT:
			t = t + 20;
			t = t + (2*d.op_metres);
			break;
		case B_P1:
			t = t + (2*d.op_metresp1);
			break;
		case B_P2:
			t = t + (2*d.op_metresp2);
			break;
		case B_M1:
			t = t + (2*d.op_metres);
			break;
		case B_H2O:
			t = t + (2*d.op_metres);
			break;
		default:
			t = 0;
			break;
	}
	func_encendreBomba(nombomba);
	Delay_05s(t);
	func_encendreBomba(0xFF);
}

void func_omplir(void)
{
	func_Bomba(d.op_bomba);
}


void Delay_20s(unsigned char t);
void llegeixM(void);
void Delay_05s(unsigned char);
void Delay_20s(unsigned char);
void getMitja(unsigned long*);
void setControlRegSelfCal(void);
void Channel(unsigned char);
void llegueixTemperatura(void);
void sendRS232New(void);
void llegeixP1(void);
void llegeixP2(void);
```

4.2.1.12. CÁLCULOS

Código:

```
#include <math.h>
#include "dades.h"
#include "mV.h"
#include "ppm.h"
#include "parametres.h"
extern struct dataType d;
extern struct mVType mV;
extern struct ppmType ppm;
extern struct paramType parametre;

void tomV(unsigned long* _adc, unsigned char i)
{	// Calculamos los mV de las muestras P1 y P2 y la concentración de la medida
	unsigned long adc = *_adc;
	float miliV = 0.0f, numerador;
	float resultat[8];
	// double mV12 = 1000 * ((2 * Vref * bits / Math.Pow(2, 24)) - Vref);
	miliV = (float)adc * 5.02f;
	miliV = miliV / (float)16777216.0f;
	miliV = miliV - 2.51f;
	miliV = miliV * 1000.0f;
	miliV = miliV * 100.0f;   //dos decimales
	switch(d.op_bomba)
	{
		case B_P1:
			mV.P1[i] = (signed long)miliV;
			break;
		case B_P2:
			mV.P2[i] = (signed long)miliV;
			break;
		case B_M1:		//Calculamos la concentración de iones en ppm
			if(parametre.m[i] != 0.0f)
			{
			numerador = miliV - (float)parametre.n[i];
			resultat[i] = numerador / (float)parametre.m[i];
			if (i<6)
				resultat[i] = (float)pow(10,resultat[i]);
			resultat[i] = resultat[i] * 100;
			}
			else
			{
				resultat[i] = 0.0f;
			}
				ppm.M1[i] = (signed long)resultat[i];
			break;
		default:
			break;
	}
}
```

```
void toCelcius(unsigned long* _adc)      //Calculamos el valor de la temperatura
{
        unsigned long adc = *_adc;
        float miliV = 0.0f;
        float resistencia = 0.0f;
        float celcius = 0.0f;
        // double mV12 = 1000 * ((2 * Vref * bits / Math.Pow(2, 24)) - Vref);
        miliV = (float)adc * 5.02f;
        miliV = miliV / (float)16777216.0f;
        miliV = miliV - 2.51f;
        miliV = miliV * 1000.0f;
        resistencia = miliV / 1.1461f;
        celcius = resistencia/100.0f;
        celcius = celcius - 1;
        celcius = celcius / 0.00385f;
        celcius = celcius * 10;  //un decimal
        switch(d.op_bomba)
        {
                case B_P1:
                        mV.tP1[0] = (signed long)celcius;
                        break;
                case B_P2:
                        mV.tP2[0] = (signed long)celcius;
                        break;
                case B_M1:
                        mV.tM1[0] = (signed long)celcius;
                        break;
                default:
                        break;
        }
}

void setParametres()            //Calculamos las pendientes y las intersecciones
{
        unsigned char i;
        float logP1,logP2;
        float numerador,denominador;
/*
pendent[i] = (p1.m[i] - p2.m[i]) / (Math.Log10(sol.P1[i]) - Math.Log10(sol.P2[i]));
interseccio[i] = p1.m[i] - (pendent[i] * Math.Log10(sol.P1[i]));
*/
        for(i = 0; i < 6; i++)
        {
                ppm.P1[i] = ppm.P2[i] / 10;
        }
        for(i = 0; i < d.op_numIons; i++)
        {
                if(i<6) {                   //Diferenciamos los cálculos del calibrado de pH
                        logP1=log10(ppm.P1[i]);
                        logP2=log10(ppm.P2[i]);
                        denominador = logP1 - logP2;
```

```
                }
                else
                        denominador = ppm.P1[i] - ppm.P2[i];
                if(denominador != 0.0f)
                {
                        if(i < 6)
                        {
                            numerador = (float)mV.P1[i] - (float)mV.P2[i];
                            parametre.m[i] = numerador / denominador;
                            numerador = (float)parametre.m[i] * logP1;
                        }
                        else
                        {
                            numerador = (float)mV.P1[i] - (float)mV.P2[i];
                            parametre.m[i] = numerador / denominador;
                            numerador = (float)parametre.m[i] * ppm.P1[i];
                          }
                parametre.n[i] = mV.P1[i] - numerador;
                }
                else
                {
                        parametre.m[i] = 0.0f;
                        parametre.n[i] = 0.0f;
                }
        }
        parametre.tCal[0] = (mV.tP1[0] + mV.tP2[0]) / 2;
}
```

4.2.2. Arduino

Código:

```
# include <SoftwareSerial.h>
# include <SD.h>

// variables constantes:
SoftwareSerial mySerial(0, 1);   // RX, TX

const int chipSelect = 4;    // Guardamos en que entrada de Arduino está conectado el pin CS del módulo SD.
const int calibrarPin = 6;   // Número del pin al cual conectaremos el pulsador para calibrar.
const int medirPin = 5;   // Número del pin al cual conectaremos el pulsador para medir.
const int automaticoPin = 7;   // Número del pin al cual conectaremos el interruptor para calibrar y medir en automático.
const int calibrarphPin = 8;   // Número del pin al cual conectaremos el pulsador para calibrar pH.
const int analogOutPin = 9;      // Pin 9 salida analógica
const int analogOutPin2 = 3;    // Pin 3 salida analógica

// Variables:
```

```
int outputValue = 0;        // valor del ciclo de trabajo del PWM del pin 9.
int outputValue2 = 0;       // valor del ciclo de trabajo del PWM del pin 3.
int calibrarState = LOW;        // variable para leer el estado del pulsador de calibrar.
int medirState = LOW;       // variable para leer el estado del pulsador de medir.
int automaticoState = LOW;        // variable para leer el estado del interruptor de
calibrar y medir en automático.
int calibrarphState = LOW;      // variable para leer el estado del pulsador de calibrar pH.
char toSend[4];                 // vector con la trama que se envía al microcontrolador.
byte toRead[80];   // vector donde se guarda la trama recibida del microcontrolador.
long minutos = 300, countmin = 300, c = 600, ccount = 600;   // variables para el control
del tiempo en el modo automático.
int calibrarcada = 16, countcal = 0; // variables con las cuales establecemos cada cuantas
mediciones queremos hacer un calibrado en el modo automático.
String iones[8] = {"NH4", "Cl", "K", "Na", "Ca", "NO3", "pH", "T"}; // Strings con el
nombre de cada uno de los iones para la SD.
String piones[8] = {"pNH4", "pCl", "pK", "pNa", "pCa", "pNO3", "ppH", "pT"};
String Iiones[8] = {"iNH4", "iCl", "iK", "iNa", "iCa", "iNO3", "ipH", "iT"};
double pendientes[8];       // vector donde se guardan las pendientes del calibrado de
cada uno de los iones.
double intersecciones[8];   // vector donde se guardan las intersecciones del calibrado de
cada uno de los iones.
double m[8];                // vector donde se guardan las medidas de cada uno de los iones.
double m1;
double temperatura;         // variable para la temperatura.
int estado;
char operacio;
int tmp;
long time;

void setup() {

  // El pin CS por defecto de la placa arduino debe ser configurado como salida.
  pinMode(10, OUTPUT);
  // inicializamos los pines donde conectamos los pulsadores como entradas:
  pinMode(calibrarPin, INPUT);
  pinMode(medirPin, INPUT);
  pinMode(automaticoPin, INPUT);
  pinMode(calibrarphPin, INPUT);

  Serial.begin(9600);  //abre el puerto serie y establece la velocidad en barios de la
transmisión de datos.

  // Comprobamos que la tarjeta SD se lee correctamente.
  if (!SD.begin(chipSelect)){
// En este punto podemos informar por el puerto serie si ha habido error al leer la tarjeta.
    //Serial.println("Error al leer la tarjeta.");
    return;
  }
  delay(5000);     //retraso de 5s.

}
```

```
void loop() {

 // leemos el estado de los pulsadores y el interruptor:
  calibrarState = digitalRead(calibrarPin);
  medirState = digitalRead(medirPin);
  automaticoState = digitalRead(automaticoPin);
  calibrarphState = digitalRead(calibrarphPin);

  // En caso de que el interruptor para el modo automático este accionado enviamos al
microcontrolador la instrucción de calibrar o medir según toque por tiempos.
  if ((automaticoState == HIGH)&&(Serial.available() == 0)) {
   if (ccount == c) {
   ccount = 0;
   if (countmin == minutos) {
    countmin = 0;
    if (countcal % calibrarcada == 0) {
     toSend[0] = 0x03;   // el número 3 es el valor que hemos definido como la
instrucción de calibrar.
     toSend[1] = 0x00;
     toSend[2] = 0x00;
     toSend[3] = 0x00;
     Serial.println(toSend);
     countcal = 0;
    }
    else {
     toSend[0] = 0x02;  // el número 2 es el valor definido como la instrucción de medir.
     toSend[1] = 0x00;
     toSend[2] = 0x00;
     toSend[3] = 0x00;
     Serial.println(toSend);
    }
    countcal++;
   }
   countmin++;
   }
   ccount++;
  }

  // En caso de recibir una trama a través del puerto serie:
  if (Serial.available()>0) {
   for(int i = 0; Serial.available()>0; i++)  //Mientras haya datos en el buffer haz:
    {
      delay(50);   //Ponemos un pequeño delay para mejorar la recepción de datos.
      toRead[i]=Serial.read();  //Lee un carácter
    }

   operacio = toRead[1];  // diferenciamos si la operación recibida es calibrar (01) o
medir (02)

switch (operacio) {
```

```
    case 05:
// Este caso está definido en principio para recibir únicamente el estado del microcontrolador, es decir, calibrando, leyendo o en reposo... para posibles futuros usos.
      estado = toRead[1];
      break;
    case 01:          //Recibe una calibración, guardamos pendientes, intersecciones y temperatura.
      for (int j = 0; j < 8; j++) {
        tmp = toRead[3 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[4 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[5 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[6 + 4 * j];
        pendientes[j] = tmp;
        pendientes[j] = pendientes[j] / 100;  // dividimos por 100 para obtener dos decimales, previamente el valor ha sido multiplicado por 100 en el microcontrolador.
      }
      for (int j = 0; j < 8; j++) {
        tmp = toRead[36 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[37 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[38 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[39 + 4 * j];
        intersecciones[j] = tmp;
        intersecciones[j] = intersecciones[j] / 100;
      }
      tmp = toRead[69];
      tmp <<= 8;
      tmp |= toRead[70];
      tmp <<= 8;
      tmp |= toRead[71];
      tmp <<= 8;
      tmp |= toRead[72];
      temperatura = tmp;
      temperatura = temperatura / 10;
      break;
    case 02:     // Recibe una medición.
     for (int j = 0; j < 8; j++) {
        tmp = toRead[3 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[4 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[5 + 4 * j];
        tmp <<= 8;
        tmp |= toRead[6 + 4 * j];
        m[j] = tmp;
        m[j] = m[j] / 100;
```

```
        }
        tmp = toRead[36];
        tmp <<= 8;
        tmp |= toRead[37];
        tmp <<= 8;
        tmp |= toRead[38];
        tmp <<= 8;
        tmp |= toRead[39];
        temperatura = tmp;
        temperatura = temperatura / 10;

        //for (int i = 0; i < 8; i++) {
        m1 = m[0] * 10;      // esto es debido a la escala que queremos utilizar a la hora de
enviar el valor de las medidas por la señal PWM.
        //}
        break;
      default:
        break;
    }

   // escribimos en la tarjeta SD
    //si la operación recibida es un calibrado o medidas.
    if (operacio == 1 || operacio == 2){

         File dataFile = SD.open("NTS.txt", FILE_WRITE); //Creamos un archivo ".txt"
en la tarjeta SD con el nombre NTS.
         if (dataFile) {
           time = millis();  // la función millis() cuenta los milisegundos desde que se inicia
el programa.
           time = time/1000;
           time = time/60;
           dataFile.print("time: ");
           delay(50);
           dataFile.print(time); // escribimos en la tarjeta el valor recibido por la función
millis() para tener una referencia de tiempo ya que aún no disponemos de reloj.
           delay(500);
           dataFile.println(" minutos");
           delay(500);

           if (operacio == 1){     // En caso de recibir un calibrado, escribimos las pendientes
e intersecciones recibidas.
             dataFile.println("-----Pendientes-----");
             delay(5);
             for (int i = 0; i < 8; i++) {
               dataFile.print(piones[i] + ": ");
               delay(5);
               dataFile.println(pendientes[i]);
               delay(5);
             }
             dataFile.println("-----Intersecciones-----");
             delay(5);
```

```
          for (int i = 0; i < 8; i++) {
            dataFile.print(Iiones[i] + ": ");
            delay(5);
            dataFile.println(intersecciones[i]);
            delay(5);
          }
        }

      else{        // en caso de recibir las medidas, escribimos estas en la tarjeta SD.
          dataFile.println("-----Medidas-----");
          delay(5);
          for (int i = 0; i < 8; i++) {
            dataFile.print(iones[i] + ": ");
            delay(5);
            dataFile.println(m[i]);
            delay(5);
          }
        }

       // Cerramos el archivo.
       dataFile.close();
       } else{
         // En este punto podemos informar si hay algún problema a la hora de escribir en
la tarjeta SD.
         //Serial.println("Error al escribir en NTS.txt");
       }
   }
 }

  // Comprobamos si los pulsadores de calibrar, medir y calibrar pH han sido
presionados.
  // En caso de que así sea, enviamos la trama correspondiente al microcontrolador:
  if (calibrarState == HIGH) {
   toSend[0] = 0x01;    // el valor 01 representa la instrucción calibrar.
   toSend[1] = 0x00;
   toSend[2] = 0x00;
   toSend[3] = 0x00;
   Serial.println(toSend);
  }
  if (medirState == HIGH) {
   toSend[0] = 0x02;    // el valor 02 representa la instrucción medir.
   toSend[1] = 0x00;
   toSend[2] = 0x00;
   toSend[3] = 0x00;
   Serial.println(toSend);
  }
  if (calibrarphState == HIGH) {
   toSend[0] = 0x04;    // el valor 04 representa la instrucción calibrar pH.
   toSend[1] = 0x00;
   toSend[2] = 0x00;
   toSend[3] = 0x00;
   Serial.println(toSend);
```

```
  }

  // la función map() nos permite convertir un rango de variación en otro.
  // En nuestro caso pasamos del rango sobre el cual varia el valor de una medida de
concentración de uno de los iones al rango de variación del ciclo de trabajo del PWM.
  outputValue = map(m1, 0, 100, 0, 255);  // Valor del ciclo de trabajo del PWM pin 9.
  outputValue2 = map(m[1], 0, 1000, 0, 255);  // Ciclo de trabajo del PWM del pin 3.
  // Cambiamos el valor de las salidas analógicas:
  analogWrite(analogOutPin, outputValue);
  analogWrite(analogOutPin2, outputValue2);
  delay(10); // esperamos 10 ms antes de reiniciar el loop.
}
```

5. RESULTADOS EXPERIMENTALES

5.1. Introducción

Por último, una vez completada la fase de montaje y comprobado el funcionamiento del equipo, se obtienen los resultados experimentales que vamos a poder observar en las siguientes gráficas obtenidas mediante un data logger con entrada analógica 4-20 mA y con los fitxeros de la tarjeta SD. En los resultados experimentales destaca la fiabilidad de los datos obtenidos mediante la señal 4-20 mA y el correcto funcionamiento de todo el equipo.

Es importante mencionar que el equipo ha pasado un periodo de prueba en la empresa AGBAR, durante el cual los resultados obtenidos han sido bastante satisfactorios. AGBAR, es el nombre que recibe la Sociedad General de Aguas de Barcelona, entre otras cosas se encargan de la gestión del ciclo integral del agua, es decir, desde la captación hasta la potabilización, transporte y distribución. Además, se encarga del servicio de saneamiento y depuración de aguas residuales para su retorno al medio natural o su reutilización.

En el centro de la imagen, podemos ver a nuestro AMI C7 una vez instalado en la empresa AGBAR. En este caso concreto nos pidieron que nuestro AMI C7 pudiera analizar las concentraciones de iones de amonio (NH_4), cloro (Cl) y pH en un punto concreto de un proceso de filtrado de aguas.

5.2. Datos guardados en la tarjeta SD

En la siguiente fotografía vemos el formato con el cual se guardan los datos registrados en la tarjeta SD. Como ya se ha mencionado con anterioridad, los datos se guardan en la tarjeta SD en un fichero al que hemos denominado por software *"NTS.txt"*. En este caso estamos midiendo una solución, hecha a ojo por nosotros mismos, con aproximadamente 2 ppm de amoni (NH_4^+) y 40 ppm de clor (Cl^-).

```
time: 5 minutos
-----Pendientes-----
pNH4: 49.37
pCl: -37.94
pK: 15.41
pNa: 15.44
pCa: 11.83
pNO3: 51.84
ppH: -56.66
pT: 0.00
-----Intersecciones-----
iNH4: -82.18
iCl: 253.95
iK: -260.40
iNa: -183.72
iCa: 321.01
iNO3: 200.42
ipH: -248.72
iT: 0.00
time: 8 minutos
-----Medidas-----
NH4: 1.99
Cl: 41.65
K: 6.90
Na: 2.47
Ca: 2.60
NO3: 2.48
pH: 6.97
T: 0.00
```

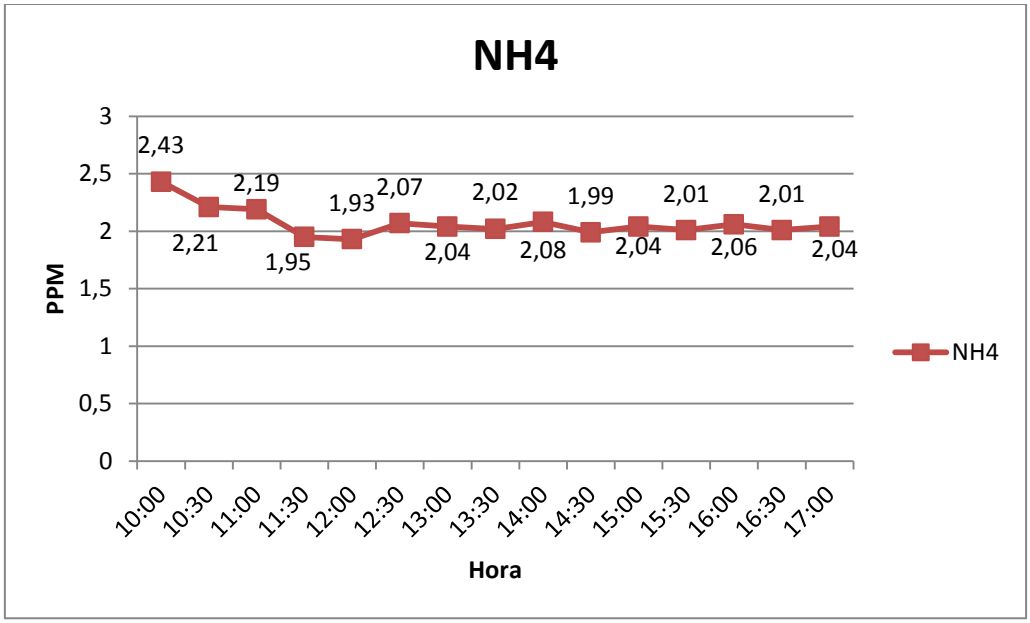
Como se puede observar en la figura tenemos tres secciones: pendientes, intersecciones y medidas, las dos primeras se registran en la tarjeta SD al hacer la calibración del instrumento de medida; la última, medidas, se guarda al realizar una medida de la concentración de iones en la muestra que se desea analizar. Se aprecia también que en primer lugar se indica el tiempo en el que se realiza cada acción como no disponemos de un reloj, se marcan los minutos desde que se inicio el AMI C7, esto se lleva a cabo gracias a la función *"millis ( )"* de Arduino que cuenta los milisegundos desde que se inicio el programa. Por otro lado, en el nombre de los iones vemos escrita una 'p' delante del nombre en el caso de las pendientes y una 'i' en el caso de las intersecciones, esto es a petición de la empresa para que sea más fácil separar los datos a posteriori.

En la figura, vemos que aparecen los nombres de siete iones mas la temperatura 'T', en realidad los únicos que realmente se están analizando son los iones NH4 y Cl, pero hemos dejado que escriba el resto de iones para que se vea que aunque por la salida 4-20 mA únicamente se analizen uno o dos iones, en la memoria SD se puede registrar el valor de todos.

Si vemos el resultado final de la medida, tanto el valor de iones de amonio como de cloro se aproxima bastante al valor que queríamos darle a la solución acuosa utilizada, hemos de tener en cuenta que la precisión con la que hemos añadido la concentración de iones a la muestra no es 100% exacta, con lo que creo que el resultado obtenido es bastante precisso. Hemos de decir también que en un principio hemos necesitado dejar acondicionar los electrodos en la muestra para que los valores obtenidos fueran mas estables y precisos.

5.3. Datos recibidos por la señal de salida de 4-20 mA

En este caso conseguimos los datos con un data logger con entrada analógica de 4-20 mA, al que previamente hemos introducido la relación entre el rango de valores de la señal de 4 a 20 mA y el rango de valores de las concentraciones de iones de NH4 que es de 0 a 10 ppm, tenemos que tener en cuenta que contamos con un decimal significativo. En este caso medimos, desde las 10:00 hasta las 17:00 en periodos de media hora (15 medidas), una solución hecha por nosotros mismos con aproximadamente 2 ppm de amonio.



Como se aprecia en la imagen el resultado obtenido tarda en estabilizarse, coma pero a partir de la tercera medida vemos que la señal es bastante estable con un error que no supera el 10%, recordemos que solo contamos con un decimal significativo por lo que creemos que los resultados obtenidos una vez estabilizada la señal són bastante precisos. Vemos que principalmente en la primera medida el error obtenido es muy alto, esto se debe principalmente a que el electrodo necesita un breve periodo de acondicionamiento antes de empezar a medir.

5.4. Comparación entre los datos de la tarjeta SD y los de la señal 4-20 mA

Tarjeta SD	2,49	2,28	2,14	1,99	1,97	2,01	2,07	2,05	2,03	1,94	2,00	2,03	2,01	2,03	2,07
Señal 4-20	2,43	2,21	2,19	1,95	1,93	2,07	2,04	2,02	2,08	1,99	2,04	2,01	2,06	2,01	2,04

En esta tabla comparamos las medidas del señal 4-20 mA estudiadas en el apartado anterior con las recogidas en la tarjeta de memoria SD. Como se puede apreciar los resultados son prácticamente idénticos si quitamos el segundo decimal de las medidas de la señal 4-20 mA, que como ya hemos explicado por cuestiones de escalas no es significativo. Por lo tanto por el señal 4-20 mA estamos leyendo el mismo valor que el Arduino recibe de el microcontrolador PIC18F4550.

6. CONCLUSIONES

6.1. Conclusiones finales

En este proyecto se ha diseñado, realizado y comprobado parte del prototipo de un producto para la empresa NT SENSORS. El producto trata de un sensor ISE (Electrodo selectivo de iones) con un potenciómetro, el cual automáticamente o por indicación del usuario a través de pulsadores, es capaz de realizar su propia calibración a partir de dos muestras y dar la información de las medidas a través de un señal 4-20 mA o escribiendo la información en una tarjeta SD.

Este potenciómetro se alimenta de la corriente de la red, gracias a diversos transformadores, esto se debe a que de este modo es más cómodo para el usuario. En principio el potenciómetro en si se alimenta con una tensión de 24 V y una corriente de unos 3 A.

Hemos de decir que para realizar este prototipo, partíamos de producto previo cuya función era similar al nuestro, pero debía estar conectado a un ordenador personal, por lo que un programa de ordenador era desde donde se realizaban todos los cálculos, se interactuaba con el usuario y donde se recibía la información de las medidas y las calibraciones.

El trabajo práctico realizado en nuestro proyecto principalmente ha sido:

- Entender el software (del microcontrolador y del ordenador) y el hardware del producto anterior.
- Introducir todos los cálculos de las calibraciones y las medidas en el software del microcontrolador.
- Añadir al producto la incorporación de un arduino, lo cual aumenta el rango de mejora del producto. He incluso invirtiendo más tiempo y algo más de presupuesto, el arduino podría realizar las tareas de microcontrolador.
- Establecer la interacción con el usuario a través de pulsadores, de este modo eliminamos completamente la dependencia de un ordenador personal.
- Diseñar y realizar la comunicación entre microcontrolador y arduino.
- Escribir los datos de salida, tanto pendientes e intersecciones de las calibraciones como las medidas de concentración de iones realizadas, en una tarjeta SD.
- Diseñar, realizar y comprobar una salida de los datos de las medidas de concentración de iones a través de una señal 4-20 mA.

En primer lugar, se inició el proyecto intentando sacar una señal de 4-20 mA con los datos de las concentraciones de iones directamente del microcontrolador PIC18F4550 del primer potenciómetro, ya que el requisito de la señal 4-20 mA era uno de los más importantes ya que varias empresas habían preguntado por ello. Se intentó con el chip AD421, convertidor de una señal digital a una señal de corriente de 4-20 mA, pero no logramos realizar la comunicación entre ambos chips. Entonces surgió la idea de incorporar el arduino a nuestro producto ya que de este modo también se abría la posibilidad de implementar otras mejoras que en la empresa ya tenían en mente.

Finalmente con la incorporación del arduino y un nuevo diseño de la salida de 4-20 mA se consiguió realizar el producto. Como se comenta en los resultados experimentales, el resultado final en cuanto a la señal de 4-20 mA tiene algunas restricciones en cuanto a nombre de decimales significativos, dependiendo de la amplitud del rango de valores de cada concentración de iones. Esto es debido a que el ciclo de trabajo de la señal PWM que sale de nuestro arduino tiene un rango de ciclo de trabajo de 0 a 255, con lo cual tenemos que pasar el valor de la concentración de un rango de valores a otro y esto hace que si el rango de valores de las concentraciones de iones es muy alto se pierda precisión a la hora de convertirlo en una señal PWM y luego en una señal 4-20 mA. Pero en principio la precisión obtenida es la suficiente según las exigencias de la empresa.

Por otro lado, los datos que se recibe el arduino y que se escriben en la tarjeta SD, son exactamente los mismos que se obtenían con el producto anterior por lo que por esta banda la precisión del producto sigue siendo la misma.

Como conclusión final podemos decir que se han cumplido los objetivos marcados e incluso cabe destacar que nuestro producto ha pasado un periodo de prueba en una empresa de filtraje del agua y los resultados obtenidos han sido los correctos.

7. MEDIDAS Y PRESUPUESTO

8.1. Medidas

8.1.1. Capítulo 1-Etapa µC PIC18F4550

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
9321357	**u Microcontrolador PIC18F4550**								
	Microcontrolador Microchip PIC18F4550-I/P MCU, 8BIT, PIC18, 48MHZ, DIP-40.								
		1				1			
							1		
1438538	**u Convertidor A/D AD7712**								
	ANALOG DEVICES AD7712ANZ IC, 24BIT ADC S-DELTA, SIGNAL COND .								
		1				1			
							1		
1077111	**u Multiplexor DG408DJ**								
	VISHAY SILICONIX DG408DJ-E3 IC, MUX 8 CH SP, DIP16.								
		1				1			
							1		
9486070	**u AO LMC6484**								
	Amplificador operacional TEXAS INSTRUMENTS, cuádruple, 1.5 MHz.								
		10				10			
							10		
1097452	**u AO OPA4277**								
	Amplificador operacional TEXAS INSTRUMENTS, cuádruple, 1 MHz.								
		10				10			
							10		

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
2323589	**u Inversor LT1054IP**								
	TEXAS INSTRUMENTS DC-DC CONV, DIP-8.								
		1				1			
							1		
9758640	**u AO MCP6022-I/P**								
	Amplificador Operacional MICROCHIP, Dual, 10 MHz, 2, 7 V/µs.								
		10				10			
							10		
1468834	**u Ref. de tensión LM4040A**								
	Referencia de tensión TEXAS INSTRUMENTS, fija, 5V, TO-226AA-3								
		1				1			
							1		
1627197	**u MCP3201-BI/P**								
	MICROCHIP MCP3201-BI/P 12BIT ADC, 2.7V, 1CH, SPI, 8-DIP.								
		1				1			
							1		
719-2768	**u LDO NCP7805TG**								
	Regulador de tensión lineal, NCP7805TG, 1A, 5 V, TO-220, 3 pines.								
		10				10			
							10		
1324808	**u Disipador de calor**								
	Disipador ABL 1ºC/W (100x120x77)mm.								
		1				1			
							1		

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
0389722	**u Transformador AC/DC**								
	Transformador AC 110 V / 220 V a 24 V DC, 10.5 A, 250 W.								
		1				1			
							1		
1578401	**u LDO MCP1825S-3002E/AB**								
	MICROCHIP MCP1825S-3002E/AB IC, LDO, 3.0 V, 500 mA, TO-220-3.								
		10				10			
							10		
1459126	**u Transistor KSP13BU**								
	Transistor de Unión Bipolar Único ON SEMICONDUCTOR , Darlington, NPN, 30 V, 125 MHz, 625 mW, 500 mA								
		25				25			
							25		
7948025	**u Resonador de 20 MHz**								
	Resonador cerámico ABRACON, 20 MHz.								
		10				10			
							10		
1652574	**u Resonador de 10 MHz**								
	RESONADOR CERÁMICO ABRACON, 10 MHz.								
		10				10			
							10		
1692372	**u CONDENSADORES 1 µF**								
	Condensador EPCOS, 1 µF, 63 V.								
		50				50			
							50		

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
1141778	**u Condensador de 0.22 µF**								
	Condensador cerámico VISHAY, 0.22 µF, ±10%, 50 V, X7R.								
		50				50			
							50		
4903225	**u Resistencia de 100 kΩ**								
	VISHAY BC COMPONENTS METAL FILM RESISTOR, 100 kohm, ± 1%.								
		10				10			
							10		
8767483	**u Condensador de 10 µF**								
	Condensador electrolítico PANASONIC, 10 µF, 63 V.								
		50				50			
							50		
1466399	**u Resistencia de 10 kΩ**								
	VISHAY BC COMPONENTS METAL FILM RESISTOR, 10 kohm, ± 1%								
		50				50			
							50		
1100407	**u Condensador de 10 nF**								
	Condensador Cerámica Multicapa AVX, 0.01 µF, ± 10%, X7R, 100 V								
		50				50			
							50		
8767254	**u Condensador de 100 µF**								
	Condensador Electrolítico PANASONIC, 100 µF, ± 20%, 35 V.								
		50				50			
							50		

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
1833325	**u Resistencia de 2,2 kΩ**								
	Resistencia WELWYN, 2K2, 500 mW, 1%								
		10				10			
							10		
1833334	**u Resistencia de 220 kΩ**								
	Resistencia WELWYN, 220R, 500 mW, 1%								
		10				10			
							10		
1127964	**u Resistencia 68 kΩ**								
	Resistencia MULTICOMP, 68 kohm, 500 mW, 5%								
		10				10			
							10		
1833304	**U Resistencia de 1 kΩ**								
	Resistencia WELWYN, 1 k, 500MW, 1%								
		50				50			
							50		
1101345	**u Zócalo, DIL, 0.3", 8 VIAS**								
	Zócalo DIL de 8 vías para integrado								
		5				5			
							5		
1101346	**u Zócalo, DIL, 0.3", 14 VIAS**								
	Zócalo DIL de 14 vías para integrado								
		5				5			
							5		

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
1101347	**u Zócalo, DIL, 0.3", 16 VIAS**								
	Zócalo DIL de 16 vías para integrado								
		2				2			
							2		

8.1.2. Capítulo 2-Etapa Arduino

Código	Descripción	U.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
2075382	**u Arduino UNO**								
	Sensor Efecto Hall LEM, LA 55-P, Current Transducer, 50 A, PCB.								
		1				1			
							1		
1487840	**u Módulo SD para Arduino**								
	Módulo de lectura/escritura en tarjetas SD para Arduino.								
		1				1			
							1		
1469469	**u Optoacoplador 6N136**								
	OPTOACOPLADOR VISHAY SEMICONDUCTOR , FOTOTRANSIST, 5300 V_{RMS}								
		10				10			
							10		
1461144	**u Convertidor V/I XTR110**								
	TEXAS INSTRUMENTS XTR110BG IC, CONVERTIDOR V/I, PRECISION								
		2				2			
							2		

Código	Descripción	U.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
9103546	**u Mosfet P-Channel IRF9510**								
	VISHAY SEMICONDUCTOR Transistor MOSFET, Canal P, 3 A, 100 V, 1.2 ohm.								
		5				5			
							5		
2309119	**u Potenciómetro 100 k**								
	Potenciómetro ajustable, 23 vueltas, 100 k								
		5				5			
							5		
1634626	**u Pulsador**								
	MULTICOMP INTERRUPTOR, BLACK PUSH BUTTON, SPDT								
		3				3			
							3		
9968407	**u Interruptor**								
	Mini interruptor empotrable STD On/Off R81 MDT.								
		1				1			
							1		
9731148	**u Conector 2 Posiciones**								
	Conector MOLEX, 2 posiciones, 2.54 mm								
		7				7			
							7		

Código	Descripción	u.	Long.	Ampl.	Alt.	Parcial	Cant.	Preu U.	Importe
143126	**u Conector hembra 2 Posiciones**								
	Conector hembra MOLEX, 2 posiciones, 2.54 mm								
		7				7			
							7		
9773789	**u Contacto para cables**								
	Contacto para cables MOLEX, 22-30AWG, CRIMP, (paquete de 100 u)								
		1				1			
							1		

8.1.3. Capítulo 3-Mano de Obra

Código	Descripción	U.	Long.	Ampl.	Alt.	Parcial	Cant.	Precio U.	Importe
A025126	**h Ingeniero técnico Industrial - Grupo III**								
	Ingeniero técnico industrial realizador de los convertidores.								
		150				150			
							150		
A025327	**h Ingeniero técnico Industrial - Grupo II**								
	Ingeniero técnico industrial supervisor.								
		20				20			
							20		

8.2. Precios Unitarios

8.2.1. Capítulo 1-Etapa µC PIC18F4550

Código	U	Descripción	Precio	
9321357	U	U Microcontrolador PIC24F4550	5.99	Cinco euros con noventa y nueve céntimos
1438538	U	U Convertidor AD AD7712	30.11	Treinta euros con once céntimos
1077111	U	U Multiplexor DG408DJ	2.29	Dos euros con veintinueve céntimos
9486070	U	U AO cuádruple LMC6484	3.13	Tres euros con doce céntimos
1097452	U	U AO cuádruple OPA4277	7.22	Siete euros con veintidós céntimos
2323589	U	U Inversor LT1054IP	3.70	Tres euros con setenta céntimos
9758640	U	U AO doble MCP6022-I/P	1.15	Un euro con quince céntimos
1468834	U	U Referencia de tensión LM4040A	2.06	Dos euros con seis céntimos
1627197	U	U Convertidor AD MCP3201-BI/P	2.60	Dos euros con sesenta céntimos
719-2768	U	U LDO NCP7805TG	0.304	Treinta céntimos con cuatro
1324808	U	U Disipador de calor ABL (100x120x77)mm	12.48	Doce euros con cuarenta y ocho céntimos
0389722	U	U Transformador AC/DC	23.67	Veintitrés euros con sesenta y siete céntimos
1578401	U	U LDO MCP1825S-3002E/AB	0.552	Cincuenta y cinco céntimos con dos
1459126	U	U Transistor Bipolar KSP13BU	0.138	Trece céntimos con ocho
7948025	U	U Resonador de 20 MHz	0.23	Veintitrés céntimos
1652574	U	U Resonador de 10 MHz	0.344	Treinta y cuatro céntimos con cuatro
1692372	U	U Condensador 1 µF	0.337	Treinta y tres céntimos con siete
1141778	U	U Condensador de 0,22 µF	0.188	Dieciocho céntimos con ocho
4903225	U	U Resistencia de 100 kΩ	0.051	Cinco céntimos con uno
8767483	U	U Condensador de 10 µF	0.0576	Cinco céntimos con setenta y seis
1466399	U	U Resistencia de 10 kΩ	0.023	Dos céntimos con tres
1100407	U	U Condensador de 10 nF	0.216	Veintiún céntimos con seis
8767254	U	U Condensador de 100 µF	0.0922	Nueve céntimos con veintidós

1833325	U	U Resistencia de 2,2 kΩ	0.073	Siete céntimos con tres
1833334	U	U Resistencia de 220 kΩ	0.073	Siete céntimos con tres
1127964	U	U Resistencia de 68 kΩ	0.025	Dos céntimos con cinco
1833304	U	U Resistencia de 1 kΩ	0.073	Siete céntimos con tres
1324808	U	U Zócalo, DIL, 0.3", 8 VIAS	0.146	Catorce céntimos con seis
1324808	U	U Zócalo, DIL, 0.3", 14 VIAS	0.146	Catorce céntimos con seis
1324808	U	U Zócalo, DIL, 0.3", 16 VIAS	0.17	Diecisiete céntimos

8.2.2. Capítulo 2-Etapa Arduino

Código	U	Descripción	Precio	
2075382	U	U Arduino UNO	24.29	Veinticuatro euros con veintinueve céntimos
1487840	U	U Módulo SD para Arduino	2.23	Dos euros veintitrés céntimos
1469469	U	U Optoacoplador 6N136	1.37	Un euro con treinta y siete céntimos
1461144	U	U Convertidor V/I XTR110	68.55	Sesenta y ocho euros con cincuenta y cinco céntimos
9103546	U	U Mosfet P-Channel IRF9510	1.08	Un euro con ocho céntimos
2309119	U	U Potenciómetro 100 kΩ	1.13	Un euro con trece céntimos
1634626	U	U Pulsador	3.57	Tres euros cincuenta y siete céntimos
9968407	U	U Interruptor	0.95	Noventa y cinco céntimos
9731148	U	U Conector MOLEX 2 posiciones	0.31	Treinta y un céntimos
143126	U	U Conector hembra MOLEX 2 posiciones	0.22	Veintidós céntimos
9773789	U	U Contacto para cables	10.02	Diez euros con dos céntimos

8.2.3. Capítulo 3-Mano de Obra

Código	U	Descripción	Precio	
A025126	H	H Ingeniero técnico Industrial - Grupo III	12.50	Doce euros con cincuenta céntimos
A025327	H	H Ingeniero técnico Industrial - Grupo II	17.50	Diecisiete euros con cincuenta céntimos

8.3. Presupuesto

8.3.1. Capítulo 1 – Etapa µC PIC18F4550

Código	Descripción	U	Long.	Ampl.	Alz.	Parcial	Cant.	Precio Unid.	Importe
9321357	**u Microcontrolador PIC18F4550**								
	Microcontrolador Microchip PIC18F4550-I/P MCU, 8BIT, PIC18, 48MHZ, DIP-40.								
		1				1			
							1	5.99	5.99
1438538	**u Convertidor A/D AD7712**								
	ANALOG DEVICES AD7712ANZ IC, 24BIT ADC S-DELTA, SIGNAL COND .								
		1				1			
							1	30.11	30.11
1077111	**u Multiplexor DG408DJ**								
	VISHAY SILICONIX DG408DJ-E3 IC, MUX 8 CH SP, DIP16.								
		1				1			
							1	2.29	2.29
9486070	**u AO LMC6484**								
	Amplificador operacional TEXAS INSTRUMENTS, cuádruple, 1.5 MHz.								
		10				10			
							10	3.13	31.3
1097452	**u AO OPA4277**								
	Amplificador operacional TEXAS INSTRUMENTS, cuádruple, 1 MHz.								
		10				10			
							10	7.22	72.2

Código	Descripción	U	Long.	Ampl.	Alz.	Parcial	Cant.	Precio Unid.	Importe
2323589	**u Inversor LT1054IP**								
	TEXAS INSTRUMENTS DC-DC CONV, DIP-8.								
		1				1			
							1	3.70	3.70
9758640	**u AO MCP6022-I/P**								
	Amplificador Operacional MICROCHIP, Dual, 10 MHz, 2, 7 V/µs.								
		10				10			
							10	1.15	11.5
1468834	**u Ref. de tensión LM4040A**								
	Referencia de tensión TEXAS INSTRUMENTS, fija, 5V, TO-226AA-3								
		1				1			
							1	2.06	2.06
1627197	**u Convertidor AD MCP3201-BI/P**								
	MICROCHIP MCP3201-BI/P 12BIT ADC, 2.7V, 1CH, SPI, 8-DIP.								
		1				1			
							1	2.60	2.60
719-2768	**u LDO NCP7805TG**								
	Regulador de tensión lineal, NCP7805TG, 1A, 5 V, TO-220, 3 pines.								
		10				10			
							10	0.304	3.04
1324808	**u Disipador de calor**								
	Disipador ABL 1ºC/W (100x120x77)mm.								
		1				1			
							1	12.48	12.48

Código	Descripción	u	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
0389722	**u Transformador AC/DC**								
	Transformador AC 110 V / 220 V a 24 V DC, 10.5 A, 250 W.								
		1				1			
							1	23.67	23.67
1578401	**u LDO MCP1825S-3002E/AB**								
	MICROCHIP MCP1825S-3002E/AB IC, LDO, 3.0 V, 500 mA, TO-220-3.								
		10				10			
							10	0.552	5.52
1459126	**u Transistor KSP13BU**								
	Transistor de Unión Bipolar Único ON SEMICONDUCTOR , Darlington, NPN, 30 V, 125 MHz, 625 mW, 500 mA								
		25				25			
							25	0.138	3.45
7948025	**u Resonador de 20 MHz**								
	Resonador cerámico ABRACON, 20 MHz. .								
		10				10			
							10	0.23	2.30
1652574	**u Resonador de 10 MHz**								
	RESONADOR CERÁMICO ABRACON, 10 MHz. .								
		10				10			
							10	0.344	3.44
1692372	**u CONDENSADORES 1 µF**								
	Condensador EPCOS, 1 µF, 63 V.								
		50				50			
							50	0.337	16.85

Código	Descripción	u	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
1141778	**u Condensador de 0.22 µF**								
	Condensador cerámico VISHAY, 0.22 µF, ±10%, 50 V, X7R.								
		50				50			
							50	0.188	9.40
4903225	**u Resistencia de 100 kΩ**								
	VISHAY BC COMPONENTS METAL FILM RESISTOR, 100 kohm, ± 1%.								
		10				10			
							10	0.051	0.51
8767483	**u Condensador de 10 µF**								
	Condensador electrolítico PANASONIC, 10 µF, 63 V.								
		50				50			
							50	0.0576	2.88
1466399	**u Resistencia de 10 kΩ**								
	VISHAY BC COMPONENTS METAL FILM RESISTOR, 10 kohm, ± 1%								
		50				50			
							50	0.023	1.15
1100407	**u Condensador de 10 nF**								
	Condensador Cerámica Multicapa AVX, 0.01 µF, ± 10%, X7R, 100 V								
		50				50			
							50	0.216	10.8
8767254	**u Condensador de 100 µF**								
	Condensador Electrolítico PANASONIC, 100 µF, ± 20%, 35 V.								
		50				50			
							50	0.0922	4.61

Código	Descripción	u	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
1833325	**u Resistencia de 2,2 kΩ**								
	Resistencia WELWYN, 2K2, 500 mW, 1%								
		10				10			
							10	0.073	0.73
1833334	**u Resistencia de 220 kΩ**								
	Resistencia WELWYN, 220R, 500 mW, 1%								
		10				10			
							10	0.073	0.73
1127964	**u Resistencia 68 kΩ**								
	Resistencia MULTICOMP, 68 kohm, 500 mW, 5%								
		10				10			
							10	0.025	0.25
1833304	**U Resistencia de 1 kΩ**								
	Resistencia WELWYN, 1 k, 500MW, 1%								
		50				50			
							50	0.073	3.65
1101345	**u Zócalo, DIL, 0.3", 8 VIAS**								
	Zócalo DIL de 8 vías para integrado								
		5				5			
							5	0.146	0.73
1101346	**u Zócalo, DIL, 0.3", 14 VIAS**								
	Zócalo DIL de 14 vías para integrado								
		5				5			
							5	0.146	0.73

Código	Descripción	u	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
1101347	**u Zócalo, DIL, 0.3", 16 VIAS**								
	Zócalo DIL de 16 vías para integrado								
		2				2			
							2	0.17	0.34

TOTAL CAPÍTULO 1 - ETAPA µC PIC18F4550..…… 268.28 €

8.3.2. Capítulo 2 - Etapa Arduino

Código	Descripción	U	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
2075382	**u Arduino UNO**								
	Sensor Efecto Hall LEM, LA 55-P, Current Transducer, 50 A, PCB.								
		1				1			
							1	24.29	24.29
1487840	**u Módulo SD para Arduino**								
	Módulo de lectura/escritura en tarjetas SD para Arduino.								
		1				1			
							1	2.23	2.23
1469469	**u Optoacoplador 6N136**								
	OPTOACOPLADOR VISHAY SEMICONDUCTOR , FOTOTRANSIST, 5300 V_{RMS}								
		10				10			
							10	1.37	13.70
1461144	**u Convertidor V/I XTR110**								
	TEXAS INSTRUMENTS XTR110BG IC, CONVERTIDOR V/I, PRECISION								
		2				2			
							2	68.55	137.10

Código	Descripción	U	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
9103546	**u Mosfet P-Channel IRF9510**								
	VISHAY SEMICONDUCTOR Transistor MOSFET, Canal P, 3 A, 100 V, 1.2 ohm.								
		5				5			
							5	1.08	5.40
2309119	**u Potenciómetro 100 k**								
	Potenciómetro ajustable, 23 vueltas, 100 k								
		5				5			
							5	1.13	5.65
1634626	**u Pulsador**								
	MULTICOMP INTERRUPTOR, BLACK PUSH BUTTON, SPDT								
		3				3			
							3	3.57	10.71
9968407	**u Interruptor**								
	Mini interruptor empotrable STD On/Off R81 MDT.								
		1				1			
							1	0.95	0.95
9731148	**u Conector 2 Posiciones**								
	Conector MOLEX, 2 posiciones, 2.54 mm								
		10				10			
							10	0.02	0.20
143126	**u Conector hembra 2 posiciones**								
	Conector hembra MOLEX, 2 posiciones, 2.54 mm								
		10				10			
							10	0.22	2.20

Código	Descripción	U	Long.	Ampl.	Alt.	Parcial	Cant.	Precio Unid.	Importe
9773789	**u Contacto para cables** Contacto para cables MOLEX, 22-30AWG, CRIMP, PK100 (Paquete de 100).								
		1				1			
							1	10.02	10.02

TOTAL CAPÍTULO 2 - ETAPA ARDUINO…….…….. 212.45 €

8.3.3. Capítulo 3-Mano de Obra

Código	Descripción	U	Long.	Ampl.	Alz.	Parcial	Cant.	Precio Unit.	Importe
A025126	**h Ingeniero técnico Industrial – Grupo III** Ingeniero técnico industrial realizador de los convertidores.								
		56				56			
							56	12.50	700
A025327	**h Ingeniero técnico Industrial – Grupo II** Ingeniero técnico industrial supervisor.								
		20				20			
							20	17.50	350

TOTAL CAPÍTULO 3 – MANO DE OBRA……..…1050 €

TOTAL……..2430.21 €

8.4. Resumen del Presupuesto

Capítulo	Resumen		Importe
C_01	Etapa µC PIC18F4550……..		268.28 €
C_02	Etapa Arduino……..		212.45 €
C_03	Mano de obra……..		1050 €
	Total ejecución material……..		1530.73 €
	13% Gastos generales…….	198.99 €	
	6% Beneficio Industrial……	91.84 €	
	Suma de DG i BI ……..		290.83 €
			1821.56 €
	16% IVA.		291.45 €
	Total presupuesto contrata		2113.01 €
	Total presupuesto General		**2113.01** €

El presupuesto general aumenta a la mencionada cantidad de DOS MIL CIENTO TRECE EUROS con UN CÉNTIMO

8. PLIEGO DE CONDICIONES

9.1. Condiciones Administrativas

9.1.1. Condiciones Generales

El presente pliegue de condiciones tiene por objetivo definir futuras investigaciones que continúen el estudio de energías renovables en la plataforma DIP. DPI2006-15627-CO3-03, ya sea trabajando sobre los prototipos construidos u otros que se deriven de ellos.

Los convertidores fabricados son circuitos que están en fase de desarrollo. Estos prototipos se han elaborado para confirmar de forma experimental los estudios teóricos y las simulaciones por ordenador, pero no está preparado para trabajar a nivel industrial. No obstante, se prevé que circuitos derivados tengan una aplicación industrial, adaptando los circuitos a otros aspectos como protecciones, interferencias, etc.

Cada convertidor tiene incorporado su control analógico en la misma placa de potencia.

En caso de modificaciones o mal uso de los circuitos diseñados, el técnico realizador del proyecto no se hace cargo del mal funcionamiento o averías que puedan acontecer tanto en los convertidores como en las fuentes de energía conectadas a ellos.

9.1.2. Normas, Permisos i Certificaciones

Todas las unidades de obra se ejecutarán cumpliendo las prescripciones indicadas en las instalaciones.

Todos los aparatos e instrumentos usados deberán estar homologados. Además, los instrumentos de medida deberán tener a disposición sus correspondientes certificados de calibración.

9.1.3. Descripción General del Montaje

En la elaboración de los prototipos se han definido una serie de pasos a seguir con riguroso orden, donde no es posible empezar uno hasta finalizar el anterior. A continuación se detallarán las actividades a realizar para fabricar los circuitos. Estos pasos son los que se deben seguir para construir las placas diseñadas en el presente proyecto.

1. Comanda y compra del material.
2. Construcción de los inductores.
3. Fabricación de las placas de circuito impreso.
4. Colocación y soldadura de los diferentes elementos sobre la placa.
5. Colocación de los elementos de disipación necesarios.
6. Verificación y ajuste de las placas.
7. Interconexión de los módulos.
8. Verificación y ajuste de los módulos interconectados.
9. Mantenimiento de los equipos.

9.2. Condiciones Económicas

9.2.1. Precios

El importe calculado en el presupuesto del presente proyecto puede sufrir variaciones debidas a cambios en los precios de los componentes utilizados. Estos precios unitarios se entiende que comprenden la ejecución total de un prototipo, incluyendo todos los trabajos complementarios y los materiales así como la parte proporcional de imposición fiscal, las cargas laborales y otros gastos que puedan derivarse.

El presupuesto no incluye los gastos de tipo energético ocasionados por el proceso de instalación ni por el uso del prototipo. Tampoco incluye las obras que sean necesarias, las cuales irían a cargo de la empresa contratante.

9.2.2. Responsabilidades

Los costes que pueda provocar el incumplimiento de las especificaciones expuestas en el presente capítulo, en la manipulación de los circuitos construidos, recae sobre el instalador o el usuario.

El instalador o usuario es el único responsable de todas las acciones en contra de lo acordado, que él o las personas que estén bajo su responsabilidad cometan durante la ejecución de las operaciones relacionadas con las mismas. También es responsable de los accidentes o daños que por errores, inexperiencia o aplicación de métodos inadecuados se produzcan a personas ajenas al desarrollo del prototipo.

El instalador o usuario es el único responsable del incumplimiento de las disposiciones vigentes en materia laboral respecto de su personal y, por tanto, de los accidentes que puedan suceder y los derechos que puedan derivarse de ellos.

En el caso que se implemente la totalidad o una parte del contenido del proyecto para la elaboración de circuitos para usos industriales, la persona responsable de la ejecución (contratista) tendrá la obligación de hacerse cargo de todos los gastos originados por el trabajo mal ejecutado sin que sirva de excusa que el Técnico Director haya examinado y aprobado las obras.

9.2.3. Cláusula del Proyecto

Los estudios y manufacturas realizados en el presente proyecto se han efectuado exclusivamente con finalidades académicas y en ningún caso se podrá sacar beneficio económico sin un acuerdo previo con el Dr. Hugo Valderrama Blavi, director de la investigación.

9.3. Condiciones Facultativas

9.3.1. Personal

Todas las acciones que se desarrollen serán ejecutadas por personal cualificado con conocimientos de electrónica de potencia. También será necesaria experiencia en

software de simulación de circuitos electrónicos, diseño de placas en circuito impreso y uso de aparatos e instrumentos de medida como osciloscopios, multímetros, cargas activas…

El personal se someterá a las normas y reglas previstas por la comunidad autonómica, país u organismos internacionales sobre este tipo de trabajos. El técnico realizador del proyecto, así como el personal investigador no se hace responsable de los desperfectos provocados por su incumplimiento.

El contratista tendrá en la obra, el nombre y clase de operarios que haga falta por el volumen y naturaleza de los trabajos que se realicen, los cuales serán de reconocida aptitud y experimentados en el oficio. El contratista estará obligado a separar de la obra al personal que a juicio del Director Técnico no cumpla con sus obligaciones, realice el trabajo defectuosamente, ya sea por falta de conocimientos o bien por no obrar correctamente.

9.3.2. Reconocimientos i Ensayos Previos

Cuando el Director Técnico lo considere oportuno, podrá encargar el análisis, ensayo o comprobación de los materiales, elementos o instalaciones, ya sea en la fábrica de origen, laboratorios oficiales o en la misma obra, según lo que crea más conveniente, aunque estos no estén indicados en este pliego.

En el caso de discrepancia, los ensayos o pruebas se efectuarán en el laboratorio que el Técnico Director de la obra designe.

Los gastos ocasionados por estas pruebas y comprobaciones irán a cargo del contratista.

Antes de la alimentación de los prototipos serán necesarios unos reconocimientos previos de las placas del circuito impreso, que incluirán: verificación de conexiones y comprobación del buen estado de todos los componentes. Una vez alimentada se comprobará el funcionamiento de todos los componentes y se substituirán los elementos defectuosos, en caso de existir.

9.3.3. Materiales

Todos los materiales cumplirán las especificaciones y tendrán las características indicadas en el proyecto. Además deberán cumplir la calidad indicada y especialmente los elementos de precisión. En el caso de que no se encuentre en el mercado algún producto, ya sea porque se ha acabado o porque ya no se fabrica, el operario encargado del montaje tendrá que estar capacitado para sustituirlo por uno similar.

Cualquier otra especificación o característica de los materiales que figuran en solo uno de los documentos del proyecto, aunque no aparezca en la resta, será igualmente obligatoria.

9.3.3.1. Conductores

Los conductores de unión entre módulos (señal) serán cables de cobre de sección 1.5 mm^2, ya que deben soportar potencias de hasta 150 W. Para evitar pérdidas en los cables, se recomienda disminuir todo lo posible su longitud e incluso se pueden utilizar conductores con una sección superior.

9.3.3.2. Resistencias

Las resistencias son los elementos más usados en los circuitos electrónicos. Tienen unidades de Ohm que es el cociente entre un Voltio y un Amperio.

Una resistencia no es exacta y es necesario establecer los extremos máximos y mínimos entre los cuales está comprendido su valor. La tolerancia marca el intervalo de valores admisible y se expresa normalmente en porcentaje del valor exacto. Para obtener los extremos se tendrá que multiplicar el valor nominal de la resistencia por su tolerancia, después sumar este resultado al valor nominal para saber el máximo que puede obtener o restar para saber el mínimo.

En el proyecto se utilizaran dos tipos de resistencia, de sensado y de uso general. Las resistencias de uso general son resistencias que pueden soportar como máximo una potencia de 0.25 W, normalmente son utilizadas en la parte de control. Las resistencias de sensado son de tipo SMD colocadas en la parte de potencia, normalmente son de valor más pequeño y potencia elevada, para poder controlar el corriente que circula por un determinado punto sin introducir pérdidas en el circuito.

En lo que concierne a las resistencias de uso general, las tolerancias estandarizadas son 5%, 10% y 20%. Según el valor óhmico y la tolerancia, se establecen de forma estándar una serie de valores, de forma que con ella se puedan tener toda una gama de resistencias, estos valores son los que se muestran en la *Tabla V*. El conjunto total de valores de toda la gama se obtiene multiplicando por 0, 1, 10, 10^2, 10^3, 10^4, 10^5, 10^6 y 10^7.

Tabla V
Valores estandarizados y tolerancias de las series E6, E12 y E24

Serie	Tolerancia	Valores estandarizados
E6	20%	1.0, 1.5, 2.2, 3.3, 4.7, 6.8
E12	10%	1.0, 1.2, 1.5, 1.8, 2.2, 2.7, 3.3, 3.9, 4.7, 5.6, 6.8, 8.2
E24	5%	1.0, 1.2, 1.3, 1.5, 1.6, 1.8, 2.0, 2.2, 2.4, 2.7, 3.0, 3.6, 3.9, 4.3, 4.7, 5.1, 5.2, 5.6, 6.8, 7.5, 8.2, 9.8

Para evitar la utilización de un número elevado de seros en la designación del valor de una resistencia se utilizan múltiples del ohm. Los más usados comercialmente son:

- El kilo Ohm (kΩ), 1 kΩ = 10^3 Ω.

- El mega Ohm (MΩ), 1 MΩ=10^6 Ω.

Para identificar el valor de una resistencia se utiliza un código de colores que permite cubrir toda la *Tabla V*. Este sistema consiste en pintar alrededor de la resistencia cuatro anillos de un color determinado. Los dos primeros colores son los que identifican el valor de las resistencias E6, E12 y E24, el tercer color el nombre de ceros que es necesario añadir y el cuarto color, la tolerancia de la resistencia.

9.3.3.3. Condensadores

La capacidad de los condensadores se mide en Faradios (F), pero como la unidad es excesivamente grande, se utilizan, en la práctica, otras unidades fracciones de la anterior. Las más usadas son:

- El micro Faradio (μF) = 10-6 F.
- El nano Faradio (nF) = 10-9 F.
- El pico Faradio (pF) = 10-12 F.

Igual que las resistencias, los condensadores también tienen una tolerancia. Esta acostumbra a ser del 5.10 o 20%. Aunque en los electrolíticos puede llegar a ser del 50%. Existen muchos tipos de condensadores, pero en este proyecto solo se han utilizado de dos tipos, los de película de poliéster y los SMD.

9.3.3.4. Inductores

Los inductores son componentes pasivos formados por un núcleo magnético y un hilo de cobre esmaltado enrollado a su alrededor en forma de espiras, las cuales generan un flujo magnético que mayoritariamente circula por el núcleo.

La magnitud física relacionada es la inductancia, la cual se expresa en Henrios (H), aunque, en la práctica, se suelen utilizar los dos submúltiples siguientes:

- El mili Henrio (mH), 1mH = 10^{-3} H.
- El micro Henrio (μH), 1μH =10^{-6} H.

Los inductores son los componentes con menos exactitud, ya que los ha de fabricar el instalador. Del mismo modo, existen aparatos de medida de inductancias que permiten obtener buenas aproximaciones.

9.3.3.5. Circuitos Integrados y Semiconductores

Los circuitos integrados se tendrán que alimentar adecuadamente teniendo en cuenta las hojas de características (datasheets). Tanto los circuitos integrados como los semiconductores nunca se deberán exponer a valores de tensión y corriente superiores a los indicados en el datasheet. Otros aspectos a tener en cuenta serán los daños que se pueden producir en estos elementos a causa de la electricidad estática. Para reducir las posibilidades de este efecto será necesario el uso de guantes de látex. De esta forma se

evita cualquier descarga indeseada a los circuitos integrados, ya que estos son los más sensibles a este tipo de descargas.

9.3.3.6. Zócalos y Torneados Tipo D.I.L.

Todos los circuitos integrados usados se montaran sobre un zócalo del tipo D.I.L. (Dual In Line) para la unión con la placa de circuito impreso. Se trata de un soporte de contacto mecanizado de gran cantidad de perfil bajo, formado por contactos internos (patas) de estaño sobre una base de cobre-berilio niquelado y con recubrimiento de carbón estañado. Los zócalos están moldeados mediante un poliéster negro con fibra de vidrio. Sus características se muestran en la *Tabla VI*.

Tabla VI
Características físicas de los zócalos tipo D.I.L.

Características	Valores
Margen de temperaturas	55 C a 125 C
Resistencia de contacto	10mΩ (máx.)
Resistencia de aislamiento	1010 Ω
Fuerza de inserción por contacto	120 g
Fuerza de extracción por contacto	80 g
Fuerza de retención por contacto	400 g (min)

El uso de zócalos torneados para la inserción de circuitos integrados reduce el tiempo de substitución por otro circuito integrado y además se evita el calentamiento de las patas de los integrados en el proceso de soldadura, que podría producir su deterioro o, incluso, la destrucción del dispositivo.

9.3.3.7. Placas de Circuito Impreso

Todas las placas de circuito impreso serán construidas a partir de una lámina de estaño fresada. Las placas se fabricarán a doble cara.

9.3.3.8. Interconexión de las Placas de Circuito Impreso

Todas las placas dispondrán de sus conexiones pertinentes, tanto para unirlas a la lámpara en la salida, como a la fuente de alimentación a la entrada, como para unirlas entre ellas. Además de las conexiones de señales de la etapa de control. Cada terminal de conexión tendrá un agujero por el cual se introducirá un conductor o una pata que irá soldada a una pista determinada de la placa.

9.3.4. Condiciones de Ejecución

En este apartado se describirán los procesos a realizar en la fabricación de un prototipo.

9.3.4.1. Encargo y Compra del Material

La compra de los materiales, componentes y aparatos necesarios deberá de realizarse con suficiente antelación de forma que estén disponibles en el momento que se inicie el montaje de las placas de circuito impreso.

9.3.4.2. Construcción de los Inductores

Para la construcción de los inductores se utilizará hilo de cobre esmaltado y soldable de 0.07 mm^2 de sección y se calculará el nombre de hilos necesarios para conseguir la sección deseada. Después se trenzaran los hilos y se enrollarán alrededor de un núcleo de ferrita toroidal hasta llegar al nombre de vueltas necesarias para obtener la inductancia deseada.

Seguidamente, mediante un soldador o un baño de estaño se extrae el esmalte aislante de los extremos del hilo y se estañan las puntas para poder conectarlas a la placa de circuito impreso.

9.3.4.3. Fabricación de las Placas de Circuito Impreso

Con tal de realizar las placas de circuito impreso se ha utilizado un innovador método, que consiste en descargar el archivo software donde esta generado el esquema del circuito correspondiente, en una máquina de control numérico.

Ella sola va fresando una capa de aluminio, delimitando las pistas y realizando los agujeros pertinentes. Este método permite realizar placas más rápidamente y eficientemente.

9.3.4.4. Soldadura de los Componentes

Existen diversos métodos para poner en contacto permanente dos conductores eléctricos, pero la que combina mejor y ofrece una menor resistencia de contacto, sencillez, seguridad y rapidez es la soldadura realizada mediante la fusión de estaño.

El proceso de soldadura consiste en unir dos conductores (hilos o terminales de los componentes) de forma que mediante la adición de un tercer material conductor fundido se cree un compuesto intermetálico entre los tres conductores, de tal manera que al enfriarse y llegar a la temperatura ambiente se obtenga una unión rígida y permanente.

Tanto los materiales a soldar como las herramientas de soldadura han de cumplir unos requisitos previos de limpieza, ya que la presencia de óxidos, grasas o cualquier otro tipo de suciedad impiden que la soldadura sea de la calidad necesaria para que pueda mantenerse sin degradación con el tiempo.

9.3.4.5. Ensayos, Verificaciones i Medidas

Antes de alimentar los módulos se verificará la continuidad de todas las conexiones internas. A continuación se comprobará que todas las tensiones sean las adecuadas para cada módulo.

Se recomienda que se verifiquen las formas de onda obtenidas en los diferentes puntos del circuito mediante un osciloscopio de alta sensibilidad.

El posible funcionamiento inadecuado del equipo puede ser debido a diversas causas que pueden ser resumidas en los dos puntos siguientes:

- Conexiones defectuosas

- Componentes defectuosos

9.3.4.6. Reglamento Electrotécnico de Baja Tensión

Todos los aspectos técnicos de la instalación que directa o indirectamente estén incluidos en el Reglamento Electrotécnico de Baja Tensión, deberán de cumplir lo que dispongan las respectivas normas. Las instrucciones más importantes en la realización del presente proyecto son las siguientes:

1.- M.I.B.T. 029 Instalaciones a pequeñas tensiones.

2.- M.I.B.T. 030 Instalaciones a tensiones especiales.