Universidad Zaragoza
1542

Trabajo Fin de Grado

Plataforma de prácticas sobre microcontrolador para la asignatura de electrónica

Autor

Alberto Gascón Roche

Director

Miguel Ángel Guillomía San Bartolomé

Ponente

Jorge Falcó Boudet

Escuela de Ingeniería y Arquitectura

2017

Plataforma de prácticas sobre microcontrolador para la asignatura de electrónica

Resumen

El uso de microcontroladores está cada vez más extendido y su función permite facilitar muchas tareas cotidianas, como, por ejemplo, a través de la domótica, en la que se usan estos elementos para automatizar las viviendas (sistemas de seguridad, gestión energética, comunicaciones…). Por ello se ha decidido llevar a cabo este trabajo, en el que se realizará una plataforma de prácticas basada en el uso del microcontrolador MSP430G2553.

En este proyecto se ha desarrollado una aplicación que permite, a través de un sensor de temperatura, obtener la tasa respiratoria de una persona, basándose en los cambios de temperatura provocados durante la inspiración y la exhalación. Además, se ha programado una página web que permite mostrar los datos obtenidos a través de ella, utilizando para ello un módulo wifi.

Para llegar hasta este sistema se ha buscado información acerca de los sistemas de medición de la tasa respiratoria actuales, para conocer las diversas alternativas y elegir la más adecuada. Una vez elegida la opción a utilizar se ha procedido a la selección del sensor más apropiado, así como a la fabricación de su correspondiente sistema acondicionador de la señal. Posteriormente, se ha diseñado el programa del microcontrolador para llevar a cabo la medición del período basado en la búsqueda de un máximo y un mínimo. Por último, se ha llevado a cabo la programación del módulo wifi, creándose la página web que se ha usado para mostrar las medidas obtenidas.

Tras haber desarrollado este sistema se ha procedido a su división en tres partes para crear lo que es el objetivo del proyecto, la plataforma de prácticas. Estas prácticas consisten cada una respectivamente en la selección de un sensor y la fabricación de su circuito acondicionador, en la programación del microcontrolador, y en la programación del módulo wifi.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

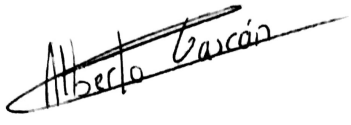
D./Dª. Alberto Gascón Roche,

con nº de DNI 73010971R en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Plataforma de prácticas sobre microcontrolador para la asignatura de electrónica.

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de septiembre de 2017



Fdo: Alberto Gascón Roche

ÍNDICE

1. Introducción

1.1 Objetivos y alcance

El trabajo se centra en el uso del microcontrolador MSP430G2553 de Texas Instruments, llevando a cabo la creación de un sistema de medición de la tasa respiratoria como ejemplo de sistema para captar estímulos externos. El objetivo final del proyecto es realizar una plataforma de prácticas en las que mostrar el funcionamiento de un sistema para captar estímulos externos.

Se hará uso del conversor analógico digital, de subrutinas de interrupciones, de los timers y de la comunicación serie para comunicar el microcontrolador con el PC. Además del microcontrolador, también se hará uso de otros elementos como amplificadores operacionales para acondicionar la señal de entrada al microcontrolador.

La parte final de este proyecto llevará a cabo la realización de una página web que permita obtener los datos de la tasa respiratoria mediante su uso.

Para alcanzar este objetivo se ha estructurado de la siguiente manera:

- ***Documentación sobre medición de tasa respiratoria***: en esta primera parte se busca información sobre los sistemas de medición de la tasa respiratoria y se decide que el método para llevar a cabo dicha medición será uno basado en los cambios de temperatura.
- ***Desarrollo de un sistema inicial***: se comienza la programación del microcontrolador para crear un sistema inicial básico con el que poder comprobar el funcionamiento de los sensores.
- ***Elección de los sensores***: una vez llevadas a cabo una serie de pruebas con los sensores (LPM35, el termopar AN-369 y la termopila ZTP-135SR) se elige el ZTP-135SR por ser el que ofrece mayor precisión.
- ***Mejora del sistema inicial***: elegido el sensor y a partir del sistema inicial se llevan a cabo una serie de modificaciones para conseguir un funcionamiento óptimo del sistema de medición. También es necesaria la creación de un sistema acondicionador de la señal mediante el uso de OrCAD, y tras unas pruebas previas se usa CircuitMaker para hacer el PCB (Printed Circuit Board).
- ***Sistema de comunicación serie***: una vez elaborado el sistema de medición se implementa un sistema de comunicación serie que permita al PC comunicarse con el microcontrolador a través de una página web.
- ***Pruebas de laboratorio***: una vez obtenido el que será el sistema final de medición se llevarán a cabo las pertinentes pruebas para comprobar su correcto funcionamiento.
- ***Diseño de una práctica basada en el trabajo anterior***: tras comprobar el correcto funcionamiento del sistema se procederá al diseño de la práctica cuyo objetivo será la realización de dicho sistema de una forma guiada.

- ***Redacción de la memoria***: una vez realizado todo lo anterior se procederá a la redacción de la memoria en la que se expondrá de forma detallada todo el proceso llevado a cabo.

1.2 Planificación

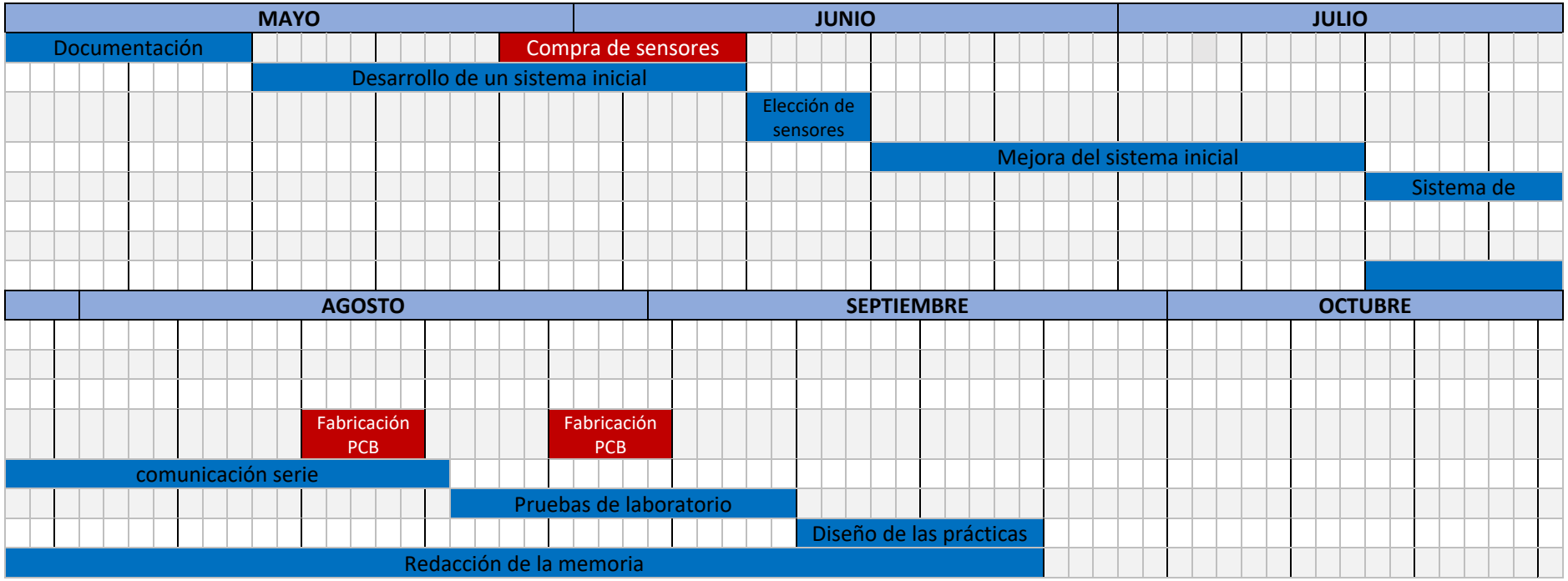


Tabla 1. Diagrama de Gantt

1.3 Contenido de la memoria

Esta memoria en la que se explican todos los pasos seguidos hasta la consecución de la plataforma de prácticas sobre microcontrolador se ha estructurado de la siguiente forma:

1. Introducción: Se describe el objetivo y el alcance del proyecto, la planificación que se va a seguir y se describe brevemente el contenido del documento.

2. Estado del arte: En este apartado se recopila información acerca de la situación actual de los sistemas de medición de la tasa respiratoria y se decide un sistema para la aplicación.

3. Diseño electrónico: Se expone el proceso de selección, diseño y montaje de todos los elementos de hardware del proyecto: sensores, sistema acondicionador y módulo wifi.

4. Diseño del firmware: Se comenta el código programado en el microcontrolador y en el módulo wifi, explicando cada una de sus funciones.

5. Diseño de las prácticas: En este apartado se diferencian tres partes del proyecto realizado y se estructuran en tres prácticas, en las que se aporta material de apoyo y se guía su realización.

   ANEXO 1: En este anexo se muestra el código usado en el microcontrolador, así como las dos alternativas buscando máximos y mínimos.
   ANEXO 2: Se muestra el código del módulo wifi.
   ANEXO 3: Se muestra el esquemático del programa CircuitMaker.

2. Estado del arte

La medición de la tasa respiratoria es un elemento fundamental para el posterior diagnóstico de posibles enfermedades respiratorias (como asma, apneas del sueño, etc) que afectan aproximadamente a un 5% de la población mundial. Este dato incrementa notablemente cuando se habla de personas de la tercera edad de países desarrollados, alcanzando el 30% [1]. Es por ello que los métodos efectivos para la medición de la tasa respiratoria son muy necesarios.

El proceso de medición de la tasa respiratoria se realiza habitualmente por medios observacionales en vez de utilizar sistemas electrónicos para ello. Sin embargo, esto puede causar dificultades en situaciones en las que el paciente no colabore para facilitar el proceso, como por ejemplo cuando un niño está llorando o está inquieto. En estas situaciones un sistema electrónico capaz de determinar la tasa respiratoria al margen de la predisposición a colaborar del paciente sería muy conveniente.

Existen diferentes procesos para la medición de este factor [2] basados en diferentes principios, pero se pueden agrupar en métodos que requieren contacto y métodos que no requieren contacto. El primer grupo comprendería:

- Métodos por detección de movimiento o Plestimografía inductiva respiratoria (PIR): se colocan unas bandas elásticas alrededor del pecho y del abdomen del paciente y se miden los cambios asociados a la respiración (Figura 1). La expansión y deflación del tórax y el abdomen son recogidas por la PIR que, tras ser calibrada, proporciona un valor de volumen [3]. Son precisos en un entorno controlado y pueden detectar asincronías sutiles debidas a trastornos respiratorios.

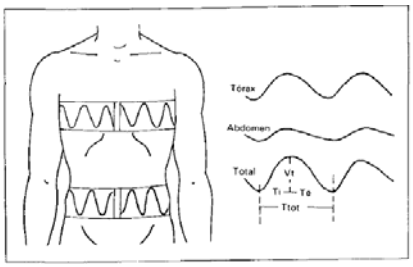


Figura 1. Esquema de las bandas elásticas y de las señales obtenidas tanto en el abdomen y en el tórax como la suma de ambas [3].

- Métodos por medición de flujo de aire: existen diferentes procedimientos dentro de este grupo, siendo los más destacados la medición de cambios de temperatura, la medición del volumen de aire exhalado y la detección del dióxido de carbono liberado (capnometría). Son precisos, pero requieren de entornos controlados y se suelen usar en los postoperatorios.
- Métodos acústicos: estos métodos se basan en analizar las vibraciones para así detectar los flujos de inspiración y expiración. Estos métodos han dado unos buenos resultados

y no se ven afectados si el paciente respira por la boca o por la nariz. Por el contrario, tragar, toser, hablar o incluso un gran ruido de fondo sí que afectaría a estas mediciones.

- Electrocardiogramas: en este procedimiento se colocan una serie de electrodos en el paciente para medir la fluctuación asociada a la respiración para determinar la tasa respiratoria. Estos sistemas ofrecen la posibilidad de monitorizar la tasa respiratoria a distancia y de forma continua.
- Fotoplestimografías: esta metodología está basada en la medición de los niveles máximos de oxígeno en sangre, para así, controlando la variación de los niveles de oxígeno en sangre monitorizar la respiración del paciente.

Entre los métodos que no requieren contacto destacan:

- Sistemas de video: se graba al paciente y se analizan las imágenes para detectar cambios debidos al movimiento mediante luces infrarrojas.
- Medición de los niveles de humedad: se utilizan dispositivos que detectan la humedad asociada al aire exhalado para así medir la tasa respiratoria.
- Ultrasonidos: se utilizan ondas de ultrasonidos que detectan las variaciones del abdomen y del pecho del paciente para calcular a partir de estas la tasa respiratoria [1].

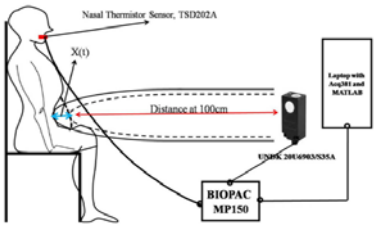


Figura 2. Montaje experimental para medición con ultrasonidos con comprobación mediante termistor [1].

- Aplicaciones móviles: existen algunas aplicaciones que detectan utilizando la cámara del dispositivo las variaciones de la cara del usuario y del pecho asociadas a la respiración.

A pesar de la existencia de numerosos métodos, ninguno se ha incorporado de manera total a las instalaciones médicas pese a presentar una mayor precisión que los métodos actuales.

Con el fin de desarrollar una práctica que utilice uno de estos sistemas para el posterior análisis de los resultados, se ha optado por el método de la medición de la temperatura del aire. Este método aprovecha el cambio de temperatura del aire en la zona de los orificios nasales debido al intercambio de calor producido en los pulmones para obtener la tasa respiratoria [4]. Entre otros factores, se ha elegido este sistema por el bajo coste de los sensores de temperatura, ya que el objetivo de este proyecto es la realización de prácticas, lo que implicaría la necesidad de numerosos sensores.

3. Diseño electrónico

3.1 Diagrama de bloques

En la siguiente imagen se muestra el diagrama de bloques de la aplicación que se lleva a cabo en este proyecto. En este diagrama se puede apreciar como todas las partes del sistema se pueden englobar dentro de dos unidades alimentadas a la misma tensión (5 voltios).

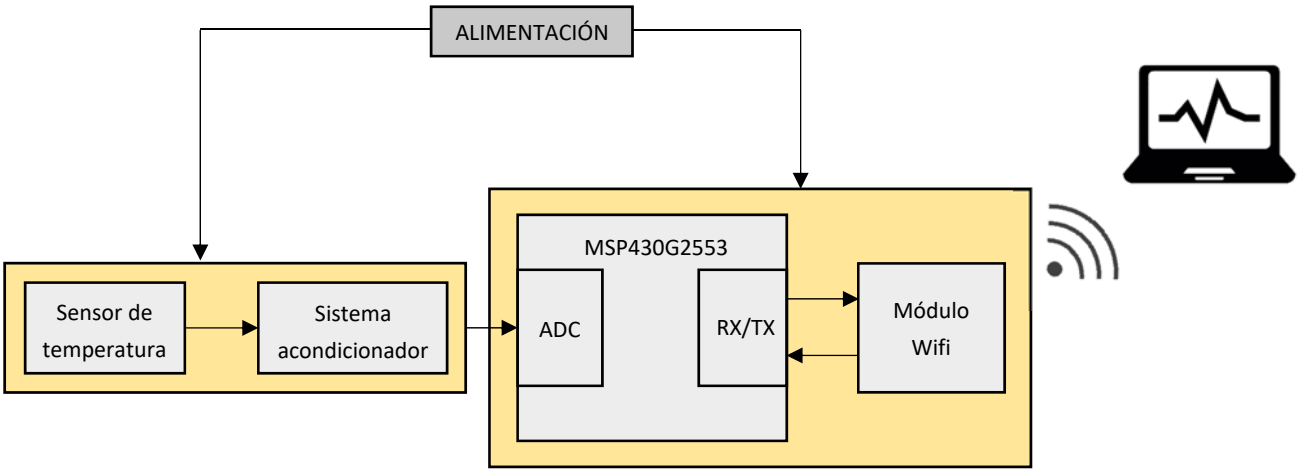


Figura 3. Diagrama de bloques del sistema para medir la tasa respiratoria mediante sensores de temperatura.

Las partes más destacadas serían el sensor de temperatura, el microcontrolador (***MSP430G2553*** [5]) y el módulo wifi; mientras que el sistema acondicionador, aunque necesario, es una consecuencia de la elección del sensor de temperatura.

Para la elección de cada una de las partes del sistema se han llevado a cabo una serie de pruebas para comprobar su correcto funcionamiento. Esto ha sido así salvo en el caso del microcontrolador, que debido a que el objetivo del proyecto es formativo se optó por la utilización de un microcontrolador disponible en el laboratorio y con el que los alumnos estuvieran familiarizados.

3.2 Sensores

Dado que el microcontrolador a usar ya estaba establecido desde un primer momento, la primera decisión en cuanto a qué hardware utilizar se da en el ámbito de los sensores de temperatura.

En un primer momento se optó por la utilización del sensor ***LM35*** [6] debido a su disponibilidad en el laboratorio, lo que evitaría tener que llevar a cabo posteriores compras para la realización de las prácticas. Sin embargo, al comprobar su hoja de características se observó que la respuesta que ofrece dicho sensor ante un cambio de temperatura en aire estático sería de varios minutos (**¡Error! No se encuentra el origen de la referencia.**, **¡Error! No se encuentra el origen de la referencia.**). Fue por este motivo por el que se descartó esta primera opción, ya que para percibir los cambios de temperatura provocados por la respiración se precisa de un sensor con una respuesta mucho más rápida que la obtenida para evitar obtener una señal errónea.

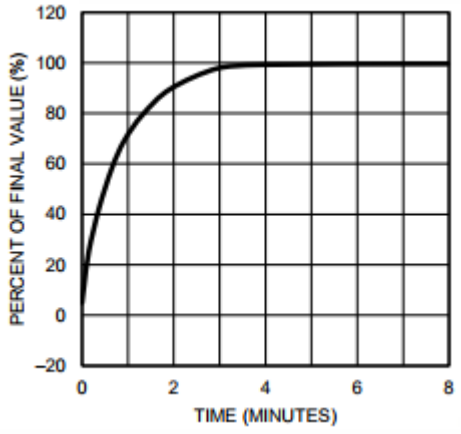


Figura 5. Respuesta térmica en aire estático [6].

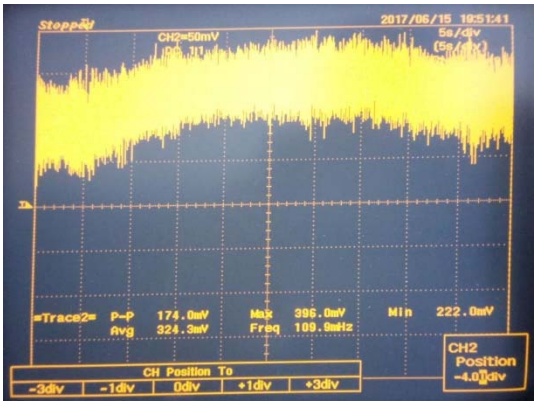


Figura 4. Comprobación en osciloscopio (5s/div y 50mV/div).

Tras comprobar que el sensor LM35 no podía ser utilizado se pasó a comprobar el termopar ***AN-369*** [7]. Al tratarse de un termopar es necesario incorporar un sistema para el acondicionamiento de la señal que se llevó a cabo mediante el ***AD-8494*** [8]. La conexión utilizada se obtuvo de su hoja de características y es la denominada conexión básica [**¡Error! No se encuentra el origen de la referencia.**].

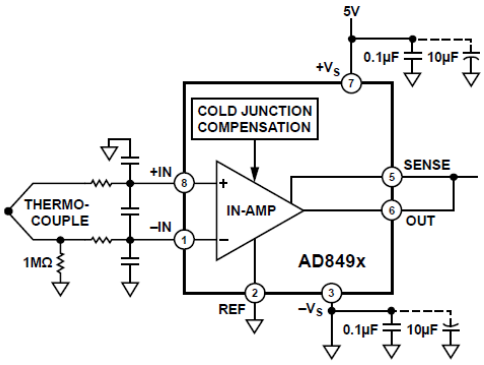


Figura 6. Conexión básica para el AD-849x. [8]

Tras realizar dicha conexión se comprobó el funcionamiento del sistema mediante el uso de un osciloscopio. Con el fin de que la respuesta fuera visible, ya que la señal proporcionada por el termopar es de muy poca magnitud, se dispuso un sistema amplificador restador que aumentara las variaciones de voltaje debidas a cambios de temperatura [Figura 7]. Tras realizar todas estas conexiones se llevó a cabo la comprobación final, donde se pudo observar que, a pesar de obtenerse una respuesta mucho más rápida que la obtenida en el caso anterior, todavía era demasiado lenta para nuestra aplicación. Se puede apreciar en la figura 8 como la forma de la onda se parece más a una onda senoidal que la anterior, pero, al ser la base en la que se observa de cinco segundos por división, se necesitaría algo más de cinco segundos para detectar una variación de temperatura que sucedería en unos tres. Además, estos cambios de temperatura eran de gran magnitud, al producirse apretando el sensor entre los dedos y liberándolo cada tres segundos aproximadamente. Esta magnitud debería verse reflejada en la medición del osciloscopio, pero sin embargo no es tan destacable la diferencia que se puede apreciar, lo que haría al termopar inservible para la detección de cambios de temperatura provocados por la respiración (debido a su menor variación).

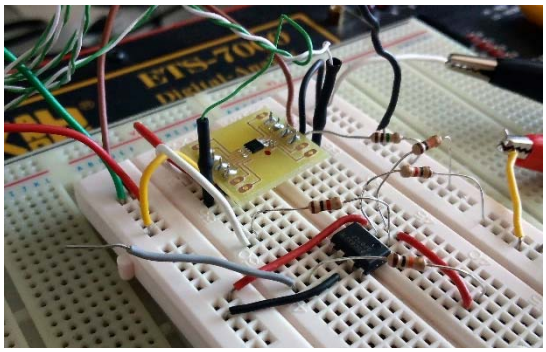


Figura 7. Circuito acondicionador y sistema amplificador de la señal del termopar.

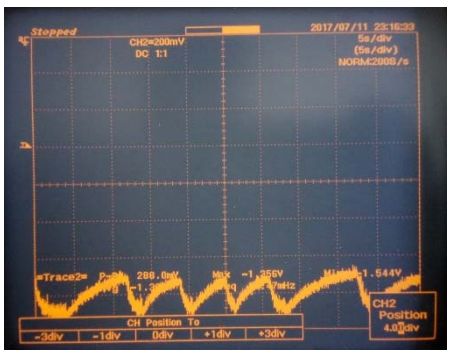


Figura 8. Comprobación en osciloscopio (5s/div y 200mV/div).

Por último, se prueba la termopila ***ZTP-135SR*** [9]. Según su hoja de características la variación de voltaje que ofrece ante cambios de temperatura es muy pequeña, tal y como se aprecia en la figura 9 en la que se ve como para una temperatura de 30ºC (que sería la temperatura media del laboratorio) saca un voltaje de 0.75 mV. Debido a este voltaje de salida tan bajo será necesario acondicionar la señal para obtener una onda en la que se aprecien más claramente los cambios de temperatura. Esto se llevará a cabo mediante un sistema acondicionador del cual se hablará más adelante. Finalmente, tras realizar su comprobación mediante el osciloscopio, se obtuvo el resultado que aparece en la figura 10. En esta imagen se puede comprobar como el sensor utilizado detectó de manera casi instantánea los cambios de temperatura provocados durante el experimento pasando la mano por encima del sensor cada segundo aproximadamente. Así pues, debido a su capacidad para detectar los cambios de temperatura de manera casi inmediata se decidió optar por esta termopila como sensor de temperatura para este proyecto.

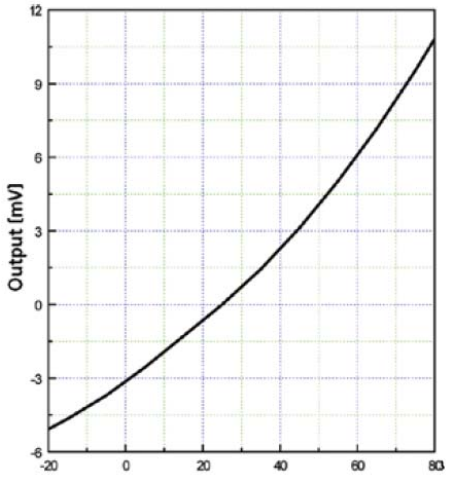


Figura 8. Gráfica de sensibilidad del sensor ZTP-135SR [9].

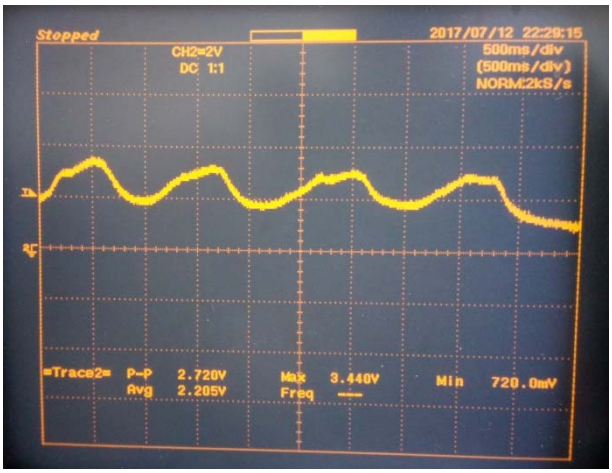


Figura 7. Comprobación en osciloscopio (500ms/div y 2V/div).

3.3 Sistema acondicionador

Como se ha expuesto en el apartado anterior, debido a los bajos valores del voltaje de salida de la termopila se hace necesaria la utilización de un sistema acondicionador de la señal que la adapte a las necesidades de la aplicación.

El sistema acondicionador deberá amplificar mucho una señal muy pequeña (se pasará del orden de milivoltios a voltios) en la que se producirán pequeñas variaciones y se necesitará un comportamiento lineal a lo largo de todo su rango de operación, por lo que será necesario un amplificador operacional rail-to-rail. El amplificador operacional elegido es el ***MCP6002*** [10] por cumplir estos requisitos.

Con el objetivo de conseguir una buena amplificación y evitar que el ruido electromagnético de fondo distorsione la señal, se añadirá un voltio al voltaje de salida de la termopila que será

posteriormente restado mediante una etapa amplificadora restadora. Dicho voltio se ha añadido mediante un divisor de tensión partiendo de una alimentación de cinco voltios ya que se pretende alimentar el circuito desde el ordenador una vez fabricado el PCB.

En un principio se pensó en colocar dos filtros de paso bajo, uno antes de la etapa amplificadora, lo que hace necesario un seguidor de tensión que actúe de acoplador de impedancias, y otro a la salida de la misma. Sin embargo, se observó que el condensador del primer filtro en vez de eliminar el ruido de la señal, hacía que se obtuviera un comportamiento erróneo en el circuito físico, por lo que fue eliminado de este.

Por último, dado que el microcontrolador solo trabaja con tensiones positivas hasta 3,6 voltios, será necesaria una etapa inversora, ya que, como se ve en la figura 11, el voltaje a la salida de la etapa amplificadora es negativo.

La mencionada figura 11 muestra el esquemático del primer sistema acondicionador, diseñado con el programa ***OrCAD*** [11] con el fin de poder realizar simulaciones sobre él. En este diseño se ha llevado a cabo la sustitución de la termopila por un generador de ondas senoidales cuya señal variaría entre los 0.75 y los 2.25 milivoltios, lo que equivaldría a una variación de 20 a 40^{O}C. También se procedió a la eliminación del condensador del primer filtro de paso bajo una vez detectado su mal funcionamiento en el circuito físico debido a la resta de la misma señal desplazada en fase por ese condensador.

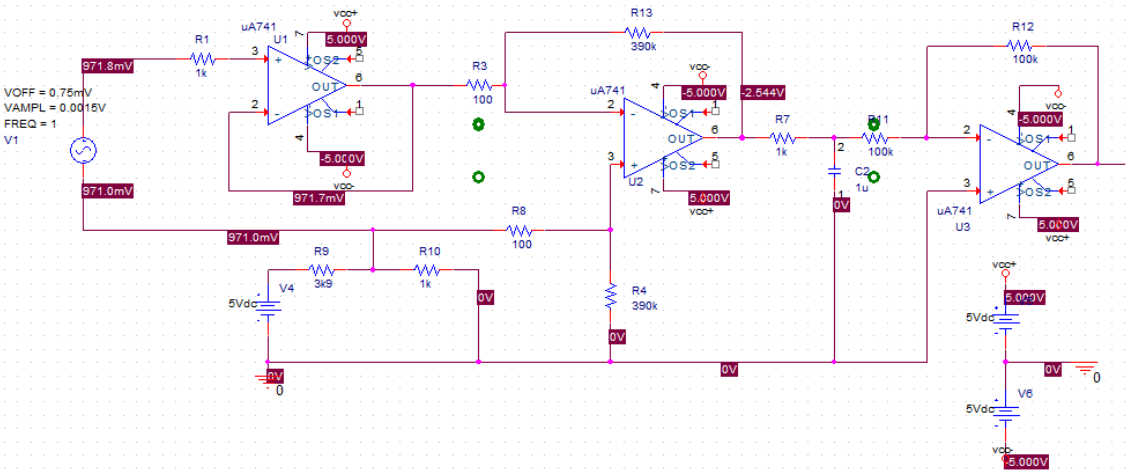


Figura 9. Esquemático del primer sistema acondicionador en OrCAD.

Debido a las limitaciones de este programa, en las simulaciones que se llevaron a cabo se utilizó el circuito de la figura 11, pero eliminando la etapa inversora final con el fin de reducir la extensión del circuito. Dichas simulaciones dieron unos resultados similares a la traza azul de la figura 14, pero se estimó que los cambios de temperatura no llegarían a esos extremos, haciendo válido dicho circuito. Así pues, se procedió al montaje del mismo en una placa de conexiones para poder comprobar el funcionamiento del circuito completo.

En la figura 12 se muestra el circuito anterior realizado en una placa de conexiones para comprobar el funcionamiento del sensor (resultado mostrado en la figura 10). Respecto al esquemático anterior, se ha cambiado la alimentación que proporcionan los cinco voltios por una fuente de tensión que proporciona directamente 1 voltio.

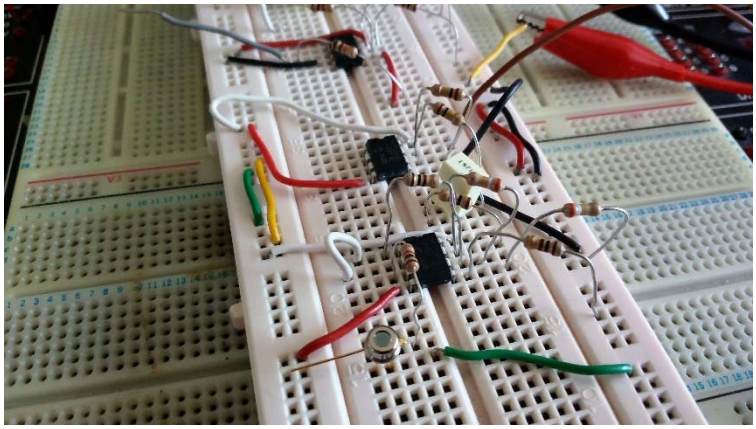


Figura 10. Sistema acondicionador con seguidor, restador e inversor para la señal de la termopila ZTP-135SR.

A pesar de los resultados aparentemente aceptables que se obtuvieron, el sensor tenía comportamientos extraños que se achacaron al ruido electromagnético del laboratorio, que se veía amplificado por el uso de cables. Por ello, con el fin de minimizar dichos efectos, se decidió fabricar una placa de circuito impreso que se mostrará en el siguiente apartado.

Una vez comprobado el funcionamiento del circuito impreso, se decidió ajustar el sistema a través de un software que permitiera trabajar con circuitos de mayor extensión. Tras una consulta a uno de los directores del trabajo, se eligió ***LTspice*** [12], que presenta un uso mucho más intuitivo que el anterior y sin ninguna limitación de tamaño a la hora de realizar las simulaciones.

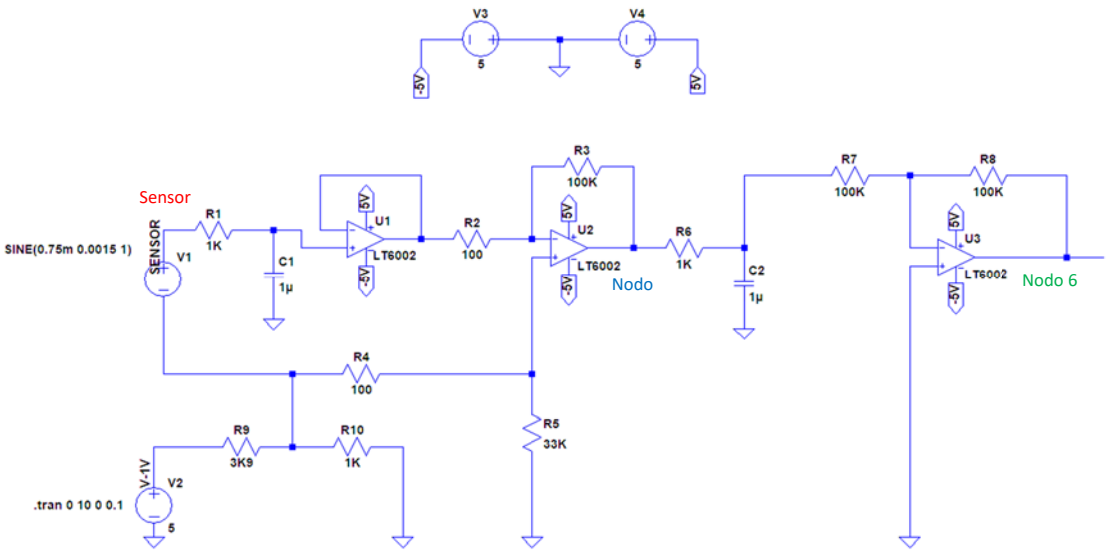


Figura 11. Esquemático del primer sistema acondicionador en LTspice.

Una vez replicado el esquemático anterior y realizada su simulación se observaron una serie de errores que pasaron desapercibidos en el montaje físico debido a que las pruebas realizadas no fueron lo suficientemente completas. El principal fallo fue el hecho de que el voltaje de salida obtenido en la simulación daba unos valores superiores a los 3.6 V e inferiores a los 0 V, lo que hacía que este circuito no fuera válido para nuestra aplicación.

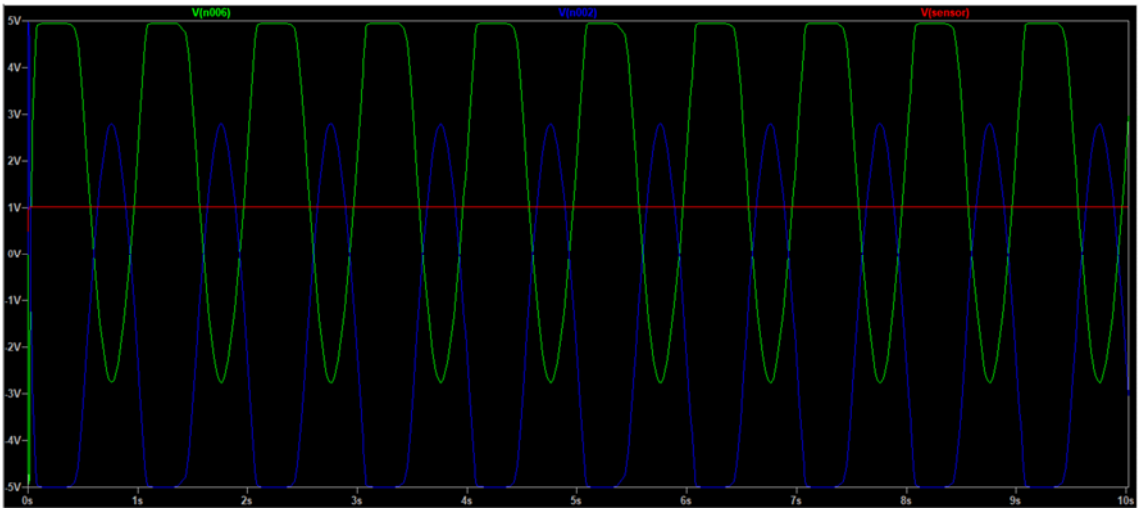


Figura 12. Resultado de la simulación del primer sistema acondicionador.

Una vez identificados los problemas se procedió a la búsqueda de soluciones. La primera alternativa en la que se pensó fue la reducción de la ganancia de la etapa amplificadora, pues disminuyéndola se disminuiría la amplitud de la onda de salida. Una vez llevado a cabo este cambio (se cambiaron los valores de R3 y R5 de 390K a 100K) se comprobó que, aunque la amplitud se había reducido y la onda ya no sobrepasaba los 3.6 V, seguía presentando valores negativos.

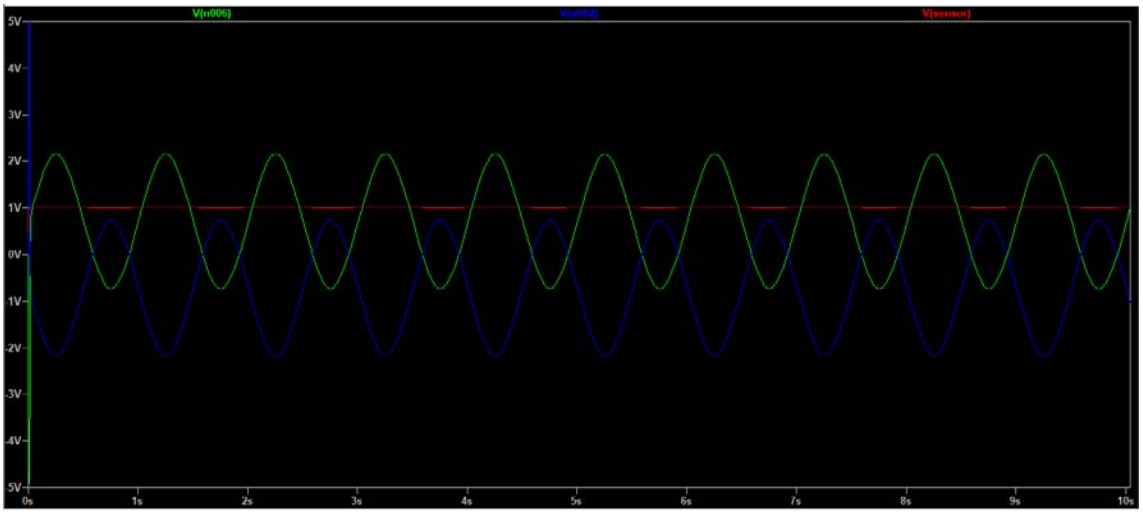


Figura 13. Resultado simulación con cambio de R3 y de R5 de 390K a 100K.

Por ello, se pensó sumarle a la onda un voltaje que hiciera que, conservando la misma amplitud de pico a pico, presentara siempre valores positivos. Con este propósito se cambió la etapa inversora anterior por otra etapa restadora que a la vez que invirtiera la onda le sumara un voltio, obteniéndose una onda como la que se muestra en la siguiente figura:

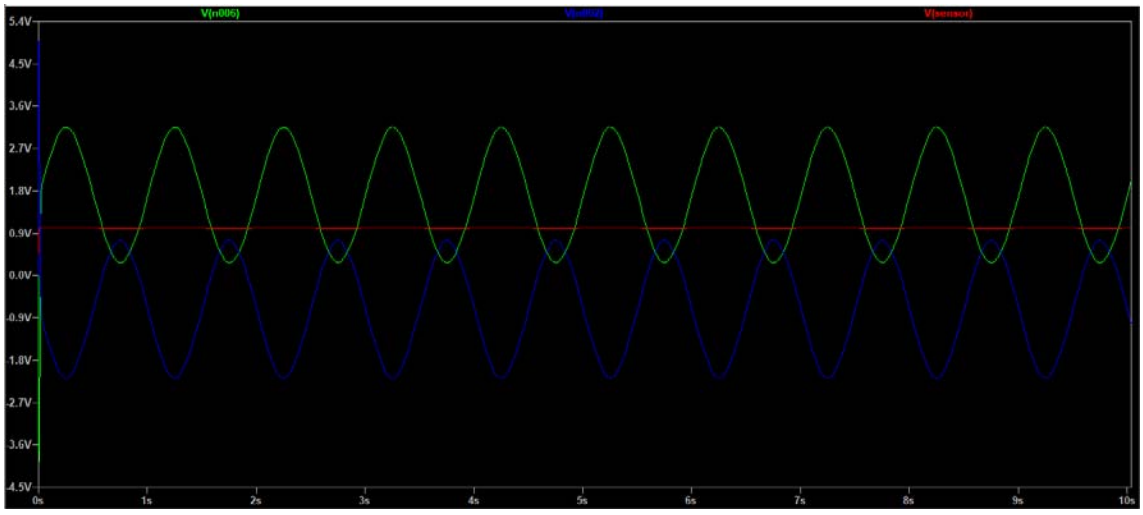


Figura 14. Resultado de la simulación del circuito amplificador con R3 y R5 de 100K y con dos etapas restadoras.

Como se puede ver en la gráfica, los valores de la onda de salida son siempre positivos y no superan los 3.6 V. Se dejó cierto margen para evitar que al montar el circuito físico se obtuvieran valores erróneos. El circuito resultante es el que se muestra en la siguiente figura.

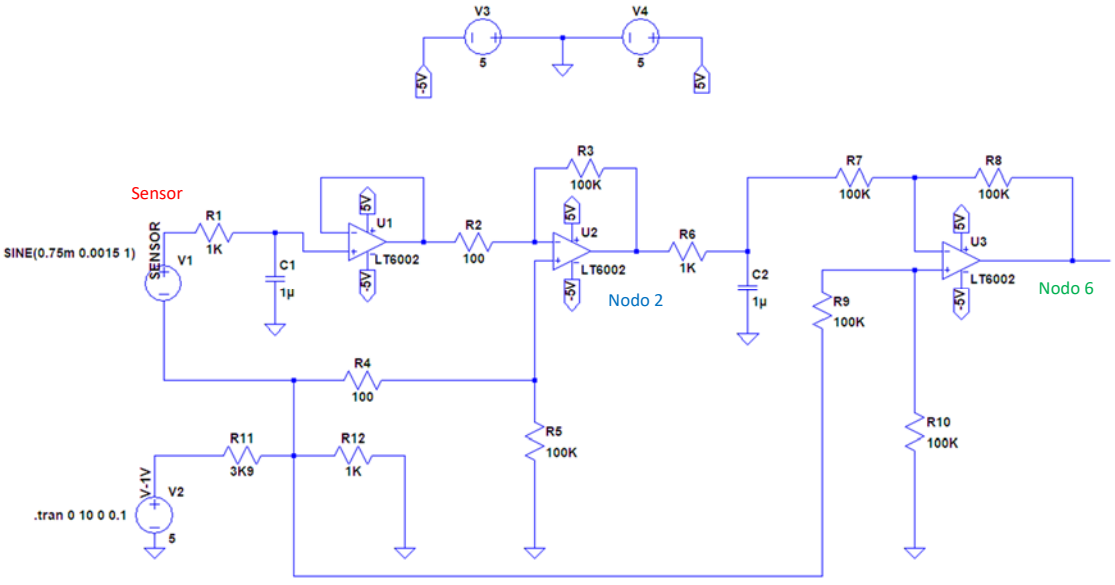


Figura 15. Esquemático del circuito amplificador con R3 y R5 de 100K y con dos etapas restadoras.

Una vez diseñado y simulado el funcionamiento del circuito se procedió a la comprobación del mismo en una placa de conexiones, ya que el comportamiento de las simulaciones es siempre

el ideal y en ellas no se tienen en cuenta factores que se dan en un circuito físico. Así pues, se procedió al montaje del circuito (figura 18) y se comprobó que el funcionamiento obtenido no era el esperado, obteniéndose una señal de salida de un valor medio de -2.714 V como se puede observar en la figura 19.

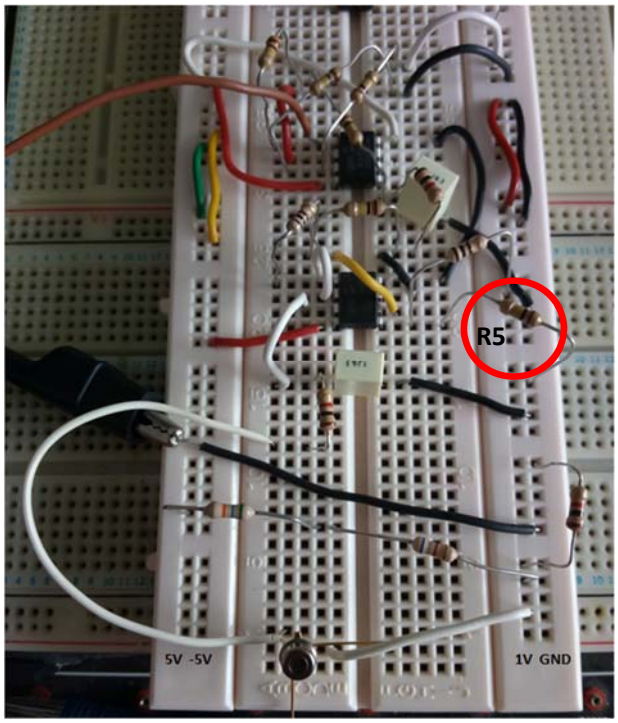


Figura 16. Montaje del circuito amplificador con R3 y R5 de 100K y con dos etapas restadoras en placa de conexiones

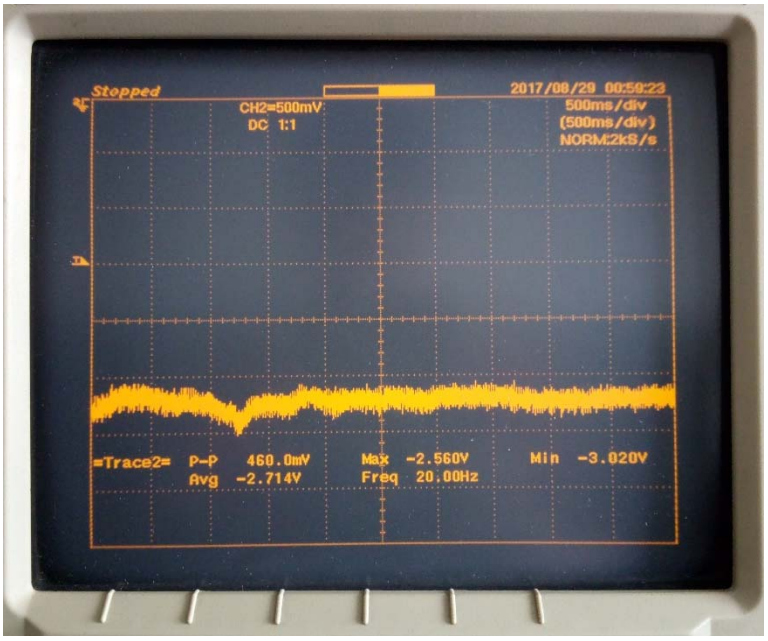


Figura 17. Voltaje de salida en osciloscopio.

Obtenido este resultado se procedió a la búsqueda del problema y se concluyó que el origen de este mal comportamiento estaba en el funcionamiento del seguidor de tensión, que añadía un valor de corriente continua a la señal que salía de él, desajustando las posteriores etapas amplificadoras. Por ello, dado que la tensión que entraba al segundo operacional por la entrada inversora había cambiado, se decidió cambiar la tensión que entraba a dicho operacional por la entrada no inversora. Este proceso se llevó a cabo mediante un potenciómetro, que se colocó en sustitución de la resistencia R5 de 100K. Este potenciómetro tenía un valor máximo de 100K y se comprobó cómo, al reducir su valor, el voltaje de salida iba creciendo hasta el valor esperado para una temperatura ambiente de 25ºC. Posteriormente, con un multímetro se comprobó el que sería el nuevo valor de R5, obteniéndose un valor de 30K. Sin embargo, dado que no existen resistencias comerciales de dicho valor se optó por una resistencia de 33K.

Finalmente, se comprobó el funcionamiento del sistema con la nueva resistencia. Para conseguir que el sensor fuera alterado únicamente por la variación de temperatura provocada por la respiración se respiró a través de una pajilla colocada sobre el sensor. El resultado obtenido se muestra en la siguiente figura:

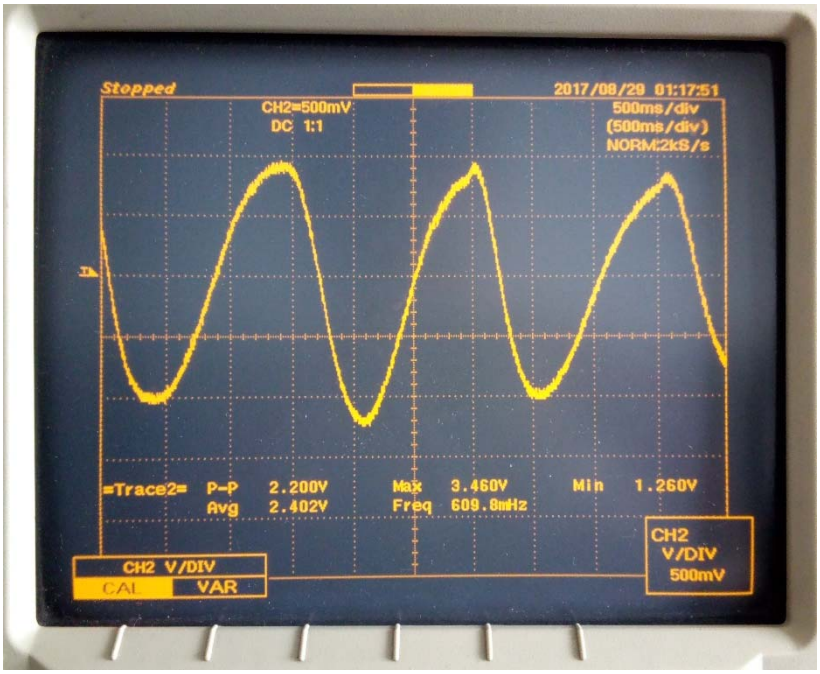


Figura 18. Voltaje de salida del circuito definitivo.

Se puede apreciar como los valores del voltaje de la onda se encuentran en todo momento dentro del rango de valores permitidos por el microcontrolador y como el ruido de la señal ha sido atenuado en su práctica totalidad, siendo el resultado una onda totalmente válida para esta aplicación. Así pues, el esquemático del circuito final será el siguiente:

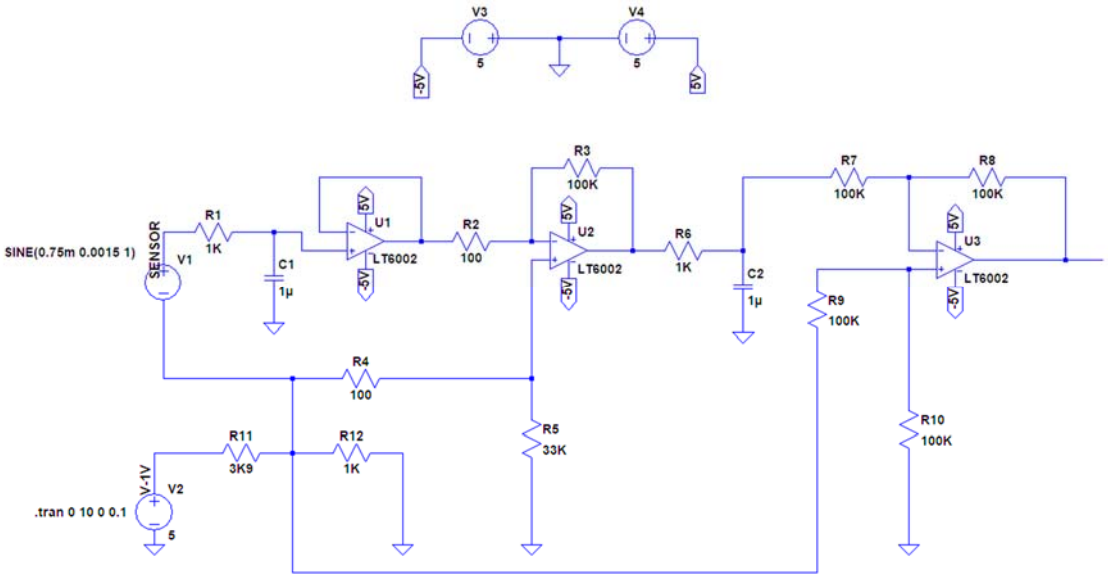


Figura 19. Esquemático del diseño definitivo.

3.4 Diseño de la PCB

Con el fin de evitar distorsiones de la señal en el circuito acondicionador debido a ruido externo, o a señales capacitivas que interfieran en nuestro sistema, se decide llevar a cabo el diseño de una PCB (Printed Circuit Board, placa de circuito impreso). Para esta tarea se utilizó el software libre para diseño de PCB ***CircuitMaker*** [13]. Este programa fue el elegido por varias razones entre las que destacarían:

- La fácil accesibilidad de cualquier persona a este software por ser libre.
- La existencia de gran contenido explicativo en la red que los alumnos podrían usar si fuera necesario.
- La recomendación de dicho programa por parte de los maestros de taller, que serán los que posteriormente fabricarán la placa.

3.4.1 Pruebas previas

Como se ha comentado anteriormente, se fabricó una placa que correspondería con el circuito de la figura 11 para comprobar su funcionamiento minimizando tanto efectos de ruido como capacitivos. Esta primera placa sirvió como toma de contacto con el programa permitiendo la familiarización con el entorno de este y su utilización. Además, se comprobó la pequeña escala en la que se trabaja, a través de la utilización de componentes SMD o de montaje superficial.

Una vez soldados todos los componentes se procedió a la prueba del sistema y, tras un correcto funcionamiento inicial, se obtuvo como resultado el fallo del mismo debido a un cortocircuito en una de las pequeñísimas resistencias SMD. A pesar del fallo de la placa, se pudo comprobar como el sensor alcanzaba las temperaturas elevadas consideradas fuera de rango. Esto hizo que se tomara conciencia de la necesidad de ajustar el voltaje de salida del circuito, y, para posteriores placas, se vio la necesidad de dejar un buen espaciado entre pistas y de utilizar siempre que sea posible componentes de mayor tamaño.

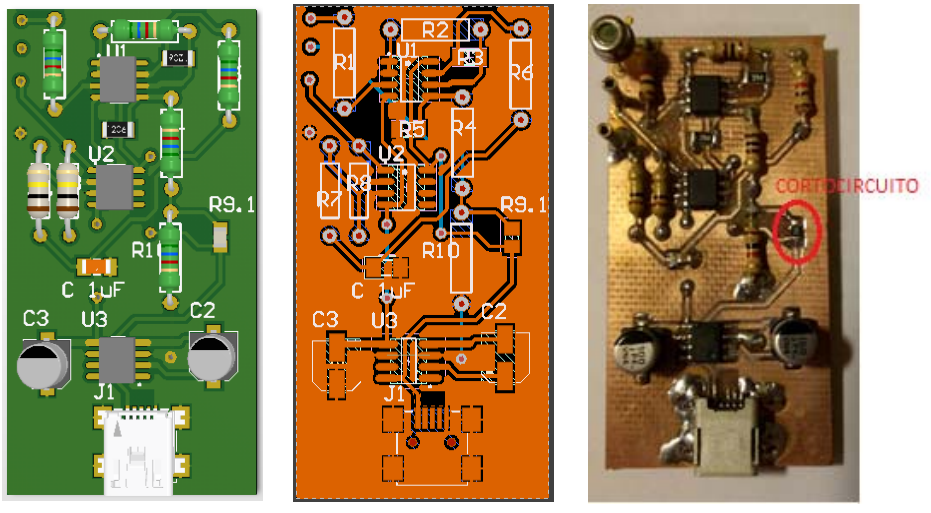


Figura 20. Circuito impreso de la primera prueba.

3.4.2 Diseño final

Una vez llevadas a cabo las pruebas previas, se procedió al diseño del esquemático en dicho programa. El esquemático a diseñar es muy similar al del apartado anterior, pero tiene una serie de diferencias relacionadas principalmente con la alimentación del circuito. Esto se debe a que, en este caso, al tratarse de un circuito impreso, se pensó en reducir al máximo dentro de lo posible todas las alimentaciones externas. Con este objetivo se pensó en utilizar una sola alimentación de 5 voltios para toda la placa, tanto para la alimentación de los operacionales como para obtener el voltio que se añade a la señal del sensor.

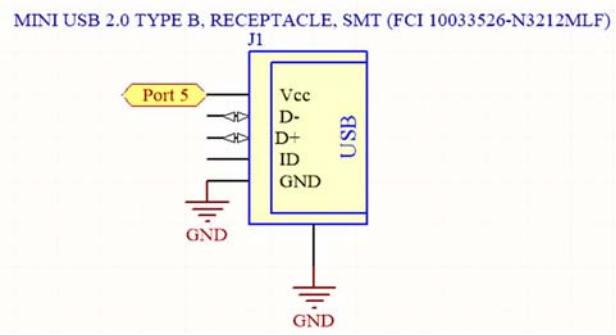


Figura 21. Representación esquemática del puerto mini USB.

Estos 5 voltios se obtendrían de un puerto mini USB que se situaría en la placa como fuente de alimentación y mediante un divisor de tensión, como el que se muestra en el apartado anterior, se obtendría el voltio necesario. Para la obtención de la alimentación negativa hizo falta un mayor esfuerzo, ya que fue necesaria la utilización de un componente adicional como es la bomba de carga. Para ello se buscó en la librería del programa y tras encontrar un componente válido para esta aplicación, ***MAX660*** [14], se procedió a la búsqueda de información con el fin de conocer la conexión que haría que se obtuviera el funcionamiento deseado. Esta conexión se obtuvo de la hoja de características del componente en la que se especifica el montaje para invertir el voltaje suministrado (figura 24).

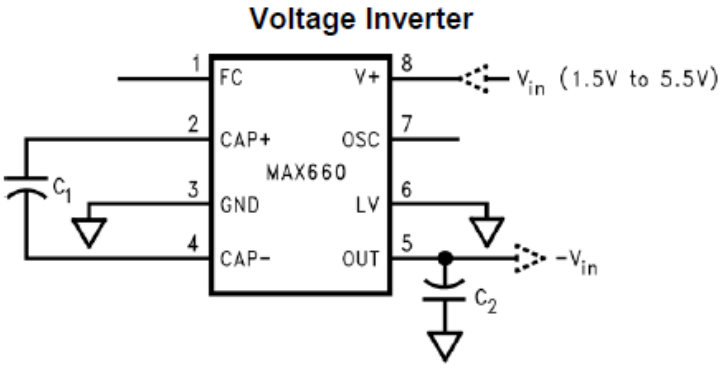


Figura 22. Esquema de conexión del inversor de voltaje [13].

Tras haber solucionado el asunto de la alimentación y el diseño del circuito se procedió a la creación de la placa de circuito impreso. Para este circuito se eligieron mayoritariamente componentes de montaje “through holes” por su disponibilidad en el laboratorio y su mayor facilidad a la hora de soldarlos. El diseño planteado, a diferencia del primer caso, tiene una disposición horizontal en la que los operacionales siguen ocupando una posición central, pero gracias a la orientación horizontal resulta mucho más sencilla la disposición de los diferentes componentes. En cuanto al puerto mini USB se sigue colocando en un extremo, ya que así se facilita el acceso para la conexión con el cable. La bomba de carga se dispone junto al puerto, siendo la zona de conexión del mini USB el lugar donde se tendrán tanto los 5 V como los -5 V, facilitándose así la posterior distribución del voltaje.

A continuación, para evitar que el ruido distorsionara la señal y para evitar hacer pistas innecesarias se procedió a la creación de un plano de tierra en la cara superior. De esta manera toda la cara superior forma un plano que conecta todos los puntos conectados a tierra salvo en las partes en las que hay pistas, que están aisladas del resto.

Puesto que esta placa se va a conectar tanto a un sensor como al microcontrolador se decidió hacer una serie de puntos destinados a la realización de dichas conexiones. Estos serían cuatro puntos, siendo dos de ellos para conectar la tierra de la termopila a 1 voltio y la salida de la termopila al circuito, y los otros dos para conectar la salida del circuito a la entrada del conversor analógico digital del microcontrolador y para conectar las tierras de ambos.

Por último, antes de que los maestros de taller procedieran a la fabricación de la placa, se aumentaron las separaciones entre las pistas y el plano de tierra y se ensancharon las pistas del circuito para evitar posibles cortocircuitos y facilitar la fabricación del circuito. Se aumentó el tamaño de todas las pistas pasando la anchura de 10 milésimas de pulgada a 20 tanto en la cara superior como en la inferior. Este ensanche provocó el aislamiento del tercer pin de la bomba de carga que estaba conectado inicialmente al plano de tierra. Para reconectarlo al plano de tierra se recurrió a la utilización de una vía que conectara este pin con otro punto conectado también a tierra. Así pues, el resultado final del diseño es el que se muestra en la siguiente figura:

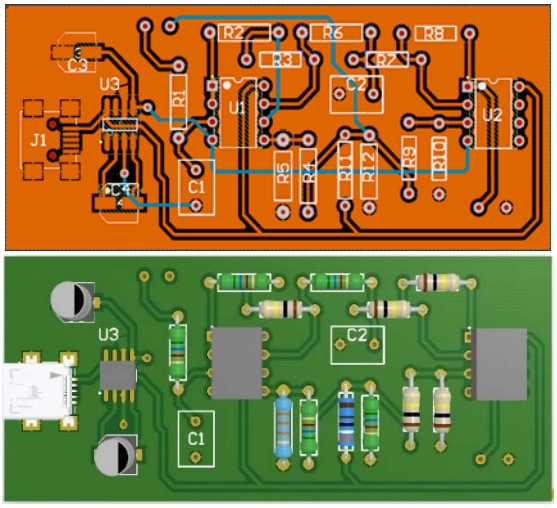


Figura 23. PCB del sistema acondicionador en Circuit Maker.

No obstante, una vez fabricada la PCB se detectó un funcionamiento anómalo de la misma, ya que el voltaje de salida era una señal con mucho ruido. Tras una serie de comprobaciones se detectó el origen de este problema, que consistía en que los voltajes que entraban al primer restador no estaban en fase, ya que, en la entrada inversora, el primer condensador introducía un retraso de 180º respecto a la señal original. Este pequeño retraso hacía que, al entrar una señal en alto y otra en bajo y como la ganancia del operacional es de 1000, las pequeñas diferencias se amplificaran desde milivoltios a voltios, generando el ruido observado. Así pues, se tomó la medida de retirar el condensador del circuito, obteniéndose un resultado mucho más satisfactorio y que se muestra a continuación:

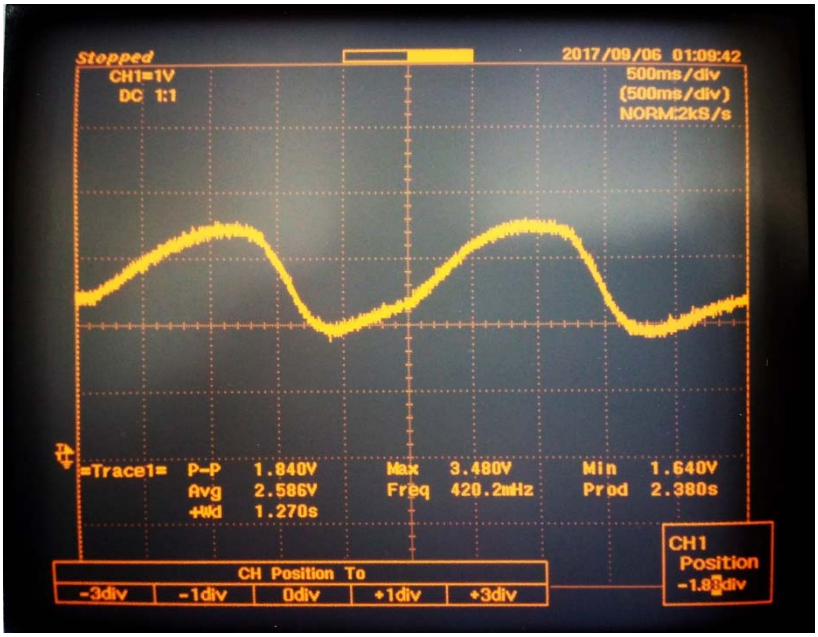


Figura 24. Señal de salida del circuito definitivo.

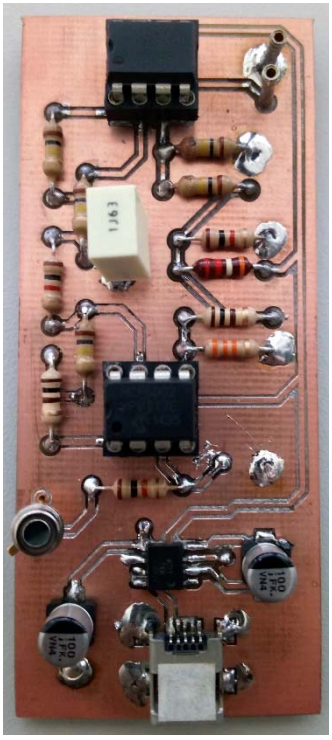


Figura 25. Circuito definitivo en PCB.

A pesar de obtener este buen resultado eliminando el condensador, el circuito volvió a fallar días después. La bomba de carga al invertir el voltaje estaba sacando una señal con mucho ruido, por lo que se decidió colocar en su salida un condensador de 10 milifaradios, eliminando así el ruido existente y recuperando el buen funcionamiento obtenido.

3.5 Módulo Wifi

Para llevar a cabo la comunicación entre el sistema de medición y los clientes se optó por una comunicación wifi, que permita a estos comprobar los resultados obtenidos desde sus propios terminales, ya sean ordenadores, móviles o demás dispositivos electrónicos.

El elemento seleccionado para llevar a cabo esta tarea fue el módulo wifi ***ESP8266*** [15], que puede ser programado mediante la IDE (Entorno de Desarrollo Integrado) de Arduino, facilitando así su programación.

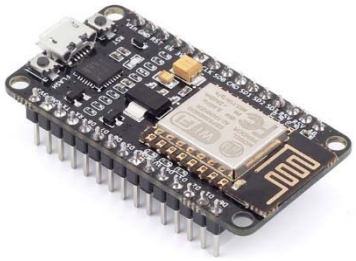


Figura 26. Modulo Wifi ESP8266.

https://aprendiendoarduino.wordpress.com/category/esp8266/

Este módulo consta de un servidor en el que se almacena la página web creada para la obtención de los datos, de modo que, el módulo se conecta a una red wifi y todos los usuarios conectados a esta misma red wifi podrán utilizar la página web para consultar las medidas obtenidas.

La alimentación de este nuevo elemento requiere 3.3V que pueden ser suministrados a través de un cable USB o de una fuente externa. En este caso, dado que el microcontrolador utilizado necesita el mismo voltaje, se alimentará el uno a través del otro evitando así conectar ambas placas mediante USB.

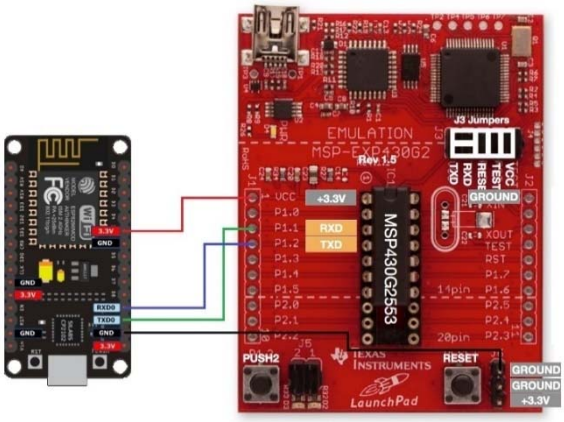


Figura 27. Esquema de conexión de ambas placas.

Estas dos placas también se conectarán mediante sus respectivos pines dedicados al puerto serie, ya que los mensajes que reciba la placa ESP8266 deberán ser trasmitidos al microcontrolador y los datos obtenidos por el microcontrolador deberán ser transmitidos al módulo wifi. Por ello la conexión consistirá en unir el pin RX del ESP8266 con el TX del MSP430G2553 y viceversa.

4. Diseño del firmware

4.1 Programación del MSP430G2553

4.1.1 Diagrama del programa

En la figura 30 se muestra de forma esquemática el funcionamiento del código en el que se basa la aplicación. Se trata de un programa que se mantiene a la espera de una acción externa. Dicha acción podría ser que se pulsara el pulsador P1.3 del microcontrolador o que recibiera una petición de información por el puerto serie. La primera de ellas haría que se iniciara el proceso de obtención de la tasa respiratoria, mientras que la segunda de ellas desencadenaría el proceso de transmisión de los datos obtenidos en el proceso. Cada vez que se finaliza uno de los ciclos mencionados se devuelve el programa a su estado de reposo inicial. El siguiente diagrama muestra de forma gráfica el procedimiento descrito:

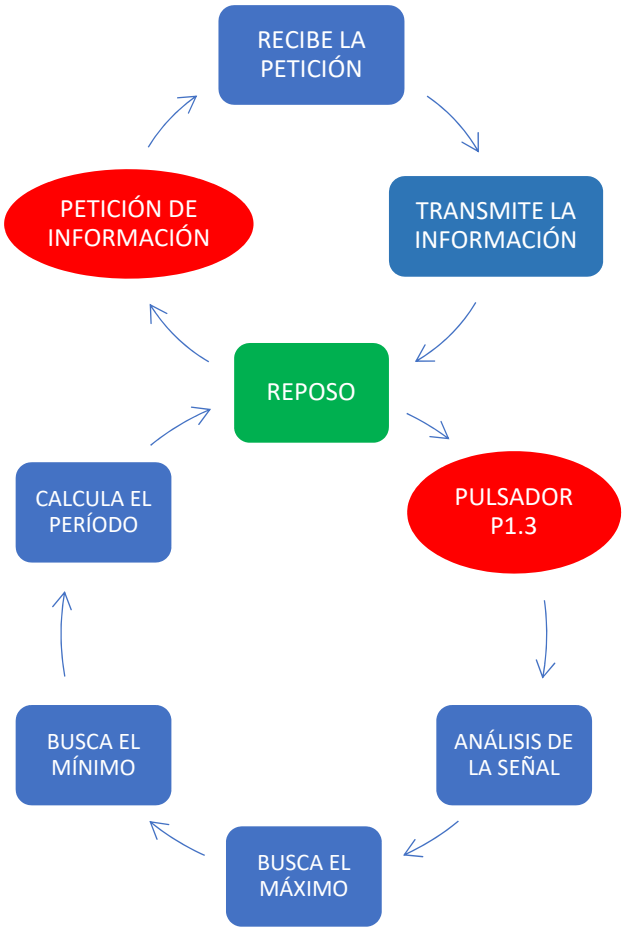


Figura 28. Esquema de funcionamiento del código del microcontrolador.

A continuación, se explicarán cada uno de los estados mostrados en este diagrama.

4.1.2 Reposo

El estado inicial del programa consiste en un estado de reposo. Esto consiste en que el programa se encuentra en un bucle en el que se comprueba constantemente el valor del pin asociado al pulsador del microcontrolador, el pin 1.3. Adicionalmente, se ha programado una subrutina de interrupciones que hará que al recibir una petición de información por el puerto serie se lleve a cabo la transmisión de la información obtenida en la función principal del programa.

Antes de entrar a este estado se ha procedido a la inicialización de los relojes, de los pines, del conversor analógico digital y de la comunicación serie (UART), y se han habilitado las interrupciones generales, ya que sin ellas no se podría llevar a cabo la comunicación serie.

Los relojes se configuran con la idea de que se va a muestrear una señal de una frecuencia muy baja, menores de 1 Hz. Así pues, será necesario poder establecer unos tiempos entre muestras del orden de décimas de segundo, que en comparación con la máxima capacidad de muestreo del microcontrolador (16MHz), es algo mínimo. Por ello, se selecciona la menor frecuencia posible para los relojes, tanto para el MCLK como para el SMCLK, que será de 1MHz. Sin embargo, como esta frecuencia sigue siendo muy alta todavía, dentro de la función principal, se decide aplicarle un factor de división, dividiéndola entre 8 para obtener una frecuencia de final de 125KHz. Para el resto del programa la frecuencia de reloj deberá ser mayor, pues a una frecuencia tan baja el sistema no podría leer los mensajes enviados mediante la comunicación serie.

La configuración de los pines es muy sencilla, ya que fundamentalmente se va a utilizar un pin para introducir la señal proveniente del sensor (pin 1.7) y los dos pines de la comunicación serie UART (pines 1.1 y 1.2). Aunque estos son imprescindibles para el desarrollo del programa, también se configuran otros pines con propósitos de comprobación y activación. Este es el caso del ya mencionado pin 1.3, que está asociado al pulsador, siendo el que inicie la función de obtención del período, y de los pines asociados al led rojo y al led verde (pines 1.0 y 1.6), que durante el proceso de creación del código se han ido utilizando a modo de comprobantes. No obstante, en el código final se ha creado un código mediante la utilización de los leds para determinar el estado en el que se encuentra el sistema (figura 31). De este modo, cuando ambos leds estén iluminados se estará llevando a cabo la obtención del período, si solo el verde está encendido indicará que existen datos que se pueden enviar y finalmente, si solo el rojo está encendido indicará que se están enviando los datos.

Figura 29. Código de colores usado.

El conversor analógico digital, es uno de los elementos fundamentales para esta aplicación, ya que sin él no se podría trabajar con la señal introducida. Para su configuración es importante establecer los límites de voltaje en los que va a trabajar que en este caso son 3.5 y 0 voltios, el reloj con el que se va a asociar (el MCLK en este caso) y el pin al que se debe asociar, que será por el que se introducirá la señal analógica correspondiente, el pin 1.7.

Para la configuración de la comunicación serie, se deben seleccionar tanto la fuente de reloj como la tasa de baudios. Para la primera se elige como fuente el SMCLK, ya que de las posibles elecciones (ACLK Y SMCLK) es la única que se ha configurado previamente. En el caso de la elección de la tasa de baudios se recurre a una tabla (figura 32) que aparece en la guía del usuario. Para esta aplicación se eligieron 9600 baudios por ser la tasa a la que trabaja el módulo wifi utilizado. En esta imagen se especifica los valores que se deben asignar a los registros correspondientes para obtener una determinada tasa de baudios según la frecuencia utilizada.

USCI Operation: UART Mode

Table 15–4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK frequency [Hz]	Baud Rate [Baud]	UCBRx	UCBRSx	UCBRFx	Max. TX Error [%]		Max. RX Error [%]	
32,768	1200	27	2	0	−2.8	1.4	−5.9	2.0
32,768	2400	13	6	0	−4.8	6.0	−9.7	8.3
32,768	4800	6	7	0	−12.1	5.7	−13.4	19.0
32,768	9600	3	3	0	−21.1	15.2	−44.3	21.3
1,048,576	9600	109	2	0	−0.2	0.7	−1.0	0.8
1,048,576	19200	54	5	0	−1.1	1.0	−1.5	2.5
1,048,576	38400	27	2	0	−2.8	1.4	−5.9	2.0
1,048,576	56000	18	6	0	−3.9	1.1	−4.6	5.7
1,048,576	115200	9	1	0	−1.1	10.7	−11.5	11.3
1,048,576	128000	8	1	0	−8.9	7.5	−13.8	14.8
1,048,576	256000	4	1	0	−2.3	25.4	−13.4	38.8
1,000,000	9600	104	1	0	−0.5	0.6	−0.9	1.2
1,000,000	19200	52	0	0	−1.8	0	−2.6	0.9
1,000,000	38400	26	0	0	−1.8	0	−3.6	1.8
1,000,000	56000	17	7	0	−4.8	0.8	−8.0	3.2
1,000,000	115200	8	6	0	−7.8	6.4	−9.7	16.1
1,000,000	128000	7	7	0	−10.4	6.4	−18.0	11.6
1,000,000	256000	3	7	0	−29.6	0	−43.6	5.2

Figura 30. Valores de los registros para la configuración de la tasa de baudios [28].

4.1.3 Análisis de la señal

El análisis de la señal podría considerarse la primera parte de la aplicación en sí, ya que hasta ahora simplemente se han llevado a cabo tareas de inicialización.

Esta parte del código realiza la labor de anticipar entre qué valores digitales máximo y mínimo se encuentran los valores analógicos introducidos por la señal generada por el sensor de temperatura. Aunque pueda parecer innecesaria, esta comprobación previa permitirá conseguir una medida del período mucho más precisa que en el caso de no llevarse a cabo. Esto se debe a que el verdadero objetivo de esta parte del código consiste en establecer dos valores límite, a partir de los cuales el muestreo para la búsqueda del máximo y el mínimo se realizará con una frecuencia muy superior a la utilizada para el resto de la onda.

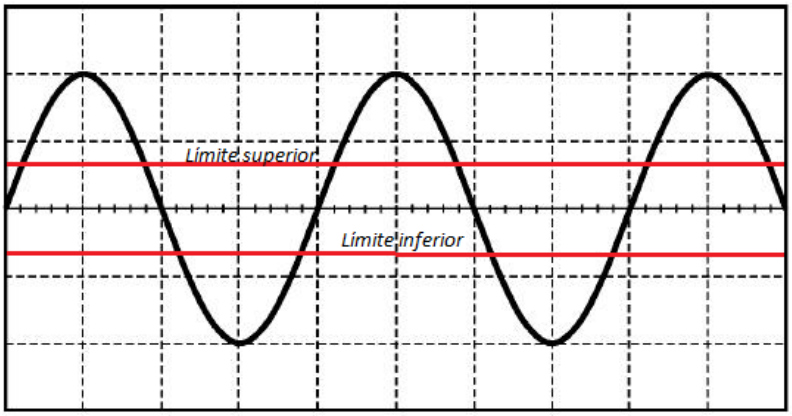


Figura 31. Onda digital recibida con los límites superior e inferior.

Como se puede ver en la figura 33, las líneas rojas indicarían los valores a partir de los cuales se empezaría a muestrear mucho más rápido para obtener una mayor precisión tanto en la determinación del máximo y del mínimo, como en el cálculo del período. Estos límites se han establecido a un tercio y a dos tercios de la amplitud de pico a pico a partir del mínimo valor obtenido en el análisis de la señal.

Podría parecer que la parte intermedia es muy pequeña en comparación con los extremos, pues es del mismo tamaño y en una se va a usar una precisión muy baja en comparación con la otra. Sin embargo, esto tiene una razón de ser. Si las partes exteriores estuvieran más próximas a los extremos podría darse el caso de que al muestrear en la zona intermedia se saltara el máximo o mínimo que se quiere hallar. Así pues, haciendo que tengan una extensión igual, se consigue que los muestreos de la zona intermedia, aunque se adentren en las zonas exteriores, nunca superen los puntos buscados, evitando falsear los resultados.

Además de la mayor precisión obtenida en los resultados, otro propósito de estas diferentes velocidades de muestreo es el de evitar utilizar una frecuencia muy elevada para el muestreo de toda la onda. Este objetivo tendría un carácter formativo, ya que, debido a las bajas frecuencias de las ondas respiratorias, no supondría un gran esfuerzo para el microcontrolador. No obstante,

para otras aplicaciones más exigentes, el muestrear siempre a la máxima frecuencia posible podría suponer un inconveniente para el microcontrolador debido a la alta carga de trabajo que involucraría.

4.1.4 Búsqueda del máximo y del mínimo

Tras establecer los límites superiores e inferiores se lleva a cabo la parte central del programa, la búsqueda del máximo y del mínimo. En este caso para el posterior cálculo del período se ha recurrido a la búsqueda de un máximo y un mínimo, aunque se podría haber hecho igualmente buscando dos máximos o dos mínimos. Lo cual, dado el carácter formativo de este proyecto, podría ser sugerido como un ejercicio para adquirir soltura con el microcontrolador y su programación [ANEXO 1].

La búsqueda del máximo (mínimo) se lleva a cabo comparando dos valores muestreados de forma contigua. Así pues, mientras el valor obtenido en segundo lugar sea mayor (menor) que el obtenido en primer lugar se seguirá buscando.

El primer paso que se lleva a cabo cosiste en comprobar que la señal que se está recibiendo, que será una onda senoidal, está creciendo, ya que el primer valor que queremos obtener será el máximo. En caso de que no esté creciendo se esperará hasta que lo haga para así poder empezar la búsqueda siempre en un mismo estado.

Una vez que la onda ya está creciendo se comienza la búsqueda del máximo. Se lleva a cabo a una frecuencia de 10 Hz en la zona central, mientras que una vez sobrepasado cualquiera de los dos límites establecidos la frecuencia aumenta hasta 125 Hz con el fin de alcanzar una mayor precisión. Una excepción es el caso de la zona exterior superior una vez encontrado el máximo, ya que, aunque se esté sobre el límite superior, se muestrea a 10 Hz, pues sería innecesario utilizar una mayor precisión.

Un elemento que podría falsear las medidas obtenidas es el ruido, ya que las señales recibidas contienen cierto componente de ruido. Para minimizar su efecto se ha recurrido a un proceso sencillo, pero efectivo, que consiste en hacer la media de una serie de valores para reducir la importancia de aquellos que de no ser compensados darían un resultado erróneo. Para ello se toman 10 valores con una frecuencia de 100 o de 1250 Hz, según se encuentren en la zona central o en los extremos, y posteriormente se hace la media de estos. Así pues, aunque el valor del máximo y del mínimo no serán totalmente correctos, sí que aportarán una mayor precisión para la posterior obtención del período.

Valores obtenidos del análisis de una señal senoidal de 0,33 Hz con una frecuencia de 8 Hz										MEDIA
30	36	29	33	33	25	27	28	36	26	30,3
25	40	38	33	33	28	32	34	34	37	33,4
37	34	41	36	42	40	42	38	42	42	39,4
37	38	44	44	43	44	44	49	47	49	43,9
47	45	44	47	49	49	51	49	52	54	48,7
55	52	52	49	52	55	51	49	59	55	52,9
56	60	54	55	55	57	51	60	54	54	55,6
53	51	54	57	53	54	54	48	52	53	52,9
52	45	51	49	53	43	47	50	49	40	47,9
40	39	39	40	45	41	44	42	44	40	41,4
35	41	41	30	37	35	41	38	35	35	36,8
41	32	32	36	35	31	34	28	28	30	32,7
30	31	33	27	33	18	27	25	28	33	28,5
23	21	24	24	27	29	22	18	17	15	22
15	20	18	16	18	17	21	10	16	19	17
17	14	12	9	12	12	16	13	9	14	12,8
17	13	8	7	17	10	15	10	10	14	12,1
6	11	9	4	10	7	4	0	1	3	5,5
0	0	4	7	5	0	3	5	4	3	3,1
5	7	13	9	10	12	8	6	15	9	9,4

Tabla 2. Ejemplo de eliminación de ruido de una señal senoidal.

En la tabla 2 se muestra el ejemplo de una onda senoidal que ha sido muestreada cada 0,125 segundos obteniéndose valores muy dispares, y que al graficarlos tendrían el aspecto que se observa en la figura 34. Se puede apreciar cómo, si no se llevara a cabo una compensación, se detectaría que la onda crece o decrece en puntos en los que no lo está haciendo.

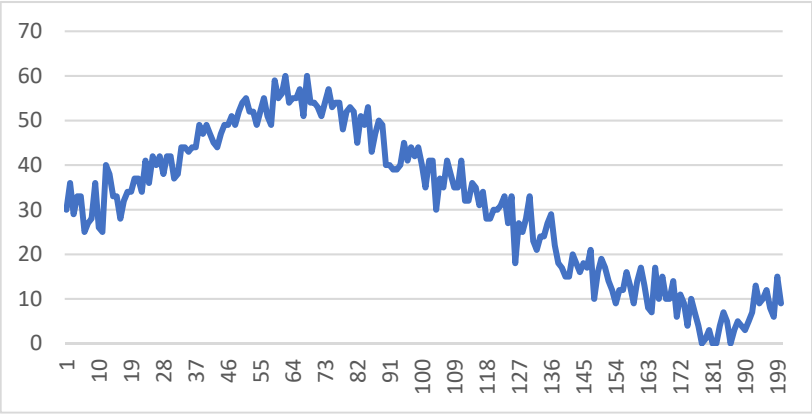


Figura 32. Señal obtenida sin compensación.

Sin embargo, gracias a la compensación llevada a cabo, la señal que percibe el microcontrolador tiene la forma que se ve en la figura 35, mucho más uniforme y suavizada. Facilitándose así el trabajar con ella a pesar del ligero error que provoca en los valores de esta.

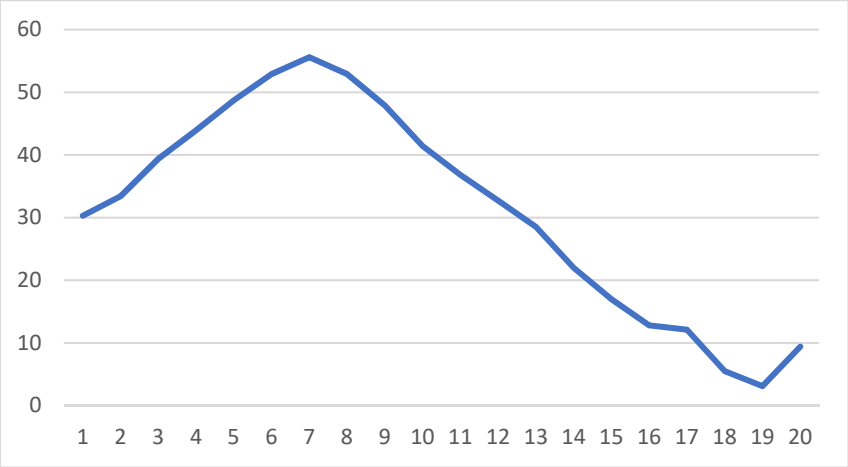


Figura 33. Señal con compensada con media de 10 valores.

4.1.5 Cálculo del período

El objetivo de esta aplicación es conocer la tasa respiratoria de una persona, y para ello es fundamental conocer el período de la onda producida por los cambios de temperatura. El cálculo del período se ha llevado a cabo a través del uso del timer y una subrutina de interrupciones y consiste en lo siguiente:

1. Dado que nos interesa conocer el período de la onda, para calcularlo se deberá conocer el tiempo desde un punto conocido hasta otro punto conocido. En este caso, como se ha comentado anteriormente, se han elegido como dichos puntos un máximo y un mínimo, obteniéndose el semiperiodo de la onda. Así pues, se guardarán los valores del timer cuando se detecte el máximo y cuando se detecte el mínimo.

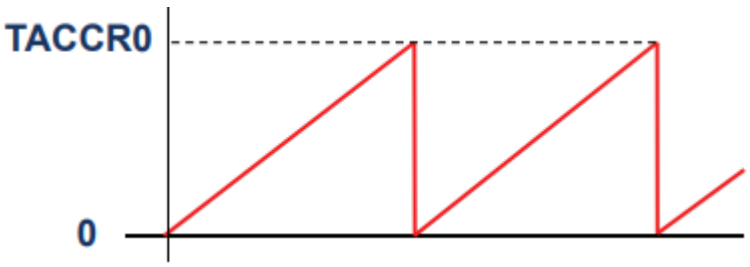


Figura 34. Timer en modo up.

2. Aunque podría bastar con lo expuesto hasta ahora para obtener el período de una gran cantidad de ondas respiratorias (ya que el timer tarda 4 segundos en alcanzar su valor

máximo (TACCR0) y, como se muestra en la figura 37, una persona adulta tarda de media en realizar una respiración unos 5 segundos), es necesario cubrir todas las posibilidades. Con este propósito, se incorpora una subrutina de interrupciones cuyo objetivo es el de contar el número de veces que el timer llega a su valor máximo, añadiendo una unidad a un contador cada vez que se dé esta circunstancia. Esta variable que actúa como contador tiene un valor inicial de -1, ya que en el caso de que el timer solo llegue una vez al máximo, en el cálculo del período no se no se debe sumar ningún valor adicional.

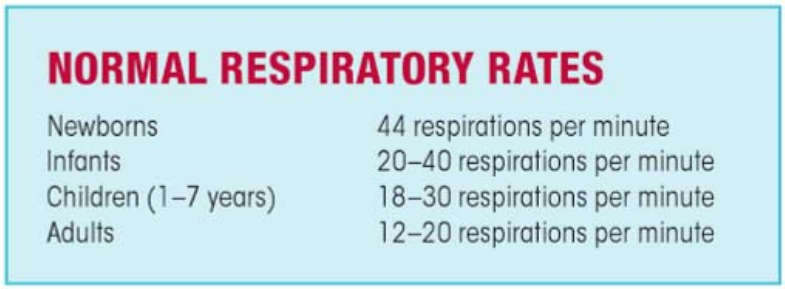


Figura 35. Tasas respiratorias normales según la edad

[Wilburta Q. Lindh; Marilyn Pooler; Carol Tamparo; Barbara M. Dahl (9 March 2009). Delmar's Comprehensive Medical Assisting: Administrative and Clinical Competencies. Cengage Learning. p. 573. ISBN 978-1-4354-1914-8.].

3. Finalmente, una vez conocidos los valores del timer en el máximo y en el mínimo y el número de veces que el timer ha alcanzado su valor máximo, es posible llevar a cabo el cálculo del período. Se diferencian dos casos para este cálculo, ya que el procedimiento no será el mismo si el timer ha llegado a su máximo en algún momento o no. En caso de que no haya llegado al máximo en ningún momento la ecuación a utilizar será la que se muestra a continuación:

$$PERÍODO = (T_{Mín} - T_{Máx}) * 2 * m$$

Se calcula la diferencia entre el valor del timer en el mínimo y en el máximo. Este valor es multiplicado posteriormente por dos, debido a que se trata del semiperiodo, y se multiplica por la pendiente del timer con el fin de pasar de unidades del timer a segundos. Dicha variable "m" tendría el valor de 4/TACCR0.

En el caso de que el timer haya llegado a su máximo en alguna ocasión la ecuación que se utilizará será la siguiente:

$$PERÍODO = (T_{Mín} + TACCR0 - T_{Máx} + v * TACCR0) * 2 * m$$

Así pues, con esta ecuación se tendría en cuenta el hecho de que los valores del TA0R obtenidos en cada caso no se encuentran en el mismo ciclo del timer. Por ello ya no se procede a obtener su diferencia, sino que se suman la distancia desde Tmax hasta TACCR0, la distancia desde 0 a Tmin y el valor debido al número de veces que se ha reiniciado el timer.

4.1.6 Comunicación serie

Para la comprobación del correcto funcionamiento de la comunicación serie se ha utilizado el programa ***CoolTerm*** [16], que permite comprobar todo lo que se envía y recibe a través del puerto serie.

Para iniciar la comunicación serie será necesaria una petición de información por parte de algún usuario, que el microcontrolador recibirá en forma de mensaje a través del puerto serie. Estos mensajes serán analizados siempre y cuando haya datos que poder enviar, y constan de diferentes partes (Tabla 3) que aportan una información al receptor tanto sobre el propio mensaje como sobre la información que pide.

Cabecera				Longitud	Tipo	Origen	Destino	Checksum		Cola	
								Check-H	**Check-L**		
128	1	2	3	3	90	1	2	0	93	165	165
0x80	0x01	0x02	0x03	0x03	0x5A	0x01	0x02	0x00	0x5D	0xA5	0xA5

Tabla 2. Estructura del mensaje de petición de información.

Cabecera: es la primera parte del mensaje y en este caso consta de cuatro bits. Dado que el microcontrolador recibirá muchas señales erróneas debidas a ruido, será necesario indicarle que la señal que está recibiendo no es ruido sino un mensaje que debe leer. Para ello los mensajes que reciba empezarán siempre con la combinación de valores que aparece en la tabla en hexadecimal.

Longitud: una vez leída la cabecera se verá la longitud del mensaje, que indica el número de bytes que comprenden desde la longitud hasta el checksum, ambos no incluidos. Conociendo este dato podremos saber hasta dónde será necesario leer para obtener toda la información del mensaje. En el caso de este mensaje el valor será 3 ya que solo hay tres bytes, que corresponderían con el tipo, el origen y el destino.

Tipo: este campo indica al microcontrolador lo que debe hacer, es decir, indica la finalidad u objetivo de este mensaje. En este caso se está realizando la petición de una serie de datos concretos mediante el valor 90. En caso de que se quisiera llevar a cabo otra acción o una petición de datos diferentes se debería cambiar el valor de dicho campo.

Origen y destino: estos dos son los siguientes campos del mensaje e indican quién ha sido el emisor del mensaje y quién ha sido el receptor. En este caso estos campos siempre tendrán esos valores, ya que solo hay un periférico.

Checksum: este elemento sirve como comprobación de que el mensaje enviado se ha recibido correctamente. Esta comprobación consiste en sumar los bytes comprendidos entre la longitud y el propio checksum, ambos no incluidos. Consta de dos bytes, esto se debe a que en algunas ocasiones el checksum puede superar el valor de 255, el máximo para un byte. En esta aplicación no se dará esta situación, aunque podría darse de cambiar el valor elegido para algunos campos.

Cola: por último, la cola se encuentra en la parte final del mensaje. Su función consiste en indicar que el mensaje ha terminado. Se ha elegido una cola formada por los caracteres ASCII ¥ ¥ en hexadecimal.

Una vez comprobada la petición anterior durante la recepción, se procederá al envío de los datos relativos al máximo, al mínimo y al período al usuario. El procedimiento de envío es muy similar al llevado a cabo en la anterior petición de información, ya que se hará mediante un mensaje que contendrá las siguientes partes:

Cabecera				Longitud	Tipo	Origen	Destino	Período		Mín.	
128	1	2	3	6	90	2	1	XX	XX	XX	XX
0x80	0x01	0x02	0x03	0x06	0x5A	0x02	0x01	0xXX	0xXX	0xXX	0xXX

Máx.		Checksum		Cola	
		Check-H	**Check-L**		
XX	XX	XX	XX	165	165
0xXX	0xXX	0xXX	0xXX	0xA5	0xA5

Tabla 4. Estructura del mensaje de envío de la información.

Como se puede apreciar, la estructura es muy similar a la del mensaje anterior. Las diferencias más notables son los tres apartados en los que se proporcionan las medidas. Sin embargo, aunque los apartados sean los mismos que en el mensaje recibido, hay algunos cambios en su contenido debido a que se trata a un mensaje diferente. Algunos de estos cambios son:

- La longitud del mensaje ha cambiado ya que se han incorporado tres apartados más que requieren dos bytes cada uno debido a su tamaño, pasando a tener un valor de nueve.
- El origen y el destino se han invertido respecto al caso anterior, pues este mensaje se está enviando desde la aplicación al usuario, es decir, al revés que el mensaje de petición de información.
- El checksum, a diferencia del caso anterior, no tendrá siempre el mismo valor, debido a que los valores de las medidas enviadas no serán siempre los mismos. Como no tendrá siempre el mismo valor conocido, podrá tener un valor de más de 255, se han utilizado al igual que en el caso anterior dos bytes para transmitir su valor, siendo necesaria su utilización en algún momento.

Otro elemento a destacar de este mensaje es el envío del período. Como se trata de un número decimal se ha decidido cambiar a una unidad menor y enviarlo en dos bytes como se ha comentado anteriormente. La unidad elegida ha sido la centésima de segundo, ya que simplifica el proceso y asegura que el valor obtenido siempre se podrá representar en dos bytes por ser menor de 255. Así pues, un período de 5.75 segundos pasaría a ser 575 (0x023F en hexadecimal) centésimas de segundo, un valor más sencillo de almacenar, ya que solo ocuparía dos bytes en vez de cuatro y se enviaría como un byte de 0x02 y otro de 0x3F.

4.2 Programación del ESP8266.

4.2.1 Diagrama del programa

La programación del módulo wifi, al realizarse en ***Arduino*** [17], tiene dos partes muy diferenciadas, la primera parte se dedica a la inicialización del módulo, mientras que la segunda parte es un bucle que se encarga de pedir las medidas obtenidas por el microcontrolador y de mostrarlas por pantalla en una página web.

Sus estructuras pueden comprobarse en la siguiente figura:

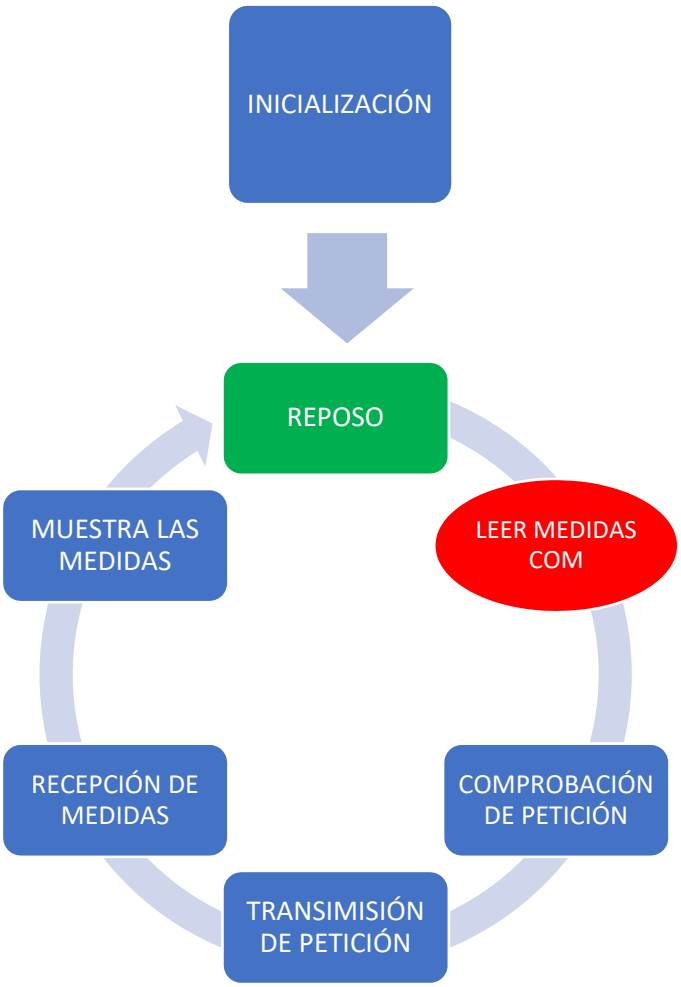


Figura 36. Esquema de funcionamiento del código del módulo wifi.

A continuación, se explicará cada una de las partes de este diagrama.

4.2.2 Inicialización

Dado que este componente se programa en Arduino, la extensa inicialización llevada a cabo en el microcontrolador se verá bastante reducida, ya que, de los parámetros configurados en este, tan solo será necesario configurar la tasa de baudios a utilizar y los leds de la placa. Sin embargo, a diferencia del caso anterior, será necesario inicializar el servidor del que consta esta placa.

Para ello, se deberá proporcionar al módulo el nombre y la contraseña de la red wifi a utilizar (TFG-Alberto y ESP-8266), así como los valores para la configuración de una IP estática. Tras estos pasos se inicia una espera de la que se saldrá al conectarse a la red wifi que se ha especificado con anterioridad. Finalmente, una vez conectado a la red wifi se arrancará el servidor web y se proporcionará la IP de conexión en la que se podrán ver y pedir las medidas.

En caso de que el módulo no consiguiera conectarse a la red especificada, se mandará un mensaje de error por el puerto serie y se reseteará la placa, reiniciándose el proceso de conexión. La red que se utilizará será proporcionada mediante un móvil, haciendo esta aplicación operativa en cualquier lugar.

4.2.3 Bucle

Esta segunda parte se trata de un bucle infinito que comprueba todo el rato el puerto serie y del que solo se puede salir reseteando la placa como en el caso anterior. Como se ha comentado previamente, este bucle se dedica a transmitir una petición de información de un usuario al microcontrolador y de devolverle los datos requeridos mostrándolos por pantalla.

El bucle comienza comprobando si hay algún usuario conectado. Una vez conectado alguien espera hasta que este haga alguna petición a través de los botones de la página web que ha creado, en la que se pueden pedir las medidas de la tasa respiratoria o encender y apagar un led (esta segunda parte se incluyó a modo de comprobación de que se ha conectado bien la placa).

La página web en la que se muestran los resultados y a través de la cual se realizan las peticiones de información es una página muy sencilla que consiste en dos contenedores, uno para el control de la comunicación serie, en el que se muestran las medidas analógicas, y otro para el control de las salidas digitales (el diodo). Se han utilizado varias librerías (Jquery y Bootstrap) para darle una apariencia más atractiva y para hacer que se ajuste a cualquier dispositivo en el que se visualice, tanto móviles como ordenadores o tablets (figura 39).

Figura 37. Página web creada para obtener las medidas de la tasa respiratoria.

En el caso de que se pulse el botón para pedir las medidas de la tasa respiratoria, el módulo wifi iniciará un proceso de comprobación del mensaje recibido por el puerto serie, que posteriormente deberá transmitir al microcontrolador. Esta comprobación será muy similar a la que se realiza posteriormente en el microcontrolador, ya que verifica que el mensaje conste de la cabecera predeterminada, que vaya dirigido a ese equipo y que el checksum sea correcto. Así pues, una vez comprobado el mensaje recibido lo transmite a la siguiente etapa y espera una contestación. Dicha contestación será el mensaje que transmitía el microcontrolador y en el que se incluían las medidas de la tasa respiratoria. Finalmente, tras comprobar de nuevo el mensaje recibido, se muestran los valores por pantalla.

Por el contrario, si se pulsaran los botones de encender o apagar el led no sería necesaria ninguna comprobación, ya que el led sobre el que se está actuando está en el propio módulo wifi, ejecutándose la acción requerida de forma inmediata.

Con la intención de poder comprobar el estado en el que se encuentra el programa, se ha programado de forma que vaya mostrando por el puerto serie diferentes mensajes en las diferentes situaciones, como, por ejemplo: cuando espera para conectarse a la red wifi, cuando ya se ha conectado, cuando se enciende o apaga el led o cuando se piden las medidas. Estos mensajes se pueden comprobar por el monitor serie que tiene la IDE de Arduino (figura 40).

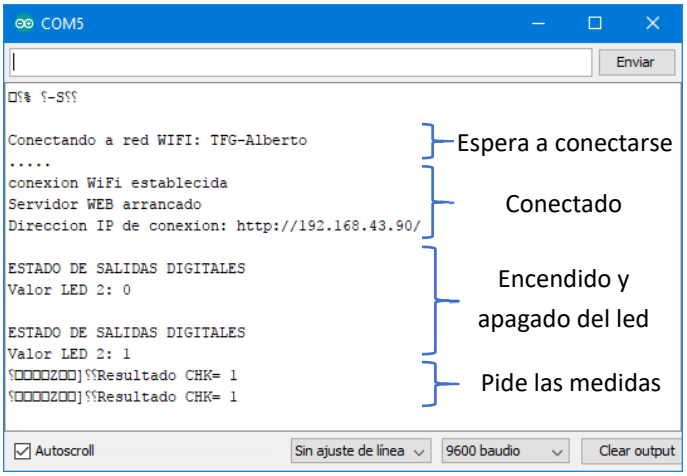


Figura 38. Monitor serie de Arduino con los diferentes mensajes enviados por el programa.

Para comprobar los mensajes que entran por el puerto serie también se ha utilizado en este caso el programa ***Breakout***, ya que este, a diferencia del programa usado anteriormente (CoolTerm) y del monitor serie de Arduino, es capaz de mostrar directamente lo que pasa por el puerto serie. Así pues, con el programa Breakout se ha podido comprobar las entradas y salidas del puerto serie.

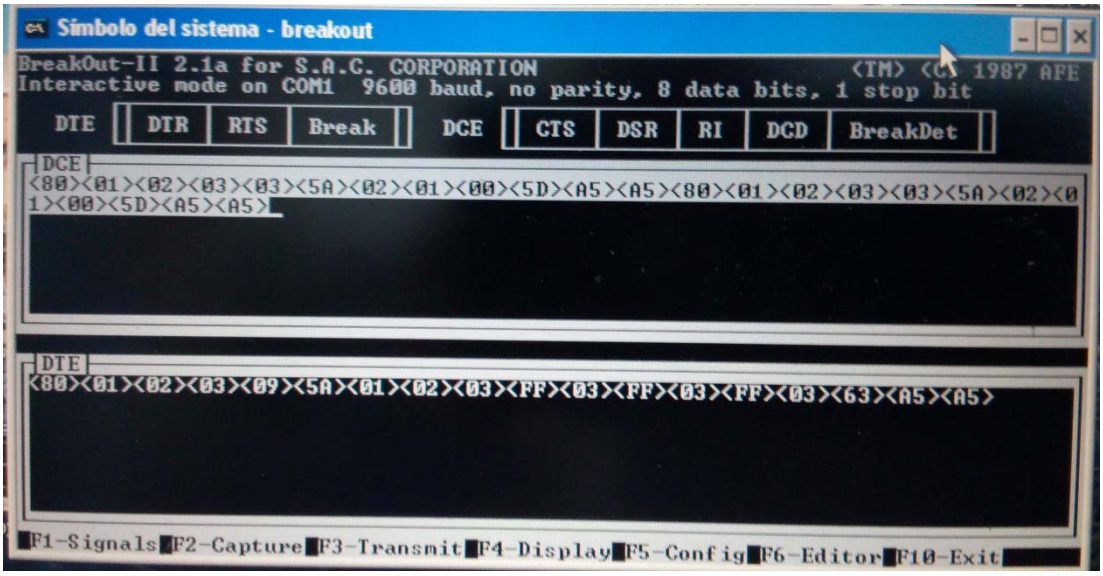


Figura 39. Programa Breakout mostrando mensajes enviados y recibidos por el puerto serie.

Sin embargo, para la utilización de este programa fue necesario un montaje para transmitir la señal al ordenador donde se ejecutaba el programa. En dicho montaje se utilizaba el operacional ***ADM3202*** [18] y un cable RS-232 y tenía la siguiente forma:

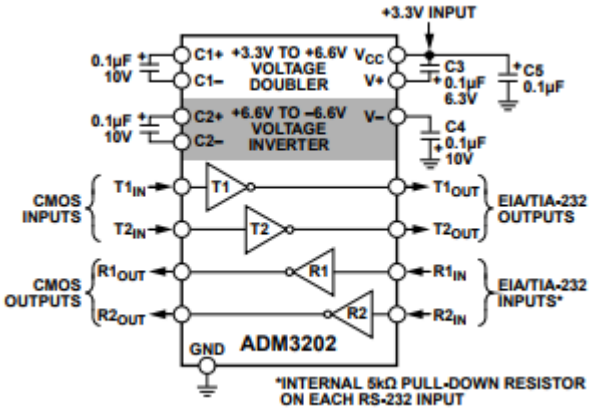


Figura 41. Esquema de montaje del operacional ADM3202

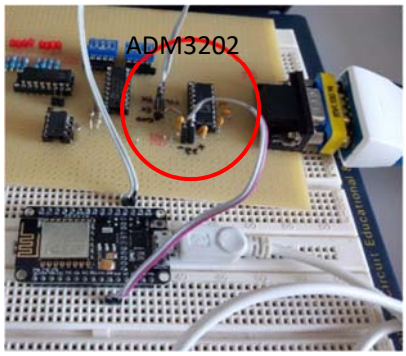


Figura 40. Montaje físico con conexión al RS232 y al ESP8266.

Así pues, el montaje final con el que se llevó a cabo la comprobación de la programación del bucle fue el siguiente:

Figura 42. Sistema final de comprobación del programa.

5. Diseño de las prácticas

5.1 Estructura de las prácticas

El diseño de esta aplicación tiene como propósito principal hacer que los alumnos adquieran una mayor soltura en el uso del microcontrolador, aprendiendo a configurar los diferentes elementos, a trabajar con secuencias de interrupciones, a recibir y transmitir información mediante la comunicación serie, etc.

No obstante, como se ha podido apreciar a lo largo de este documento, esta aplicación también profundiza en otros temas como son los sensores de temperatura, para determinar cuál es el óptimo para la aplicación, los amplificadores operacionales, para acondicionar la señal del sensor o el diseño de una placa de circuito impreso para reducir el ruido del sistema.

Debido a la amplia variedad de campos que toca esta aplicación se ha considerado que se debería dividir en diferentes partes, centrándose cada una en buscar solución a cada uno de los diferentes apartados de este trabajo. Por ello, se podrían diferenciar tres partes de una extensión similar:

1. El desarrollo del programa a utilizar en el microcontrolador.
2. La elección de un sensor con la creación de su correspondiente sistema acondicionador y el diseño y fabricación de una placa de circuito impreso.
3. La adición de un sistema de comunicación serie que permita un intercambio de información entre el microcontrolador y el PC a través de una página web.

5.2 Práctica 1

Esta primera práctica se dedicará al desarrollo del programa a utilizar en el microcontrolador. Este programa deberá ser capaz de obtener el período de una onda mediante la detección de dos puntos y el uso de un timer. Los puntos a buscar podrán ser o un mínimo y un máximo o dos mínimos o dos máximos. Este programa se va a centrar en ondas de un rango de frecuencias comprendido entre los 0.2 y los 0.4 Hz y de valor comprendido entre los 0 y los 3.6 V.

Los elementos de apoyo que se recomienda utilizar son blogs como el de embeddedrelated.com, para obtener información tanto sobre conceptos básicos del MSP430, como sobre la configuración del conversor analógico digital o las interrupciones; y las páginas tutorialspoint.com y wikibooks.org en las que hay tutoriales sobre el lenguaje de programación C, que es el que usa el microcontrolador.

Práctica 1

Programación del microcontrolador para la obtención del período de una onda.

En la siguiente práctica se va a proceder a calcular el período de una onda de frecuencia entre 0.2 y 0.4Hz utilizando para ello el microcontrolador MSP430G2553 y un generador de señales que produzca la onda deseada. Es importante tener en cuenta que el voltaje de entrada que admite el microcontrolador está entre 0 y 3.6V, por lo que la señal generada deberá encontrarse siempre entre estos valores para obtener una medición correcta.

La estructura del programa consistirá en un bucle en el que se comprobará si un pulsador se ha pulsado o no, y en caso de haberse pulsado se iniciará la búsqueda del máximo y del mínimo. Este proceso se iniciará con un escaneo de los valores que toma la onda, para establecer unos valores frontera a partir de los cuales la frecuencia de muestreo se incrementará, aumentando la precisión de la medida y evitando consumir muchos recursos durante todo el proceso.

Así pues, una vez establecidos los valores frontera se procederá a la búsqueda del primer valor. Se deberá tener en cuenta que la búsqueda debe comenzar siempre en un mismo estado, con la onda decreciendo o creciendo, por lo que se deberá añadir una espera que lo asegure.

Una vez se esté en el punto de partida se iniciará la búsqueda mediante la comparación de dos valores. Por ejemplo, en la búsqueda de un máximo, se guarda un valor, V1, y si el siguiente, V2, es mayor, se seguirá buscando y se sustituirá V1 por V2 hasta que encontremos un valor V2 que sea menor que V1, siendo V1 el máximo buscado.

Desde que se encuentre el primer punto hasta que se encuentre el segundo deberá funcionar un timer para que posteriormente, sabiendo los valores en cada uno de los puntos, se pueda calcular el período.

Para el correcto funcionamiento del programa será necesaria una buena configuración de los relojes del microcontrolador, del timer y del conversor analógico digital. Para la configuración de los relojes y del timer será necesario tener en cuenta los valores de las frecuencias con las que vamos a trabajar y seleccionar unos valores adecuados.

5.3 Práctica 2

En esta segunda práctica, se procederá a la selección de un sensor de temperatura apropiado para esta aplicación y a la creación de un circuito que acondicione la señal para que pueda ser detectada por el microcontrolador. El material de apoyo recomendado para esta parte de la práctica consiste en los apuntes de la asignatura de fundamentos de electrónica, así como las hojas de características de los componentes.

Una vez diseñado el circuito y comprobado en la placa de conexiones se procederá al diseño y fabricación del circuito impreso mediante el programa CircuitMaker. Para esta parte el material de apoyo recomendado consistirá en un video tutorial que ilustra el proceso completo mediante un ejemplo (https://www.youtube.com/watch?v=_T1c3oMAYTs&index=1&list=WL&t=203s). Para la selección de los componentes para la placa se recomiendan las páginas de RS y de Farnell por ser aquellas con las que trabaja la universidad (http://es.rs-online.com y http://es.farnell.com).

Práctica 2

Selección del sensor de temperatura y diseño y montaje del circuito acondicionador.

En esta práctica se va a llevar a cabo la elección de un sensor de temperatura de tres posibles elecciones. El sensor elegido deberá cumplir con los requisitos de precisión necesarios para llevar a cabo esta aplicación.

Para llevar a cabo la selección en varios casos será necesario montar un circuito amplificador que permita mostrar una señal visible en el osciloscopio, ya que, las variaciones que muestran algunos sensores son menores que el ruido electromagnético del laboratorio.

Una vez realizada la elección del componente de manera justificada se procederá al diseño del circuito acondicionador. Este circuito, a diferencia del anterior que tenía una función simplemente amplificadora, deberá hacer que la señal esté entre los 0 y los 3.6V. Esta labor se realizará primero mediante simulaciones en el programa LTspice y posteriormente se montará en la placa de conexiones para comprobar el correcto funcionamiento del mismo.

Por último, tras llevar a cabo el diseño de un circuito acondicionador funcional se procederá al montaje de ese circuito en una placa de circuito impreso. Para esta labor se utilizará en programa CircuitMaker, que permite la selección de los componentes (preferentemente “through holes”) a utilizar y permite a otros usuarios evaluar tu trabajo online. Una vez diseñada y tras su fabricación (por parte de los maestros de taller) se procederá al soldado de los componentes.

Finalmente se comprobará el correcto funcionamiento del circuito con el osciloscopio y con el microcontrolador al que se conectará para comprobar el resultado final de ambas prácticas.

5.4 Práctica 3

En la tercera práctica se llevarán a cabo cambios sobre el código de la primera práctica, se añadirán dos subrutinas de interrupciones para la recepción y la transmisión de datos mediante el puerto serie. Esto se llevará a cabo con el material de apoyo siguiente: los blogs de bennthomsen.wordpress.com en el que existe una introducción al uso del MSP430G2553 que habla sobre la comunicación serie, y comentado anteriormente embeddedrelated.com; también se recomienda la página http://xanthium.in/Serial-Communication-MSP430-UART-USCI_A en la que se hace una revisión muy completa de los registros necesarios para la comunicación serie.

Además de los cambios en el código ya creado, se creará otro en la IDE de Arduino para programar el módulo wifi y el diseño de la página web a través de la cual se mostrarán las medidas. Al igual que en la primera práctica, se recomienda utilizar como material de apoyo las páginas tutorialspoint.com y wikibooks.org, en las que hay tutoriales para la programación en Arduino y en HTML.

Práctica 3

Adición de un sistema de comunicación serie que permita un intercambio de información entre el microcontrolador y el PC.

En esta última práctica se procederá primeramente a la incorporación de dos subrutinas de interrupciones que permitan la recepción y la transmisión de datos desde el microcontrolador. Para ello se deberá configurar el UART y tener en cuenta que la frecuencia de reloj no puede ser muy baja, ya que si no el microcontrolador no será capaz de leer correctamente los mensajes que reciba.

Una vez cambiado el código de la primera práctica será necesario programar un módulo wifi para que a través de una página web se puedan pedir los datos. Para ello se utilizará la IDE de Arduino y será necesario un dispositivo móvil en el que se pueda crear una red wifi a la que se pueda conectar este módulo y los usuarios que quieran ver los datos.

6. Conclusiones y líneas futuras

En esta plataforma de prácticas se ha desarrollado un sistema de medición de la tasa respiratoria mediante el uso de un sensor de temperatura.

Mediante el estudio del estado del arte de los sistemas de medición de la tasa respiratoria se ha aprendido sobre la situación actual de estos sistemas y sobre los diferentes tipos. De acuerdo con los principios de economía y tiempo de desarrollo se optó por los sensores de temperatura para esta aplicación.

Este sensor ha necesitado de un sistema amplificador debido a las pequeñas variaciones de voltaje producidas por los cambios de temperatura. Esta ha sido la parte más problemática del proyecto, debido a los numerosos cambios que se han tenido que realizar. Además, en el desarrollo de la PCB se obtuvieron numerosos errores debido a la falta de experiencia, pero fueron solventados todos mediante la aplicación de los conocimientos de electrónica.

El algoritmo principal para la obtención del período se ha programado en el microcontrolador MSP430G2553, ya que el objetivo de las prácticas es el de aumentar el conocimiento y manejo de este dispositivo por parte de los alumnos. Esta parte ha supuesto un gran aprendizaje en cuanto al manejo de interrupciones y del conversor analógico digital.

Finalmente se ha añadido un sistema de comunicación por wifi mediante el dispositivo ESP8266, que permite obtener las medidas con el uso de una página web. Mediante este módulo se ha llevado a cabo una iniciación a la programación tanto en Arduino como en HTML.

Además, para el desarrollo de las prácticas se ha suministrado materiales de apoyo que faciliten su desarrollo y aprendizaje al alumno.

En cuanto a los objetivos marcados se han cumplido todos, aunque se podría seguir trabajando en algunos aspectos del proyecto, como, por ejemplo:

- Aumentar la precisión en la medición del período, ya que las medidas obtenidas siempre constan de un error entre un 1 y un 25%, haciendo que en algunos casos sean necesarias varias medidas para la obtención de una estimación apropiada.
- Mejora del sistema de alimentación de la placa de circuito impreso, ya que, en su momento, debido al desconocimiento de la placa ESP8266, se optó por una alimentación mediante una conexión USB. Sin embargo, dicha placa puede suministrar una salida de 5V, lo que sería muy adecuado para alimentar la PCB sin necesidad de una conexión USB, teniendo todo el sistema una única alimentación de 5V en lugar de dos.

7. Referencias

[1] Se Dong Min, Hangsik Shin, Yonghyeon Yun, Chungkeun Lee. “Noncontact Respiration Rate Measurement System Using an Ultrasonic Proximity Sensor”. IEEE Sensors Journal, Vol. 10, No. 11, November 2010 (1732-1739).

[2] William Daw, Ruth Kingshott, Reza Saatchi, Derek Burke, Alan Holloway, Jon Travis,Rob Evans, Anthony Jones, Ben Hughes and Heather Elphick. “Medical Devices for Measuring Respiratory Rate in Children: a Review”. Journal of Advances in Biomedical Engineering and Technology, Vol. 3, No. 1, 2016 (21-27).

[3] J. M. Marín Trigo. “Plestimografía inductiva en la monitorización respiratoria”. Archivos de bronconeumología. Vol. 24, Núm. 2, 1988 (78-80).

[4] Jin Fei and Ioannis Pavlidis. “Thermistor at a Distance: Unobtrusive Measurement of Breathing”. IEEE Transactions on Biomedical Engineering, Vol. 57, No. 4, April 2010 (988-998).

8. Bibliografía

[5] Texas Instruments, "User's guide MSP430x2xx", [En línea]. Available: http://www.ti.com/lit/ug/slau144j/slau144j.pdf (21/9/2017).

[6] Texas Instruments, "Datasheet LM35", [En línea]. Available: http://www.ti.com/lit/ds/symlink/lm35.pdf (21/9/2017).

[7] Analog Devices, "Datasheet AN-369", [En línea]. Available: http://www.analog.com/media/en/technical-documentation/application-notes/AN-369.pdf (21/9/2017).

[8] Analog Devices, "Datasheet AD-8494", [En línea]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/AD8494_8495_8496_8497.pdf (21/9/2017).

[9] Allied Electronics, "Datasheet ZTP-135SR", [En línea]. Available: https://www.alliedelec.com/m/d/9fa458adb9114ab6a505eed592be56aa.pdf (21/9/2017).

[10] Microchip, "Datasheet MCP6002", [En línea]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21733j.pdf (21/9/2017).

[11] OrCAD, "OrCAD Capture CIS Demo", [En línea]. Available: http://www.orcad.com/resources/orcad-downloads (21/9/2017).

[12] LTspice, "LTspice", [En línea]. Available: http://www.linear.com/designtools/software/ (21/9/2017).

[13] CircuitMaker, "CircuitMaker", [En línea]. Available: https://circuitmaker.com/ (21/9/2017).

[14] Texas Instruments, "Datasheet MAX660", [En línea]. Available: http://www.ti.com/lit/ds/symlink/max660.pdf (21/9/2017).

[15] Espressif, "Datasheet ESP8266", [En línea]. Available: http://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf (21/9/2017).

[16] CoolTerm, “CoolTerm”, [En línea]. Available: http://freeware.the-meiers.org/ (21/9/2017).

[17] Arduino, “ARDUINO 1.8.4”, [En línea]. Available: https://www.arduino.cc/en/Main/Software (21/9/2017).

[18] Analog Devices, “Datasheet ADM3202”, [En línea]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/ADM3202_3222_1385.pdf (21/9/2017).

[19] Embedded related, “MSP430 Launchpad Tutorial”, https://www.embeddedrelated.com/showarticle/179.php (21/9/2017).

[20] TutorialsPoint, “C Tutorial”, https://www.tutorialspoint.com/cprogramming/index.htm (21/9/2017).

[21] TutorialsPoint, “Arduino Tutorial”, https://www.tutorialspoint.com/arduino/index.htm (21/9/2017).

[22] TutorialsPoint, “HTML Tutorial”, https://www.tutorialspoint.com/html/index.htm (21/9/2017).

[23] WikiLibros, “Programación en C”, https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C (21/9/2017).

[24] WikiLibros, “Lenguaje de programación Arduino”, https://es.wikibooks.org/wiki/Lenguaje_de_programaci%C3%B3n_Arduino#E.2FS_anal.C3.B3gica (21/9/2017).

[25] WikiLibros, “Lenguaje HTML”, https://es.wikibooks.org/wiki/Lenguaje_HTML (21/9/2017).

[26] Benn Thomsen, “Introducing the MSP430G2553 Hardware”, https://bennthomsen.wordpress.com/engineering-toolbox/ti-msp430-launchpad/introducing-the-msp430g2553-hardware/ (21/9/2017).

[27] Benn Thomsen, “Hardware UART”, https://bennthomsen.wordpress.com/engineering-toolbox/ti-msp430-launchpad/msp430g2553-hardware-uart/ (21/9/2017).

[28] Xantium Enterprises, “Serial Communication using MSP430 UART(USCI_A)”, http://xanthium.in/Serial-Communication-MSP430-UART-USCI_A (21/9/2017).

9. Lista de figuras

ANEXO 1. Código del microcontrolador

1.1 Código principal del microcontrolador

```
#include <msp430g2553.h>

#define RUIDO 10
#define MUESTRA 200

volatile unsigned int Voltvalor, MAX, MIN, DECRECIENDO, BUSCANDO, APROBACION, MARCHA, VMAX, VMIN, Longitud;
long int tiempo[2], suma, Tfirst, Tsecond;
long int PERIODO;
unsigned int leidas, i; //Counter
unsigned int Buffer_SAL[20], Buffer_ENT[20]; //Variable de conformar MSJ
unsigned int CHK_SALIDA = 0x0000, CHK_ENTRADA = 0x0000; //Variable de conformar CHK del MSJ
int Vuelta;

void InitializeClocks(void);
void InitializePins(void);
void InitializeTimer(void);
void InitializeADC10(void);
void InitializeUART(void);
void BuscarMaxMin(volatile unsigned int *ALTO, volatile unsigned int *BAJO, long int *TIEMPO);
void main(void) {


    WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer

    InitializeClocks();
    InitializePins();
    InitializeADC10();
    InitializeUART();

//---------------- Activación de las interrupciones ------------------//

    IE2 |= UCA0RXIE;            // Interrupción de recepción
    _BIS_SR(GIE);               // Interrupción de transmisión

//---------------- ------------------------------- ------------------//


    while (1) {
        if (APROBACION) {
                P1OUT |= BIT6;
                P1OUT |= BIT0;
                BuscarMaxMin( &MAX, &MIN, &PERIODO);
                i = 0;
                APROBACION=0;
        }
    }
}

void InitializeClocks(void) {
    BCSCTL1 = CALBC1_1MHZ;   // Pongo el reloj a 1 MHz.
    DCOCTL = CALDCO_1MHZ;
    BCSCTL2 = SELM_0 ;       // PONGO EL RELOJ A 125000 Hz
}

void InitializePins(void) {
```

```
56.           P1DIR &= ~BIT3;         //Configuro el Pin 1.3 como pulsador.
57.           P1OUT |= BIT3;
58.           P1REN |= BIT3;
59.           P1IES |= BIT3; // Activar las interrupciones al pulsar
60.           P1IFG &= ~BIT3;
61.           P1IE  |= BIT3; // Activo las interrupciones del pulsador 1.3
62.
63.
64.           P1OUT &= ~BIT7; // Configuro el PIN 1.7 para recibir la señal.
65.           P1OUT |= BIT7;
66.           P1DIR &= ~BIT7;
67.           P1REN |= BIT7;
68.
69.           P1OUT &= ~BIT0; //Configuro el PIN 1.0 para asociarlo al led rojo.
70.           P1DIR |= BIT0;
71.
72.           P1OUT &= ~BIT6; //Configuro el PIN 1.6 para asociarlo al led verde.
73.           P1DIR |= BIT6;
74.
75.           //--------- Setting the UART function for P1.1 & P1.2 --------//
76.
77.         P1SEL  |=  BIT1 + BIT2;  // P1.1 UCA0RXD input
78.         P1SEL2 |=  BIT1 + BIT2;  // P1.2 UCA0TXD output
79.       }
80.
81.
82.       void InitializeTimer(void) {
83.           TA0CTL = TASSEL_2 + ID_3 + MC_1 + TACLR + TAIE; //Reloj smckl, divido por
          8 la frecuencia, modo up, resetea valores anteriores.
84.           TA0CCR0 = 62500;        //0.001 seg = 125 ciclos     cada 4 segundos salta

85.       }
86.
87.       void InitializeADC10(void) {
88.           ADC10CTL0 |= SREF_0 + ADC10ON;      // VCC y VSS.
89.           ADC10CTL1 |= ADC10SSEL_0 + INCH_7 + CONSEQ_0 + SHS_0;// Reloj MCLK + Canal
           de entrada 7 + Canal único.
90.           ADC10AE0 |= BIT7;                           // Pin 7 como entrada analógic
          a.
91.
92.       }
93.
94.       void InitializeUART(void) {
95.
96.         UCA0CTL1 |=  UCSSEL_2 + UCSWRST;  // USCI Clock = SMCLK,USCI_A0 disabled
97.         UCA0BR0   =  104;                 // 104 From datasheet table-
98.         UCA0BR1   =  0;                   // -selects baudrate =9600,clk = SMCLK
99.         UCA0MCTL  =  UCBRS_1;             // Modulation value = 1 from datasheet
100.
101.         UCA0CTL1 &= ~UCSWRST;             // Clear UCSWRST to enable USCI_A0
102.      }
103.
104.       void BuscarMaxMin(volatile unsigned int *ALTO, volatile unsigned int *BAJO, l
          ong int *TIEMPO){
105.
106.           IE2 &= ~UCA0RXIE;
107.           BCSCTL2 = DIVM_3 + DIVS_3;   // PONGO EL RELOJ A 125000 Hz
108.           leidas = 0;
109.           VMAX = 0;
110.           VMIN = 1000;
111.           *TIEMPO = 0;
112.           Vuelta=-1;
113.           i=0;
114.
115.
```

```
116.      //---------------- Establecimiento de limites superior e inferior ------------
          ------//
117.
118.
119.          while (i < MUESTRA) {
120.              __delay_cycles(3750);                       //Espera 0.03 segundos
121.              ADC10CTL0 |= ENC + ADC10SC;
122.              while (ADC10CTL1 & ADC10BUSY == 1) {
123.              };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
124.              if (ADC10MEM > VMAX) {
125.                  VMAX = ADC10MEM;
126.              }
127.              if (ADC10MEM < VMIN) {
128.                  VMIN = ADC10MEM;
129.              }
130.              i++;
131.          }
132.          VMAX = (VMAX - VMIN) * 2 / 3 + VMIN;
133.          VMIN = (VMAX + VMIN) / 2;
134.
135.      //---------------
          - ESPERA HASTA QUE EMPIEZA A CRECER LA ONDA COMPROBANDO QUE CRECE PRIMERO Y DE
          SPUES DECRECE ------------------//
136.          DECRECIENDO = 0;
137.          leidas = 0;
138.          i = 0;
139.
140.          while (!(DECRECIENDO)) {
141.              leidas = 0;
142.              suma = 0;
143.              while (leidas < RUIDO) {
144.                  __delay_cycles(1250);           //0.01 SEG
145.
146.                  ADC10CTL0 |= ENC + ADC10SC;
147.
148.                  while (ADC10CTL1 & ADC10BUSY == 1) {
149.                  };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
150.                  suma = suma + ADC10MEM;
151.                  leidas = leidas + 1;
152.              }
153.
154.              Voltvalor = suma / RUIDO;
155.              i = i + 1;
156.              if (i == 1) {
157.                  *ALTO = Voltvalor;
158.              } else {
159.                  if (Voltvalor < *ALTO) {
160.                      DECRECIENDO = 1;
161.                  } else {
162.                      *ALTO = Voltvalor;
163.                  }
164.              }
165.
166.          }
167.
168.          while (DECRECIENDO) {
169.              leidas = 0;
170.              suma = 0;
171.              while (leidas < RUIDO) {
172.                  __delay_cycles(1250);           //0.01 SEG
173.
174.                  ADC10CTL0 |= ENC + ADC10SC;
175.
176.                  while (ADC10CTL1 & ADC10BUSY == 1) {
177.                  };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
178.                  suma = suma + ADC10MEM;
```

```
                leidas = leidas + 1;
            }

            Voltvalor = suma / RUIDO;
            i = i + 1;
            if (i == 1) {
                *ALTO = Voltvalor;
            } else {
                if (Voltvalor > *ALTO) {
                    DECRECIENDO = 0;
                } else {
                    *ALTO = Voltvalor;
                }
            }

        }

        //---------------- Búsqueda del máximo ------------------//

        InitializeTimer();

        i = 0;
        BUSCANDO = 1;

        while (!(DECRECIENDO)) {
            leidas = 0;
            suma = 0;
            while (leidas < RUIDO) {
                __delay_cycles(1250);           //0.01 SEG

                ADC10CTL0 |= ENC + ADC10SC;

                while (ADC10CTL1 & ADC10BUSY == 1) {
                };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
                suma = suma + ADC10MEM;
                leidas = leidas + 1;
            }

            Voltvalor = suma / RUIDO;
            i = i + 1;
            if (i == 1) {
                *ALTO = Voltvalor;
            } else if (Voltvalor >= *ALTO) {
                *ALTO = Voltvalor;
            }

            if ((Voltvalor >= VMAX)) {
                tiempo[0] = TA0R;
                while (BUSCANDO) {

                    leidas = 0;
                    suma = 0;
                    while (leidas < RUIDO) {
                        __delay_cycles(100);            //0.001 SEG

                        ADC10CTL0 |= ENC + ADC10SC;

                        while (ADC10CTL1 & ADC10BUSY == 1) {
                        };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
                        suma = suma + ADC10MEM;
                        leidas = leidas + 1;
                    }

                    Voltvalor = suma / RUIDO;
                    tiempo[1] = TA0R;
                    if (Voltvalor >= *ALTO) {
```

```
                        *ALTO = Voltvalor;
                        tiempo[0] = tiempo[1];
                    } else {
                        BUSCANDO = 0;
                        DECRECIENDO = 1;
                        Tfirst = tiempo[0];
                    }
                }
            }
        }
        TA0CTL |= TAIE;
        //---------------- Búsqueda del mínimo ------------------//
        i = 0;
        BUSCANDO = 1;

        while ((DECRECIENDO)) {                 //BUSCO EL MINIMO
            leidas = 0;
            suma = 0;
            while (leidas < RUIDO) {
                __delay_cycles(1250);           //0.01 SEG

                ADC10CTL0 |= ENC + ADC10SC;

                while (ADC10CTL1 & ADC10BUSY == 1) {
                };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
                suma = suma + ADC10MEM;
                leidas = leidas + 1;
            }

            Voltvalor = suma / RUIDO;
            i = i + 1;
            if (i == 1) {
                *BAJO = Voltvalor;
            } else if (Voltvalor <= *BAJO) {
                *BAJO = Voltvalor;
            }

            if ((Voltvalor <= VMIN)) {
                tiempo[0] = TA0R;
                while (BUSCANDO) {

                    leidas = 0;
                    suma = 0;
                    while (leidas < RUIDO) {
                        __delay_cycles(100);            //0.001 SEG

                        ADC10CTL0 |= ENC + ADC10SC;

                        while (ADC10CTL1 & ADC10BUSY == 1) {
                        };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
                        suma = suma + ADC10MEM;
                        leidas = leidas + 1;
                    }
                    tiempo[1] = TA0R;
                    Voltvalor = suma / RUIDO;

                    if (Voltvalor <= *BAJO) {
                        *BAJO = Voltvalor;
                        tiempo[0] = tiempo[1];
                    } else {
                        BUSCANDO = 0;
                        DECRECIENDO = 0;
                        Tsecond = tiempo[0];
                    }
                }
            }
```

```
311.          }
312.          TA0CTL &= ~ TAIE;       //DESACTIVAR INTERRUPCIONES DEL TIMER PARA EVITAR
      QUE SE ATASQUE
313.          if (Vuelta<0) {
314.              *TIEMPO = (Tsecond-Tfirst)/100;
315.              *TIEMPO=*TIEMPO*1.28;
316.          }
317.          else {
318.              *TIEMPO = (Tsecond+(62500-Tfirst)+Vuelta*62500)/100;
319.              *TIEMPO=*TIEMPO*1.28;
320.          }
321.
322.          P1OUT &= ~ BIT0;
323.          InitializeClocks();     //PARA QUE EL TX Y RX PUEDAN FUNCIONAR CORRECTAMEN
      TE
324.          IE2 |= UCA0RXIE;                // Enable the receive interrupt
325.      }
326.
327.      #pragma vector=PORT1_VECTOR
328.      __interrupt void PORT1_ISR(void){
329.
330.          APROBACION=1;
331.          P1IFG &= ~BIT3;
332.      }
333.
334.
335.      #pragma vector = TIMER0_A1_VECTOR
336.      __interrupt void TimerInterrupt(void){
337.
338.          Vuelta=Vuelta+1;
339.          TA0CTL &= ~TAIFG;
340.      }
341.
342.      #pragma vector = USCIAB0RX_VECTOR
343.      __interrupt void ReceiveInterrupt(void) {
344.
345.          if (PERIODO == 0) {
346.              IE2 &= ~UCA0RXIE;
347.
348.              UCA0TXBUF = 'P';
349.              __delay_cycles(1000);
350.              UCA0TXBUF = '1';
351.              __delay_cycles(1000);
352.              UCA0TXBUF = '.';
353.              __delay_cycles(1000);
354.              UCA0TXBUF = '3';
355.              __delay_cycles(1000);
356.              __delay_cycles(1000);
357.              UCA0TXBUF = ' ';
358.
359.              IE2 |= UCA0RXIE;
360.              IFG2 &= ~UCA0RXIFG; // Clear RX flag
361.          }
362.          else {
363.
364.              P1OUT &= ~ BIT6;
365.              P1OUT |= BIT0;
366.              Buffer_ENT[i] = UCA0RXBUF;
367.              i = i > 19 ? 0 : i + 1;//Ternary operator https://en.wikipedia.org/wik
      i/%3F:#C
368.              if (i == 4) {
369.                  if ((Buffer_ENT[0] != 0x80) || (Buffer_ENT[1] != 0x01) || (Buffer_
      ENT[2] != 0x02) || (Buffer_ENT[3] != 0x03)) {
370.                      i = 0;
371.                      P1OUT &= ~ BIT0;
372.                      P1OUT |= BIT6;
```

```
                    }
                }
                if (i == 5) {
                    Longitud = Buffer_ENT[4];
                }
                if (i == (5 + Longitud + 2)) {
                    CHK_ENTRADA = 0x0000;
                    for (i = 5; i < (5 + Longitud); i++) {
                        CHK_ENTRADA = Buffer_ENT[i] + CHK_ENTRADA;
                    }
                    if ((Buffer_ENT[(5 + Longitud)] == ((CHK_ENTRADA  & 0xFF00)>> 8))
        || (Buffer_ENT[(5 + Longitud + 1)] ==(CHK_ENTRADA & 0x00FF))) {
                        i = (5 + Longitud + 2);
                    }
                    else{
                        i = 0;
                        P1OUT &= ~ BIT0;
                        P1OUT |= BIT6;
                    }
                }
                if (i == (5 + Longitud + 4)) {
                    if ((Buffer_ENT[5 + Longitud + 2] == 0xA5) && (Buffer_ENT[5 + Long
        itud + 3] == 0xA5)) {
                        IE2 &= ~UCA0RXIE;
                        IE2 |= UCA0TXIE;                // Enable the Transmit interrup
        t
                    }
                    else {
                            i=0;
                            P1OUT &= ~ BIT0;
                            P1OUT |= BIT6;
                    }

            }
        }

            IFG2 &= ~UCA0RXIFG; // Clear RX flag
        }

        #pragma vector = USCIAB0TX_VECTOR
        __interrupt void TransmitInterrupt(void) {

            switch (Buffer_ENT[5]) {
            case 0x5A:
                Buffer_SAL[0] = 0x80; //Se manda cabecera del mensaje caracter
                Buffer_SAL[1] = 0x01; //Se manda cabecera del mensaje caracter
                Buffer_SAL[2] = 0x02; //Se manda cabecera del mensaje caracter
                Buffer_SAL[3] = 0x03; //Se manda cabecera del mensaje caracter
                Buffer_SAL[4] = 0x09; //Longitud del mensaje '14'
                Buffer_SAL[5] = Buffer_ENT[5];  //Tipo de mensaje '5A'
                Buffer_SAL[6] = Buffer_ENT[6];  //Destino del mensaje '01'
                Buffer_SAL[7] = Buffer_ENT[7];  //Origen del mensaje '02'
                Buffer_SAL[8] = (PERIODO & 0xFF00)>> 8;       //Medida 1
                Buffer_SAL[9] =  PERIODO & 0x00FF;          //Medida 1
                Buffer_SAL[10] =(MIN & 0xFF00)>> 8;           //Medida 2
                Buffer_SAL[11] = MIN & 0x00FF;              //Medida 2
                Buffer_SAL[12] =(MAX & 0xFF00)>> 8;           //Medida 3
                Buffer_SAL[13] = MAX & 0x00FF;              //Medida 3
                Buffer_SAL[16] = 0xA5;  //Caracteres de final de mensaje
                Buffer_SAL[17] = 0xA5;  //Caracteres de final de mensaje
                CHK_SALIDA = Buffer_SAL[5] + Buffer_SAL[6] + Buffer_SAL[7] + Buffer_SA
        L[8] + Buffer_SAL[9] + Buffer_SAL[10] + Buffer_SAL[11]+ Buffer_SAL[12]+ Buffer
        _SAL[13];
                Buffer_SAL[14] = (CHK_SALIDA  & 0xFF00)>> 8; //Carga la parte alta del
         CHECKSUM de salida
```

```
432.                Buffer_SAL[15] = CHK_SALIDA & 0x00FF; //Se carga la parte baja del CHE
        CKSUM de salida
433.
434.                UCA0TXBUF = 0;
435.                Longitud = 5 + Buffer_SAL[4] + 4;
436.                for (i = 0; i < (Longitud); i++) {
437.                    UCA0TXBUF = Buffer_SAL[i];
438.                    __delay_cycles(1000);
439.                }
440.                break;
441.            }
442.
443.            for(i=0;i<20; i++) {
444.                Buffer_ENT[i]=0;
445.            }
446.
447.            __delay_cycles(100000);
448.            P1OUT &= ~ BIT0;
449.            P1OUT |= BIT6;
450.            i = 0;
451.            Longitud = 0;
452.
453.            IE2 &= ~UCA0TXIE;
454.            IE2 |= UCA0RXIE;
455.
456.        }
```

1.2 Función principal buscando dos máximos

```
1.      //---------------- Búsqueda del primer máximo ------------------//
2.
3.          InitializeTimer();
4.
5.          i = 0;
6.          BUSCANDO = 1;
7.
8.          while (!(DECRECIENDO)) {
9.              leidas = 0;
10.             suma = 0;
11.             while (leidas < RUIDO) {
12.                 __delay_cycles(1250);           //0.01 SEG
13.
14.                 ADC10CTL0 |= ENC + ADC10SC;
15.
16.                 while (ADC10CTL1 & ADC10BUSY == 1) {
17.                 };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
18.                 suma = suma + ADC10MEM;
19.                 leidas = leidas + 1;
20.             }
21.
22.             Voltvalor = suma / RUIDO;
23.             i = i + 1;
24.             if (i == 1) {
25.                 *ALTO = Voltvalor;
26.             } else if (Voltvalor >= *ALTO) {
27.                 *ALTO = Voltvalor;
28.             }
29.
30.             if ((Voltvalor >= VMAX)) {
31.                 tiempo[0] = TA0R;
32.                 while (BUSCANDO) {
33.
34.                     leidas = 0;
35.                     suma = 0;
36.                     while (leidas < RUIDO) {
37.                         __delay_cycles(100);            //0.001 SEG
38.
39.                         ADC10CTL0 |= ENC + ADC10SC;
40.
41.                         while (ADC10CTL1 & ADC10BUSY == 1) {
42.                         };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
43.                         suma = suma + ADC10MEM;
44.                         leidas = leidas + 1;
45.                     }
46.
47.                     Voltvalor = suma / RUIDO;
48.                     tiempo[1] = TA0R;
49.                     if (Voltvalor >= *ALTO) {
50.                         *ALTO = Voltvalor;
51.                         tiempo[0] = tiempo[1];
52.                     } else {
53.                         BUSCANDO = 0;
54.                         DECRECIENDO = 1;
55.                         Tfirst = tiempo[0];
56.                     }
57.                 }
58.             }
59.         }
60.         TA0CTL |= TAIE;
```

```
61.         //---------ESPERA A QUE CREZCA LA ONDA---------//
62.         DECRECIENDO = 1;
63.         leidas = 0;
64.         i = 0;
65.         while (DECRECIENDO) {
66.             leidas = 0;
67.             suma = 0;
68.             while (leidas < RUIDO) {
69.                 __delay_cycles(1250);           //0.01 SEG
70.
71.                 ADC10CTL0 |= ENC + ADC10SC;
72.
73.                 while (ADC10CTL1 & ADC10BUSY == 1) {
74.                 };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
75.                 suma = suma + ADC10MEM;
76.                 leidas = leidas + 1;
77.             }
78.
79.             Voltvalor = suma / RUIDO;
80.             i = i + 1;
81.             if (i == 1) {
82.                 *ALTO = Voltvalor;
83.             } else {
84.                 if (Voltvalor > *ALTO) {
85.                     DECRECIENDO = 0;
86.                 } else {
87.                     *ALTO = Voltvalor;
88.                 }
89.             }
90.
91.         }
92.         //---------------- Búsqueda del segundo máximo ------------------//
93.
94.         InitializeTimer();
95.
96.         i = 0;
97.         BUSCANDO = 1;
98.
99.         while (!(DECRECIENDO)) {
100.            leidas = 0;
101.            suma = 0;
102.            while (leidas < RUIDO) {
103.                __delay_cycles(1250);           //0.01 SEG
104.
105.                ADC10CTL0 |= ENC + ADC10SC;
106.
107.                while (ADC10CTL1 & ADC10BUSY == 1) {
108.                };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
109.                suma = suma + ADC10MEM;
110.                leidas = leidas + 1;
111.            }
112.
113.            Voltvalor = suma / RUIDO;
114.            i = i + 1;
115.            if (i == 1) {
116.                *BAJO = Voltvalor;
117.            } else if (Voltvalor >= *BAJO) {
118.                *BAJO = Voltvalor;
119.            }
120.
121.            if ((Voltvalor >= VMAX)) {
122.                tiempo[0] = TA0R;
123.                while (BUSCANDO) {
124.
125.                    leidas = 0;
126.                    suma = 0;
```

```
127.                    while (leidas < RUIDO) {
128.                        __delay_cycles(100);            //0.001 SEG
129.
130.                        ADC10CTL0 |= ENC + ADC10SC;
131.
132.                        while (ADC10CTL1 & ADC10BUSY == 1) {
133.                        };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
134.                        suma = suma + ADC10MEM;
135.                        leidas = leidas + 1;
136.                    }
137.
138.                    Voltvalor = suma / RUIDO;
139.                    tiempo[1] = TA0R;
140.                    if (Voltvalor >= *BAJO) {
141.                        *BAJO = Voltvalor;
142.                        tiempo[0] = tiempo[1];
143.                    } else {
144.                        BUSCANDO = 0;
145.                        DECRECIENDO = 1;
146.                        Tsecond = tiempo[0];
147.                    }
148.                }
149.            }
150.        }
151.        TA0CTL &= ~ TAIE;       //DESACTIVAR INTERRUPCIONES DEL TIMER PARA EVITAR QUE
            SE ATASQUE
152.        if (Vuelta<0) {
153.            *TIEMPO = (Tsecond-Tfirst)/100*0.64;
154.        }
155.        else {
156.            *TIEMPO = (Tsecond+(62500-Tfirst)+Vuelta*62500)/100*0.64;
157.        }
158.
159.        P1OUT &= ~ BIT0;
160.        InitializeClocks();     //PARA QUE EL TX Y RX PUEDAN FUNCIONAR CORRECTAMENTE

161.        IE2 |= UCA0RXIE;                // Enable the receive interrupt
162.    }
```

1.3 Función principal buscando dos mínimos

```
1.        //---------------- Búsqueda del primer mínimo ------------------//
2.            i = 0;
3.            BUSCANDO = 1;
4.
5.            while ((DECRECIENDO)) {                 //BUSCO EL MINIMO
6.                leidas = 0;
7.                suma = 0;
8.                while (leidas < RUIDO) {
9.                    __delay_cycles(1250);           //0.01 SEG
10.
11.                   ADC10CTL0 |= ENC + ADC10SC;
12.
13.                   while (ADC10CTL1 & ADC10BUSY == 1) {
14.                   };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
15.                   suma = suma + ADC10MEM;
16.                   leidas = leidas + 1;
17.               }
18.
19.               Voltvalor = suma / RUIDO;
20.               i = i + 1;
21.               if (i == 1) {
22.                   *ALTO = Voltvalor;
23.               } else if (Voltvalor <= *ALTO) {
24.                   *ALTO = Voltvalor;
25.               }
26.
27.               if ((Voltvalor <= VMIN)) {
28.                   tiempo[0] = TA0R;
29.                   while (BUSCANDO) {
30.
31.                       leidas = 0;
32.                       suma = 0;
33.                       while (leidas < RUIDO) {
34.                           __delay_cycles(100);            //0.001 SEG
35.
36.                           ADC10CTL0 |= ENC + ADC10SC;
37.
38.                           while (ADC10CTL1 & ADC10BUSY == 1) {
39.                           };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
40.                           suma = suma + ADC10MEM;
41.                           leidas = leidas + 1;
42.                       }
43.                       tiempo[1] = TA0R;
44.                       Voltvalor = suma / RUIDO;
45.
46.                       if (Voltvalor <= *BAJO) {
47.                           *BAJO = Voltvalor;
48.                           tiempo[0] = tiempo[1];
49.                       } else {
50.                           BUSCANDO = 0;
51.                           DECRECIENDO = 0;
52.                           Tfirst = tiempo[0];
53.                       }
54.                   }
55.               }
56.           }
57.           TA0CTL |= TAIE;
58.
59.       //---------------Espera a que la onda decrezca----------------//
60.           DECRECIENDO = 0;
61.           leidas = 0;
```

```
62.             i = 0;
63.
64.             while (!(DECRECIENDO)) {
65.                 leidas = 0;
66.                 suma = 0;
67.                 while (leidas < RUIDO) {
68.                     __delay_cycles(1250);           //0.01 SEG
69.
70.                     ADC10CTL0 |= ENC + ADC10SC;
71.
72.                     while (ADC10CTL1 & ADC10BUSY == 1) {
73.                     };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
74.                     suma = suma + ADC10MEM;
75.                     leidas = leidas + 1;
76.                 }
77.
78.                 Voltvalor = suma / RUIDO;
79.                 i = i + 1;
80.                 if (i == 1) {
81.                     *ALTO = Voltvalor;
82.                 } else {
83.                     if (Voltvalor < *ALTO) {
84.                         DECRECIENDO = 1;
85.                     } else {
86.                         *ALTO = Voltvalor;
87.                     }
88.                 }
89.
90.             }
91.
92.             //---------------- Búsqueda del segundo mínimo ------------------//
93.             i = 0;
94.             BUSCANDO = 1;
95.
96.             while ((DECRECIENDO)) {                 //BUSCO EL MINIMO
97.                 leidas = 0;
98.                 suma = 0;
99.                 while (leidas < RUIDO) {
100.                    __delay_cycles(1250);           //0.01 SEG
101.
102.                    ADC10CTL0 |= ENC + ADC10SC;
103.
104.                    while (ADC10CTL1 & ADC10BUSY == 1) {
105.                    };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
106.                    suma = suma + ADC10MEM;
107.                    leidas = leidas + 1;
108.                }
109.
110.                Voltvalor = suma / RUIDO;
111.                i = i + 1;
112.                if (i == 1) {
113.                    *BAJO = Voltvalor;
114.                } else if (Voltvalor <= *BAJO) {
115.                    *BAJO = Voltvalor;
116.                }
117.
118.                if ((Voltvalor <= VMIN)) {
119.                    tiempo[0] = TA0R;
120.                    while (BUSCANDO) {
121.
122.                        leidas = 0;
123.                        suma = 0;
124.                        while (leidas < RUIDO) {
125.                            __delay_cycles(100);            //0.001 SEG
126.
127.                            ADC10CTL0 |= ENC + ADC10SC;
```

```
128.
129.                             while (ADC10CTL1 & ADC10BUSY == 1) {
130.                             };  //ESPERA MIENTRAS REALIZA CONVERSIÓN.
131.                             suma = suma + ADC10MEM;
132.                             leidas = leidas + 1;
133.                         }
134.                         tiempo[1] = TA0R;
135.                         Voltvalor = suma / RUIDO;
136.
137.                         if (Voltvalor <= *BAJO) {
138.                             *BAJO = Voltvalor;
139.                             tiempo[0] = tiempo[1];
140.                         } else {
141.                             BUSCANDO = 0;
142.                             DECRECIENDO = 0;
143.                             Tsecond = tiempo[0];
144.                         }
145.                     }
146.                 }
147.             }
148.
149.             TA0CTL &= ~ TAIE;       //DESACTIVAR INTERRUPCIONES DEL TIMER PARA EVITAR
        QUE SE ATASQUE
150.             if (Vuelta<0) {
151.                 *TIEMPO = (Tsecond-Tfirst)/100*0.64;
152.             }
153.             else {
154.                 *TIEMPO = (Tsecond+(62500-Tfirst)+Vuelta*62500)/100*0.64;
155.             }
156.
157.             P1OUT &= ~ BIT0;
158.             InitializeClocks();     //PARA QUE EL TX Y RX PUEDAN FUNCIONAR CORRECTAMEN
        TE
159.             IE2 |= UCA0RXIE;                // Enable the receive interrupt
160.         }
```

ANEXO 2. Código principal del módulo wifi

```
1.        /*
2.          Este prgrama vale tanto para el modulo NODEMCU como para el EPS8266.
3.          Para cargarlo en el NODEMCU fabricado por AMICA -AESA- (NODEMCU LOLIN
4.          NO funciona) compilarlo con la placa=NodeMCU 1.0
5.          (ESP-12E Module), Flash Size=4M(3Mhz SPIFSS), CPU Frecuency=80Mhz
6.
7.
8.          Para cargarlo en ESP8266, compilarlo con la opcion GENERIC ESP8266 Module
9.          Flash mode= DIO, Flash Size=512k (64k SPIFFS), Debug Port=Disabled, Debug
10.         Level=Ninguno, Reset Method=ck, Flash Frecuency=40Mhz, CPU Frecuency=80Mhz
11.         Upload Speed=115200
12.         Las lineas TX y RX del FTDI o CH340 con el ESP-01 van cruzadas
13.       */
14.
15.       #include <ESP8266WiFi.h>
16.       #include <SoftwareSerial.h>
17.
18.       //Configuracion valores para IP estatica
19.       IPAddress ip(192, 168, 43, 90);
20.       IPAddress gateway(192, 168, 43, 1);
21.       IPAddress subnet(255, 255, 255, 0);
22.       IPAddress DNS(192, 168, 43, 1);
23.
24.       //const char* ssid = "wifimags";
25.       //const char* password = "wifimags";
26.
27.       const char* ssid = "TFG-Alberto";
28.       const char* password = "ESP-8266";
29.
30.       // Se configuran los puertos GPIO
31.
32.       int ledPin2 = 2; // GPIO2
33.
34.       unsigned int dirorigen = 0x01;              //Direccion del equipo CTRL
35.       unsigned int dirdestino1 = 0x02;            //Direccion del esclavo 1.
36.       unsigned int accion = 0x00;                 //Variable que recoge la accion a re
          alizar
37.       unsigned int accionlecturaCOM = 0x5A;       //Variable contiene el codigo asocia
          do a lectura del COMM
38.       unsigned int accionlecturaI2C = 0xAA;       //Variable contiene el codigo asocia
          do a lectura del I2C
39.
40.       int NumEquipo = 1;      //Direccion de este dispositivo
41.
42.
43.       //Variables para ctrl de comunicaciones
44.       unsigned int msjTX[12];                     //Variable general de trasnmision CO
          MM
45.       unsigned int msjRX[30];                     //Variable general de recepcion COMM

46.
47.       int NCR = 0;              //Contador de caracteres recibidos
48.       int MSG_empezado = 0;
49.       int MensajeRecibido = 0;
50.       int FinMsg = 0;
51.       int LongMsg = 0;
52.
53.       //Variables ver medidas analogicas
54.
```

```
55.       unsigned long MEDIDA1 = 0x0000;
56.       unsigned long MEDIDA2 = 0x0000;
57.       unsigned long MEDIDA3 = 0x0000;
58.
59.
60.       WiFiServer server(80);
61.
62.       void setup() {
63.
64.         //Serial.println(dirdestino1,HEX);  //Representacion de numero en HEXADECIMA
          L
65.         //Serial.println(dirdestino1,BIN);  //Representacion de numero en BINARIO
66.         //Serial.println(dirdestino1);      //Representacion de numero en DECIMAL
67.
68.         //Se configura velocidad PTO COMM
69.         Serial.begin(9600);
70.         delay(10);
71.
72.         //Se configuran los modo funcionamiento salidas GPIO y estado inicial
73.
74.         pinMode(ledPin2, OUTPUT);
75.         digitalWrite(ledPin2, HIGH);
76.
77.         // Se conecta wifi a red lan
78.         Serial.println();
79.         Serial.println();
80.         Serial.print("Conectando a red WIFI: ");
81.         Serial.println(ssid);
82.
83.         WiFi.config(ip, gateway, subnet, DNS); //Comentar para IP DHCP
84.         WiFi.begin(ssid, password);
85.
86.         while (WiFi.status() != WL_CONNECTED) {
87.           delay(500);
88.           Serial.print(".");
89.         }
90.
91.         while (WiFi.waitForConnectResult() != WL_CONNECTED) {
92.           Serial.println();
93.           Serial.println("Fallo de conexion");
94.           delay(5000);
95.           ESP.restart();
96.         }
97.
98.         Serial.println("");
99.         Serial.println("conexion WiFi establecida");
100.
101.        // Arranque servidor web
102.        server.begin();
103.        Serial.println("Servidor WEB arrancado");
104.
105.        // Print the IP address
106.        Serial.print("Direccion IP de conexion: ");
107.        Serial.print("http://");
108.        Serial.print(WiFi.localIP());
109.        Serial.println("/");
110.
111.      }
112.
113.      void loop() {
114.
115.        // Se comprueba si un cliente se ha conectado
116.        WiFiClient client = server.available();
117.        if (!client) {
118.          return;
119.        }
```

```
120.
121.         // Se espera hasta que el cliente envie algun dato
122.         //Serial.println("Nueva conexion de Cliente");
123.         while (!client.available()) {
124.           delay(1);
125.         }
126.
127.         // Read the first line of the request
128.         String request = client.readStringUntil('\r');
129.         //Serial.println(request);
130.         client.flush();
131.
132.         ////////////////////////////////////////////////
133.         // Lectura del boton pulsado y estado del resto
134.         ////////////////////////////////////////////////
135.
136.         int value2 = HIGH;
137.
138.         ////////////////////////////////////////////////////////////
139.         // Para realizar accion asociada al boton LEER MEDIDAS COM
140.         ////////////////////////////////////////////////////////////
141.
142.         if (request.indexOf("/ENVIAMSJ=LEERCOM") != -1)  {
143.           //Serial.println("");
144.           //Serial.println("¡¡¡ AVISO: HA TRANSMITIDO MSJ DESDE WIFI A COMM !!!");
          //Se transmite por el COMM
145.           accion = accionlecturaCOM;
146.           RealizarAccionCOM(accion);
147.           delay(100);
148.           LecturaCOM();
149.           if (MensajeRecibido = 1) {
150.             GestionaMSG();
151.           }
152.         }
153.
154.         ////////////////////////////////////
155.         // Para el boton 2
156.         ////////////////////////////////////
157.
158.         if (request.indexOf("/LED2=ON") != -1)  {
159.           digitalWrite(ledPin2, LOW);
160.           value2 = LOW;
161.           //Serial.println("");
162.           //Serial.println("ESTADO DE SALIDAS DIGITALES");
163.           //Serial.print("Valor LED 2: ");
164.           //Serial.println(value2);
165.         }
166.
167.         if (request.indexOf("/LED2=OFF") != -1)  {
168.           digitalWrite(ledPin2, HIGH);
169.           value2 = HIGH;
170.           //Serial.println("");
171.           //Serial.println("ESTADO DE SALIDAS DIGITALES");
172.           //Serial.print("Valor LED 2: ");
173.           //Serial.println(value2);
174.         }
175.
176.         //Se pone el estado del led seleccionado por el boton pulsado
177.
178.         //Se mandan las cabeceras de establecimiento HTTP
179.         client.println("HTTP/1.1 200 OK");
180.         client.println("Content-Type: text/html");
181.         client.println(""); //  do not forget this one
182.         client.println("<!DOCTYPE HTML>");
183.
184.         //Se comienza la pagina web
```

```
185.         client.println("<html>");
186.         client.println("<head>");
187.         client.println("<title>CTRL ESP8266</title>");
188.
189.         client.println("<script src='https://code.jquery.com/jquery-
         3.2.1.js'></script>");
190.         client.println("<link rel='stylesheet' href='https://maxcdn.bootstrapcdn.com
         /bootstrap/3.3.7/css/bootstrap.min.css' integrity='sha384-
         BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u' crossorigin=
         'anonymous'>");
191.         client.println("<script src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
         /js/bootstrap.min.js' integrity='sha384-
         Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa' crossorigin=
         'anonymous'></script>");
192.
193.         client.println("</head>");
194.         client.println("<body style='background-color:#FFFBBB;'>");
195.         client.println("<center>");
196.
197.         client.println("<div class='container'>");
198.         client.println("<div class='page-
         header'><h2>Ctrol NODEMCU y ESP8266</h2></div>");
199.
200.         ////////////////////////////////////////////////////////////////////////
201.         ////////////////////////////////////////////////////////////////////////
202.         // COMUNICACIONES SERIE    (Se leen 3 medidas analogicas)
203.         ////////////////////////////////////////////////////////////////////////
204.         ////////////////////////////////////////////////////////////////////////
205.
206.         client.println("<div class='panel panel-info'>");
207.         client.println("<div class='panel-heading'><h3 class='panel-
         title'>COMUNICACION SERIE</h3></div>");
208.         client.println("<div class='panel-body'>");
209.
210.         ////////////////////////////////////////////(/////////////
211.         //Se crea botones de ordenes a envia en comunicaciones
212.         ////////////////////////////////////////////(/////////////
213.
214.         if (accion == accionlecturaCOM) {
215.           //client.println("Ultima accion enviada al Pto COM1: ");
216.           client.println("Ultima accion enviada: ");
217.           client.println("0x");
218.           client.println(accion, HEX);
219.           client.println("    -> (PETICION DE LECTURA MEDIDAS PTO SERIE)");
220.           client.println("<br/><br/>");
221.
222.         } else {
223.           client.println("Ultima accion enviada: ");
224.           client.println("<br/><br/>");
225.
226.         }
227.
228.         client.println("<a href=\"/ENVIAMSJ=LEERCOM\"\"><button class='pinbtn btn bt
         n-lg btn-
         primary' style='cursor: pointer'>LEER MEDIDAS COM</button></a><br/><br/>");
229.         client.println("<div class='panel panel-info'>");
230.         client.println("<div class='panel-heading'><h3 class='panel-
         title'>MEDIDAS ENTRADAS ANALOGICAS</h3></div>");
231.         client.println("<div class='panel-body' style='background-
         color:#FFFFFF;'>");
232.
233.         client.print("PERIODO ->  ");
234.         client.print(MEDIDA1);
235.         client.println(" csg</br></br>");
236.
237.         client.print("MINIMO ->  ");
```

```
238.         client.print(MEDIDA2);
239.         client.println(" unidades</br></br>");
240.
241.         client.print("MAXIMO ->  ");
242.         client.print(MEDIDA3);
243.         client.println(" unidades</br>");
244.
245.         client.println("</div>");
246.         client.println("</div>");
247.
248.         client.println("</div>");
249.         client.println("</div>");
250.
251.         ////////////////////////////////////////////////////////////////////////
252.         ////////////////////////////////////////////////////////////////////////
253.         // SALIDAS DIGITALES    (Se activa salida digital por 0)
254.         ////////////////////////////////////////////////////////////////////////
255.         ////////////////////////////////////////////////////////////////////////
256.
257.         client.println("<div class='panel panel-info'>");
258.         client.println("<div class='panel-heading'><h3 class='panel-
          title'>CONTROL SALIDAS DIGITALES</h3></div>");
259.         client.println("<div class='panel-body'>");
260.
261.
262.         ////////////////////////////////////////////////
263.         //Se crea boton 2
264.         ////////////////////////////////////////////////
265.
266.         client.print("Estado Led GPIO 2: ");
267.         if (value2 == HIGH) {
268.           client.print("OFF     ");
269.         } else {
270.           client.print("ON    ");
271.         }
272.
273.         client.println("<a href=\"/LED2=ON\"\"><button class='pinbtn btn btn-lg btn-
          success' style='cursor: pointer'>LED 2 ON </button></a>");
274.         client.println("<a href=\"/LED2=OFF\"\"><button class='pinbtn btn btn-
          lg btn-danger' style='cursor: pointer'>LED 2 OFF </button></a><br/>");
275.
276.         //Se termina diseño pagina web
277.         //client.println("</fieldset>");
278.         //client.println("</center>");
279.
280.         client.println("</div>");
281.         client.println("</div>");
282.         client.println("</div>");
283.         client.println("</center>");
284.
285.         client.println("</body>");
286.         client.println("</html>");
287.
288.         delay(1);
289.         //Serial.println("");
290.         //Serial.println("Cliente Desconectado");
291.         //Serial.println("-------------------------------------------------------------
          -------------------------");
292.         //Serial.println("");
293.
294.       }
295.
296.       ////////////////////////////////////////////////
297.       ////////////////////////////////////////////////
298.       // FUNCIONES
299.       ////////////////////////////////////////////////
```

```
300.      ////////////////////////////////////////////////
301.
302.      ////////////////////////////////////////////////
303.      //Funcion para lanzar comandos a esclavo COMM
304.      ////////////////////////////////////////////////
305.
306.      void RealizarAccionCOM (unsigned int OrdenCodigo) {
307.
308.        // Se conforma cabecera MSJ
309.        msjTX[0] = 0x80;
310.        msjTX[1] = 0x01;
311.        msjTX[2] = 0x02;
312.        msjTX[3] = 0x03;
313.        // Se conforma longitud MSJ
314.        msjTX[4] = 0x03;
315.        // Se conforma accion a realizar MSJ (leer datos accion = 0x19)
316.        msjTX[5] = OrdenCodigo;
317.        // Se conforma direccion origen MSJ (quien pregunta)
318.        msjTX[6] = dirorigen;
319.        // Se conforma direccion destino MSJ (a quien se pregunta)
320.        msjTX[7] = dirdestino1;
321.        // Se conforma CHK alto MSJ
322.        msjTX[8] = 0x00;
323.        // Se conforma CHK bajo MSJ
324.        msjTX[9] = msjTX[5] + msjTX[6] + msjTX[7];
325.        // Se conforma cola MSJ
326.        msjTX[10] = 0xA5;
327.        msjTX[11] = 0xA5;
328.
329.        for (int i = 0; i < 12; i++) {
330.          //Serial.print(msjTX[i],HEX);  //Saca por el COMM el dato en ASCII
331.          Serial.write(msjTX[i]);        //Saca por el COMM el dato en binario
332.        }
333.        Serial.flush();   //Se espera a que termina toda la transmision
334.
335.        return;
336.
337.      }
338.
339.      /*********************************************************************
340.       *********************************************************************
341.        Estructura del mensaje de recepcion
342.        80 01 02 03 09 5A 01 02 03 FF 03 FF 03 FF 0B FD A5 A5
343.
344.        cabecera -> 80 01 02 03 (31 32 33 34)
345.        Longitud MSG -> 09
346.        Accion a realizar -> 5A
347.        Direccion destino -> 01
348.        Direccion origen -> 02
349.
350.        Medida analogica 1 -> 03 FF
351.        Medida analogica 1 -> 03 FF
352.        Medida analogica 1 -> 03 FF
353.
354.        CHK-alto -> 0B
355.        CHK-bajo -> FD
356.
357.        Fin MSG -> A5 A5
358.      *********************************************************************
359.      *********************************************************************/
360.
361.      ////////////////////////////////////////////////////////////
362.      //Funcion para sacar medidas del COMM por pantalla
363.      // La resolucion de las medidas es de 10 bits -> 1023 ctas
364.      ////////////////////////////////////////////////////////////
365.
```

```
366.      void VER_MEDIDAS_COMM () {
367.
368.        byte MED1_HIGH = msjRX[8];
369.        byte MED1_LOW = msjRX[9];
370.
371.        byte MED2_HIGH = msjRX[10];
372.        byte MED2_LOW = msjRX[11];
373.
374.        byte MED3_HIGH = msjRX[12];
375.        byte MED3_LOW = msjRX[13];
376.
377.        MEDIDA1 = MED1_HIGH << 8;
378.        MEDIDA1 = MEDIDA1 + MED1_LOW;
379.
380.        MEDIDA2 = MED2_HIGH << 8;
381.        MEDIDA2 = MEDIDA2 + MED2_LOW;
382.
383.        MEDIDA3 = MED3_HIGH << 8;
384.        MEDIDA3 = MEDIDA3 + MED3_LOW;
385.
386.        return;
387.
388.      }
389.
390.      ////////////////////////////////////////////////////
391.      //Funcion de Reset de comunicaciones
392.      ////////////////////////////////////////////////////
393.
394.      void RSTcomunicaciones () {
395.
396.        NCR = 0;
397.        MSG_empezado = 0;
398.        MensajeRecibido = 0;
399.        FinMsg = 0;
400.        LongMsg = 0;
401.
402.        for (int k = 0; k < 29; k++) {
403.          msjRX[k] = 0x00;
404.        }
405.
406.        return;
407.
408.      }
409.
410.      ////////////////////////////////////////////////////
411.      //Funcion de comprobacion de CHECKSUM
412.      ////////////////////////////////////////////////////
413.
414.      byte CalculaCHK () {
415.
416.        unsigned long CHK = 0x0000;
417.        byte CHK_H = 0x00;
418.        byte CHK_L = 0x00;
419.
420.        byte ResultadoTEST = 0;
421.        byte contador = 0;
422.
423.        while (contador < msjRX[4])    //Se comprueba si hay algo en el buffer de en
          trada
424.        {
425.          CHK = msjRX[5 + contador] + CHK;
426.          contador = contador + 1;
427.        }
428.
429.        CHK_L = CHK;        //Se queda con la parte BAJA del CHK
430.        CHK_H = CHK >> 8;   //Se queda con la parte ALTA del CHK
```

```
431.
432.       if ((CHK_H == msjRX[4 + msjRX[4] + 1]) && (CHK_L == msjRX[4 + msjRX[4] + 2 ]
           )) {
433.           ResultadoTEST = 1;
434.         }
435.         else {
436.           ResultadoTEST = 0;
437.         }
438.
439.         return ResultadoTEST;
440.
441.       }
442.
443.       ////////////////////////////////////////////////////
444.       //Funcion de Gestion de comunicaciones comunicaciones
445.       ////////////////////////////////////////////////////
446.
447.       void GestionaMSG () {
448.
449.         byte Test_MSG = CalculaCHK();
450.
451.         //Serial.print("Resultado CHK= ");
452.         //Serial.println(Test_MSG);
453.
454.         if ((Test_MSG == 1) && (msjRX[6] == NumEquipo)) {
455.
456.           VER_MEDIDAS_COMM();
457.
458.         }
459.
460.         MensajeRecibido = 0;
461.         RSTcomunicaciones();
462.
463.         return;
464.
465.       }
466.
467.       ////////////////////////////////////////////////////
468.       //Funcion de lectura de PTO COMM
469.       // Nota: a medida que se va leyendo se vacia el buffer
470.       ////////////////////////////////////////////////////
471.
472.       void LecturaCOM () {
473.
474.         //Serial.print("Total de caracteres recibidos: ");
475.         //Serial.println(Serial.available());
476.
477.         while (Serial.available() > 0)    //Se comprueba si hay algo en el buffer de
            entrada
478.         {
479.           msjRX[NCR] = (Serial.read());   //Lectura un caracter del PTO serie
480.
481.           if (NCR >= 4) {
482.             if ((msjRX[NCR - 4] == 0x80) && (msjRX[NCR - 3] == 0x01) && (msjRX[NCR -
             2] == 0x02) && (msjRX[NCR - 1] == 0x03)) {
483.               MSG_empezado = 1;
484.               LongMsg = msjRX[NCR];         //En este Byte esta la longitud del mens
           aje
485.               FinMsg = LongMsg + 4;         //2 bytes de CHK y 2 bytes de cola MSG
486.             }
487.
488.             if ((NCR - 4) >= FinMsg) {
489.               MensajeRecibido = 1;
490.             }
491.
492.           }
```

```
493.
494.           NCR = NCR + 1;
495.           if (NCR >= 29) {
496.             RSTcomunicaciones();
497.           }
498.         }
499.
500.         return;
501.
502.       }
```

ANEXO 3. Esquemático CircuitMaker