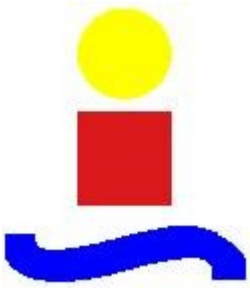


PROYECTO FIN DE CARRERA
INGENIERÍA INDUSTRIAL

Desarrollo de un medidor de distancia ultrasónico con corrección de factores ambientales.

SEVILLA, FEBRERO 2011

Autor: Javier Bermejo López

Tutor: Manuel Perales Esteve

INDICE DEL PROYECTO

1- INTRODUCCIÓN

En el campo de los sistemas activos de medición de distancias nos encontramos fundamentalmente con dos tipos de sistemas, los ópticos y los ultrasónicos. Los sistemas ópticos tienen un alcance muy amplio y son mucho más precisos, debido fundamentalmente a las propiedades de la onda de luz y a su independencia respecto a las condiciones ambientales. Su precio, sin embargo, resulta muy elevado, por lo que en aplicaciones donde no se requiera una precisión o alcance demasiado elevados, o el precio sea un factor determinante, se suele recurrir a los sensores basados en ultrasonidos.

Los sensores ultrasónicos, si bien mucho más económicos, muestran una gran variabilidad respecto a las condiciones ambientales; la razón de ello es que la correcta medición de la distancia dependerá fundamentalmente de lo exacta que sea la estimación de la velocidad del sonido en el medio en el que se opera. En la mayoría de los casos la distancia se obtendrá a partir del tiempo transcurrido entre la emisión de un tren de pulsos ultrasónicos y su recepción después de rebotar contra un obstáculo. Dicho método, conocido comúnmente como *tiempo de vuelo* o *TOF* (time of flight), calcula la distancia entre el sensor y el obstáculo de acuerdo con la ecuación;

$$d = \frac{v_s t}{2} \quad ,$$

donde d es la distancia al obstáculo en metros, v_s es la velocidad de propagación del sonido en m/s y t es el tiempo transcurrido (en segundos) entre la emisión y la recepción del pulso.

Por otro lado, podemos linealizar la dependencia de la velocidad del sonido respecto a los factores ambientales más importantes como la temperatura y la humedad relativa, obteniendo la ecuación: $v_s = 331.58 + 0.62\,T + 0.015\,HR$ donde v_s es la velocidad de propagación del sonido en m/s , T es la temperatura del medio en ^{o}C y HR es la humedad relativa en escala porcentual.

Como se puede observar, la velocidad del sonido dependerá en gran medida de la temperatura y en menor medida de la húmeda relativa; a título estimativo, por ejemplo, entre un ambiente cálido $(36ºC)$ y uno frío $(10ºC)$ tenemos diferencias del 5% en la velocidad del sonido, lo que conlleva un error de la misma magnitud en la estimación de la distancia. Todo esto redunda en la importancia de conocer las condiciones ambientales para poder realizar mediciones lo más exactas posibles.

El objetivo del presente proyecto es el diseño y construcción de un medidor de distancias por ultrasonidos, teniendo un enfoque eminentemente práctico. Nos centraremos en explicar y justificar la solución adoptada y mostrar las posibilidades del sistema de detección desarrollado. Comprobaremos cómo mediante la corrección de los factores ambientales arriba mencionados, y empleando un mínimo de componentes electrónicos para el filtrado, captura y proceso de las señales, conseguimos un sensor de distancia económico, robusto y versátil, que pueda ser usado de manera independiente como instrumento portátil de medición o como sensor en cualquier aplicación de robótica.

Para ello, utilizaremos un microcontrolador de la empresa Microchip, el pic18f4550, que será el encargado del procesado de la señal, la corrección de los factores ambientales y la comunicación con cualquier PC o dispositivo que admita conexión USB o puerto serie.

Dividiremos el proyecto en varios capítulos temáticos.

El capítulo 2 se centra en explicar la física que hay detrás de nuestro sistema de medición por ultrasonidos. Comentaremos brevemente las propiedades de las ondas sonoras y su influencia respecto a las condiciones ambientales, usandouna linealización de la ecuación termodinámica del sonido que será la que empleará la unidad de proceso (microcontrolador). Una vez explicadas las propiedades de los ultrasonidos, explicaremos las propiedades electromecánicas de los transductores ultrasónicos empleados, detallando su alcance, directividad y errores asociados. Ya por último se estudiará la problemática asociada a la medición por ultrasonidos.

El capítulo 3 se centrará en explicar con detalle la solución adoptada, dividiendo el dispositivo de detección en bloques funcionales; emisión, recepción, alimentación, control, comunicación y visualización.

El capítulo 4 explicará el software que gestiona todo el sistema y que está incluido en el microcontrolador, así como el interfaz de usuario que emplea el dispositivo para conectarse con otros periféricos.

El capítulo 5 mostrará los resultados de nuestro sistema de medición sobre el terreno, su robustez frente a diferentes condiciones ambientales así como el consumo de potencia, precisión, rango de funcionamiento y demás características que definen el prototipo.

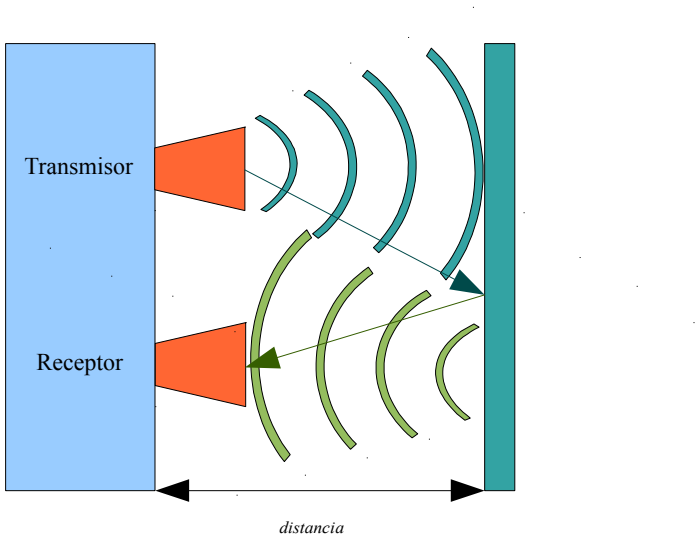
Por último, en los anexos pertenecientes al proyecto incluiremos en detalle los esquemáticos y PCB's de los los circuitos utilizados, así como el código fuente introducido en el microcontrolador. Además, se incluye el código en matlab de una interfaz de usuario para el sensor de ultrasonidos desarrollado, donde se podrá configurar, calibrar y utilizar desde cualquier PC.

2- GENERACIÓN Y PROPAGACIÓN DE ULTRASONIDOS

2.1 - Funcionamiento físico del sensor de ultrasonidos

Los sistemas de ultrasonidos pueden ser utilizados para determinar la distancia a un obstáculo. El funcionamiento básico consiste en la emisión de un tren de ondas a una determinada frecuencia y la captura del haz de ondas de retorno, obteniendo información del entorno a partir de las características de la onda que rebota.

Figura 2.1.1: funcionamiento básico de un sistema de ultrasonidos



Medición de distancia por tiempo de vuelo: se calcula el tiempo que tarda el sonido en recorrer la distancia de ida y vuelta hasta el objeto (trayectorias en azul y verde). La distancia será proporcional al tiempo de vuelo.

Para obtener la distancia entre el obstáculo y el sensor se calculará el tiempo transcurrido entre la emisión del pulso de ultrasonidos por parte del emisor y su recepción por parte del receptor. Esta técnica, conocida como técnica del tiempo de vuelo o TOF (time of flight), calcula la distancia al objeto de acuerdo con la fórmula;

$$d = \frac{t\,V_s}{2}$$

donde t es el tiempo transcurrido entre la transmisión y la recepción y V_s la velocidad estimada del sonido. Será fundamental, por tanto, una correcta estimación de la velocidad del sonido para reducir en la medida de los posible los errores en la medición.

2.2 - Ecuación de ondas y dependencia de la velocidad del sonido respecto a factores ambientales

El sonido se propaga en un fluido en forma de ondas mecánicas longitudinales de presión, comprimiendo y expandiendo el medio a través del cual viaja la onda sonora; la velocidad de propagación del sonido dependerá por tanto de las características del medio, de acuerdo con la ecuación;

$$\nabla^2 p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} \quad ,$$

donde p es la presión del medio en el que se propaga la onda y c es la velocidad de propagación del sonido en el medio. Esta ecuación solamente será derivable para geometrías muy sencillas y medios homogéneos. Además en condiciones reales debemos tener también en cuenta factores como la frecuencia de la onda en medios dispersivos (fundamentalmente debido al CO_2) y los efectos del viento y las turbulencias. Todo ello hace que sea muy difícil calcular la velocidad de propagación, requiriendo herramientas de discretización y cálculo numérico.

En cualquier caso, y dado que el objetivo de nuestro proyecto no es el estudio teórico de las ondas sonoras, de ahora en adelante trabajaremos con un modelo simplificado de esta ecuación, considerando al aire como un medio homogéneo y de comportamiento similar al de un gas perfecto biatómico. Así, obtenemos la ecuación;

$c^2 = \frac{\gamma P}{V \rho}$,donde γ es la relación de calores específicos (1.4 para el aire seco), P es la presión del medio, V el volumen y ρ la densidad del medio.

Sustituyendo en la fórmula la ecuación de los gases perfectos tenemos al fin la ecuación;

$c=\sqrt{\frac{\gamma RT}{M}}$,donde T es la temperatura en Kelvin, R es la constante universal de los gases y M es la masa molecular del gas.

Sin embargo, dado que a efectos prácticos los parámetros que podemos medir directamente mediante sensores serán la temperatura y la humedad relativa y que a efectos de cálculo nos interesa una función lo más sencilla posible y en la que estén reflejadas explícitamente estas magnitudes, recurrimos a una linealización de la anterior ecuación;

$v_s=331.58+0.62T+0.015\,HR$, donde v_s es la velocidad de propagación del sonido en m/s, T es la temperatura en ^{o}C y HR es la humedad relativa en escala porcentual 0-100%.

Tenemos pues una ecuación que relaciona linealmente la temperatura y la humedad relativa con la velocidad de propagación del sonido. Esta linealización será válida para un rango amplio de temperatura y humedad, presentando errores muy pequeños a lo largo de estos rangos. Será en el capítulo 5 donde se compruebe la bondad de esta linealización, así como la variación del comportamiento del sensor y los errores provocados ante diferentes condiciones ambientales.

Otro factor ambiental que podría considerarse sería la altitud sobre el nivel del mar, puesto que afecta a la densidad del medio en el que se propaga el sonido. No obstante dicho factor tiene una influencia de menos del 0.01% sobre la velocidad por lo que no interesa incluir ningún sensor para corregir este error.

El resto de factores ambientales como la lluvia o el régimen de viento tampoco será considerado, dada la enorme dificultad de incluir estos parámetros en la ecuación anterior y al elevado coste de los sensores implicados en relación al aumento de precisión conseguido. Además, más que afectar propiamente a la velocidad de propagación del sonido, afectarán sobre todo a la atenuación de la onda sonora y a la dirección que tome el tren de pulsos; dado que nuestra técnica se basa en calcular el tiempo de vuelo, los factores que afecten a la atenuación de la onda sonora o la distorsión de la trayectoria del tren de ondas no afectarán más que a la distancia máxima a medir o la dirección de la que provenga el eco, respectivamente.

Ya por último nos falta especificar el rango de frecuencias en el que trabajará nuestro medidor de distancias. Como hemos comentado anteriormente nuestro sensor trabajará en el rango de los ultrasonidos, que no son más que aquellas ondas que están por encima del umbral de audición humano, que es de unos 20KHz; concretamente nuestro dispositivo emitirá y recibirá ondas sonoras a 40KHz, frecuencia en la cual todas las ecuaciones y fórmulas arriba mencionadas tienen plena validez. Así pues disponemos de una ecuación en la que se recogen la contribución de los principales factores ambientales a la velocidad de propagación del sonido. Esta será nuestra ecuación de trabajo para el cálculo preciso de la distancia a un objeto mediante el envío y recepción de pulsos ultrasónicos.

2.3 - Transductores ultrasónicos

Los transductores son dispositivos que reciben un determinado tipo de energía a la entrada y la devuelven convertida en energía de otro tipo; los transductores ultrasónicos por tanto transformarán la energía acústica en energía eléctrica o viceversa, siendo los encargados de la recepción o emisión, respectivamente, de las ondas ultrasónicas.

De entre todos los tipos de transductores acústicos existentes, como los magnetorresistivos, electrostáticos, electroacústicos o dinámicos, el tipo más común de transductor ultrasónico es el piezoléctrico, basada en el fenómeno de la piezoelectricidad, por en el que ciertos materiales (fundamentalmente cerámicas) al ser sometidos a una diferencia de potencial sufren tensiones mecánicas. A este fenómeno se le conoce como efecto piezoeléctrico inverso, y será el que se produzca en los emisores de ultrasonidos. Así, en los emisores de ultrasonidos la excitación del mismo por medio de una diferencia de potencial determinada generará vibraciones a esta misma frecuencia, lo que nos permitirá generar señales ultrasónicas.

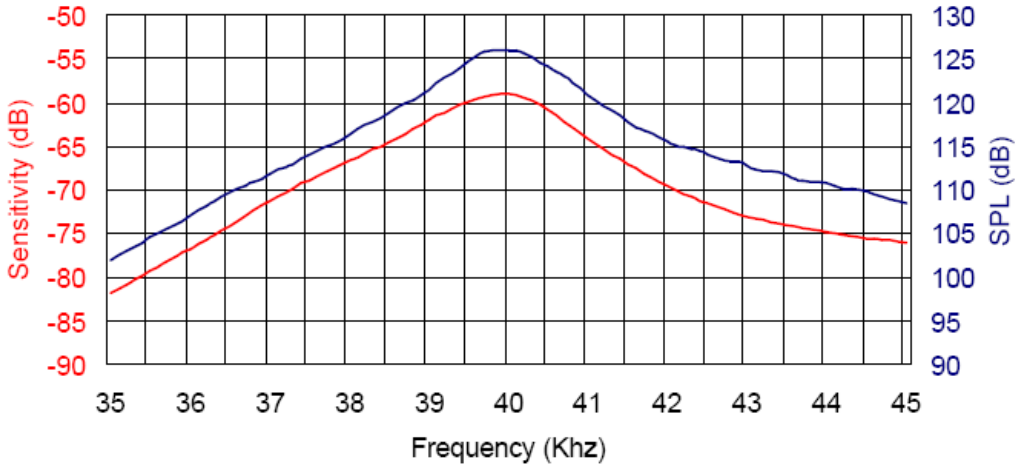
Dicho fenómeno también se produce a de manera inversa en los receptores de ultrasonidos, teniendo lugar el efecto piezoeléctrico directo, en el que las deformaciones mecánicas que se producen en el seno del receptor producen diferencia de potencial en las caras normales a la dirección del esfuerzo.

Un emisor ultrasónico, pues, transforma una señal de tensión en energía mecánica que a su vez genera presión acústica, mientras que un receptor de ultrasonidos recoge la presión sonora en forma de energía mecánica, convirtiéndola en impulsos eléctricos. En función del tipo, tamaño y estructura del material así como de las condiciones de trabajo externas se obtendrán comportamientos diferentes, variando su nivel de presión sonora, sensibilidad, directividad y ancho de banda. Estos serán los parámetros fundamentales que determinan el comportamiento del transductor. Los dos primeros determinarán fundamentalmente el alcance máximo del emisor, mientras que los dos últimos, respectivamente, determinarán el espacio de trabajo del sensor y su robustez ante perturbaciones.

El nivel de presión sonora (Sound Preassure Level o SPL) determina la presión que el emisor ejerce sobre el fluido cuando este es excitado eléctricamente, en relación a un nivel de presión de referencia. Esta referencia se suele seleccionar como la presión que ejercería el emisor sobre el fluido a una distancia de 30cm cuando es excitado mediante una señal de 10Vrms. El nivel de presión sonora decrece con la distancia r a un ritmo de $1/r$, con lo que lógicamente un mayor nivel de presión sonora implica poder transmitir mayor energía acústica al medio, lo que en aplicaciones de proximetría se traduce en aumentar el alcance del sensor. En este tipo de aplicaciones los niveles de presión sonora oscilan entre los 90 y 125dB, teniendo el transductor escogido para este proyecto un SPL de 115dB.

La sensibilidad del receptor indica por su parte cómo reacciona el receptor cuando se ve sometido a ondas ultrasónicas, esto es, cuál será la respuesta eléctrica del receptor ante un nivel de presión sonora determinado en la superficie de la pared transductora. Se expresa en Voltios por unidad de presión, aunque generalmente se suele expresar en dB frente a una referencia de $0\,dB = 1\,V/\mu\,\mathrm{bar}$, e indica cuál será el voltaje de salida correspondiente a la presión sonora mínima que es capaz de percibir. El receptor empleado tiene una sensibilidad de -65dB, estando dentro del rango típico de sensibilidad, que varía entre -55 y -75 dB. Este parámetro es importante puesto que una mayor sensibilidad ante la presión acústica implica consecuentemente una mayor respuesta en forma de voltaje, por lo que permite captar diferencias de presión más pequeñas y por tanto aumentar el umbral de detección del emisor y la resolución, aumentando en consecuencia la distancia de detección.

Figura 2.3.1: Nivel de presión sonora (emisor) y sensibilidad (receptor)



Sensibilidad y nivel de presión sonora (SPL) de una pareja de transductores ultrasónicos Murata, de similares características a los empleados en nuestro sensor de distancia.

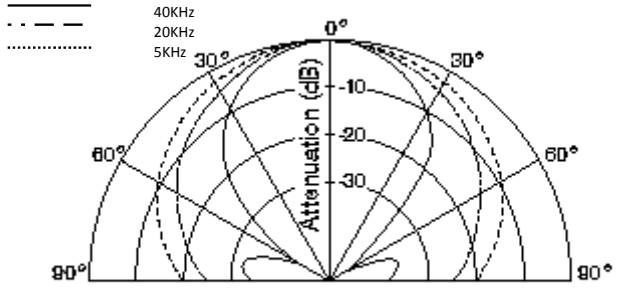
Como puede observarse, tanto el nivel de presión sonora como la sensibilidad dependen fundamentalmente de la frecuencia; como puede deducirse de la gráfica anterior, el comportamiento del transductor es similar al de un filtro de paso banda estrecho; existe un rango muy estrecho de frecuencias para los que el comportamiento como emisor y receptor es óptimo, siendo máximos respectivamente el nivel de presión sonora y la sensibilidad y reduciéndose la ganancia conforme nos alejamos de este valor. La frecuencia de trabajo del transductor se corresponde pues con el máximo de estas curvas, pudiéndose comprobar cómo es imprescindible trabajar con emisores y receptores que operen a la misma frecuencia.

El ancho de banda por otra parte determina cómo de estrecha es esta banda de frecuencias, y se expresa como la banda de frecuencias para la cual la caída de ganancia no supera ½ del valor máximo, o lo que es lo mismo, la zona determinada a derecha e izquierda del valor de frecuencia central en la que la ganancia cae menos de 6dB respecto al valor central. Los sensores ultrasónicos usados en proximetría operan generalmente a 40KHz y tienen un ancho de banda de 2-2.5 KHz, y en consecuencia un factor de calidad $f_o/\Delta f$ entre de 16-30. Por ejemplo, los transductores utilizados, para este proyecto, CEBEK-C7210, trabajan a una frecuencia de 40KHz y tienen un factor de calidad en torno a 20.

La directividad es otro factores importantes a la hora de elegir el sensor adecuado de ultrasonidos, ya que determinará la forma en la que la energía acústica se distribuye en el medio en el caso de los emisores, o como la energía es transferida al mismo en el caso de los receptores. Si los transductores son ominidireccionales emitirán y recibirán energía acústica en todas las direcciones del espacio, mientras que si existe *direccionalidad* existirán direcciones preferentes para la transducción ultrasónica. La directividad depende fundamentalmente de las propiedades geométricas del transductor, fundamentalmente del tamaño de la fuente, y de la frecuencia de trabajo del transductor, de manera que a mayor frecuencia y/o tamaño del transductor mayor será su directividad.

La directividad suele describirse mediante los denominados *diagramas de directividad*, en el que se muestra de manera gráfica cómo la radiación sonora se distribuye en el espacio en torno a la fuente, obteniéndose la respuesta del transductor en función de la dirección de las ondas sonoras en un plano esférico para una frecuencia dada. Estos diagramas muestran cómo varía la presión de la ondas sonoras en decibelios en función del ángulo respecto al eje axial. Presentan típicamente una zona preferente de emisión-recepción en forma de lóbulo orientada en torno al eje axial, y a veces, con directividades elevadas, otros lóbulos secundarios de menor tamaño orientados en ángulos diversos.

Figura 2.3.2: diagramas de directividad para 5, 20 y 40KHz



En el ejemplo de arriba, que corresponde al diagrama de directividad de un sensor comercial Murata, se observa cómo a medida que aumenta la frecuencia aumenta también la directividad, aumentando la atenuación de la radiación sonora para un mismo ángulo. Por ejemplo, cuando el transductor trabaja a 5KHz presenta una atenuación de 10 dB a 60º, lo que

equivale, más o menos, a un tercio del valor en el eje axial. Si el sensor pasa a trabajar a 40 KHz el valor la atenuación aumenta unas 100 veces, siendo prácticamente nulas la emisión y la recepción de ultrasonidos para este ángulo.

En la práctica los transductores ultrasónicos comerciales presentan una directividad elevada ya que interesa que su rango de actuación esté limitado a una zona concreta del espacio. Si su uso está destinado a la obtención de distancias una gran directividad implica en el caso de los emisores reducir la región del espacio en el que se emitirá de manera preferente radiación, concentrando la radiación sonora en un cono de emisión estrecho y perpendicular al emisor, mientras que en el caso de los receptores implica reducir la región del espacio en la que es sensible a la radiación sonora. Lógicamente una directividad elevada implica que el rango de actuación del sensor de ultrasonidos se vea reducido a un cono de emisión-recepción más estrecho, reduciéndose la problemática asociada a las mediciones por ultrasonidos.

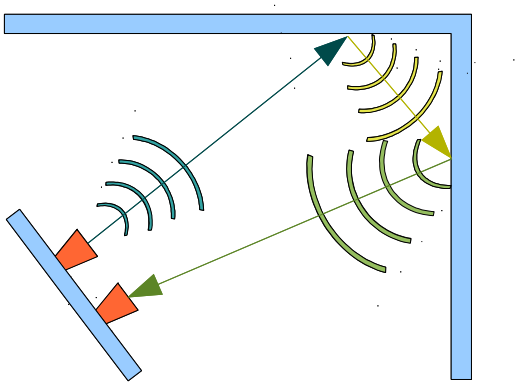
2.4 - Reflexión de los ultrasonidos y problemas asociados a la medición.

Las fuentes emisoras de ultrasonido propagan las ondas sonoras a lo largo de un cono de emisión, propagándose el sonido en linea recta y a velocidad constante en un medio isotrópico. Los ondas sonoras, al chocar con un objeto, absorben una pequeña parte de la radiación y reflejan el resto energía acústica, cumpliendo las leyes de reflexión de las ondas. El porcentaje de energía absorbida dependerá de factores como la rugosidad del material, su densidad y el angulo de incidencia de la onda, aunque suele ser muy pequeño. De hecho la energía acústica se reflejará al incidir contra un objeto que posea irregularidades de tamaño igual a o menor a la longitud de onda de la radiación incidente; en nuestro caso, con ondas de 40KHz, tendremos longitudes de onda menores o iguales a 8.5 milímetros. Ello implica que en la práctica casi cualquier superficie plana reflejará radiación de vuelta al receptor; incluso en superficies completamente planas se reflejará radiación en los bordes de tales superficies.

Esto por un lado supone una ventaja con respecto a los sistemas de detección ópticos, pues en los segundos resulta más problemático recibir el tren de ondas de retorno, debido a la menor reflectividad de la luz frente al sonido. Sin embargo, el hecho de que las ondas de ultrasonidos puedan rebotar sin apenas perder energía en múltiples superficies provoca que en la práctica se puedan recibir en la etapa receptora ecos que no corresponden con el correspondiente a una hipotética trayectoria directa entre el sensor y el objeto. Este fenómeno, conocido como *falsos ecos*, son muy corrientes en este tipo de sensores, y serán especialmente significativos cuando los sensores de ultrasonidos sean usados para la generación de un mapa del entorno a partir de mediciones en múltiples puntos del espacio. Será tarea de la unidad de control la interpretación de los resultados, sea mediante análisis geométrico, métodos probabilísticos, redes neuronales, etc...

En lo que respecta a nuestro dispositivo, orientado principalmente a la medición de distancias entre el sensor y un objeto, los falsos ecos debidos a la reflexión múltiple provocan como principal efecto perjudicial un aumento del tiempo de vuelo del tren de ondas, lo que falsea el cálculo de la distancia al estimarse una distancia mayor que la que realmente es.

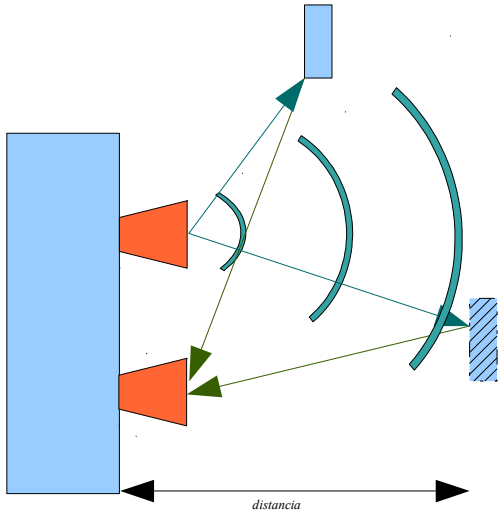
Figura 2.4.1: falsos ecos



Ejemplo de falso eco: cerca de una esquina el tiempo de vuelo medido real (tiempo que tarda el sonido en realizar la trayectoria en azul, amarillo y verde) no corresponde con el teórico (tiempo que tarda el sonido en realizar la trayectoria en azul, multiplicado por dos)

Otra consideración a tener en cuenta es que el sistema de medición por tiempo de vuelo no proporciona una medición de la posición real del objeto; la distancia medida se corresponderá con cualquier posición que esté a una distancia *d* del cono de recepción del sensor. Solo en el caso de que el obstáculo sea una superficie plana perpendicular al haz de radiación sonora, se podrá considerar que la distancia medida por tiempo de vuelo se corresponda con la distancia real al objeto.

Figura 2.4.2: imprecisión al realizar las mediciones



Ejemplo de medición ambigua: para dos objetos a distancias diferentes (en azul normal y sombreado) el sensor asigna una misma distancia, ya que tienen TOF idénticos. La razón de ello es la expansión de la onda sonora en forma de cono al ser emitida y rebotar contra los obstáculos, y a que el sensor es capaz de captar ondas desde diferentes ángulos.

El sensor desarrollado en este proyecto solamente tiene en cuenta el primer eco que recibe el sensor con amplitud suficiente como para considerarlo como ruido. Los siguientes son automáticamente descartados, paliando en parte el problema de los falsos ecos. Lógicamente, aun con esta técnica sigue sin resolverse el problema de discernir si la primera medida es una medida correcta o un falso eco. Queda fuera de consideración además el estudio y la búsqueda de soluciones frente al fenómeno de crosstalk, es decir, aparición de falsos ecos debido al uso simultáneo de múltiples unidades emisoras y receptoras de ultrasonidos; consideramos que tanto

uno como otro problema están fuera del objetivo de este proyecto, quedando relegada la solución de los problemas asociados a este fenómeno a técnicas de corrección por software.

3- DESCRIPCIÓN FÍSICA DEL MEDIDOR DE DISTANCIA

3. 1 - Funcionamiento del dispositivo de medición

El dispositivo emite un tren de pulsos cuadrados de longitud variable a una frecuencia de 40KHz, que son amplificados y emitidos por el emisor ultrasónico al medio. En cuando se han emitido los pulsos se pasa al modo de escucha, a la espera de recibir el eco de retorno de las ondas emitidas La detección del eco se realiza captando las señales de tensión del receptor ultrasónico, tensiones que como son demasiado pequeñas (del orden de 5-20 mVpp), serán amplificadas y filtradas. Para evitar que los ruidos y las perturbaciones falseen los resultados se deberá considerar un umbral mínimo de detección, de tal manera que la unidad de control, encargada de realizar la medida, reciba un “1” lógico únicamente cuando se haya captado un eco con la frecuencia adecuada y la suficiente potencia como para superar el umbral.

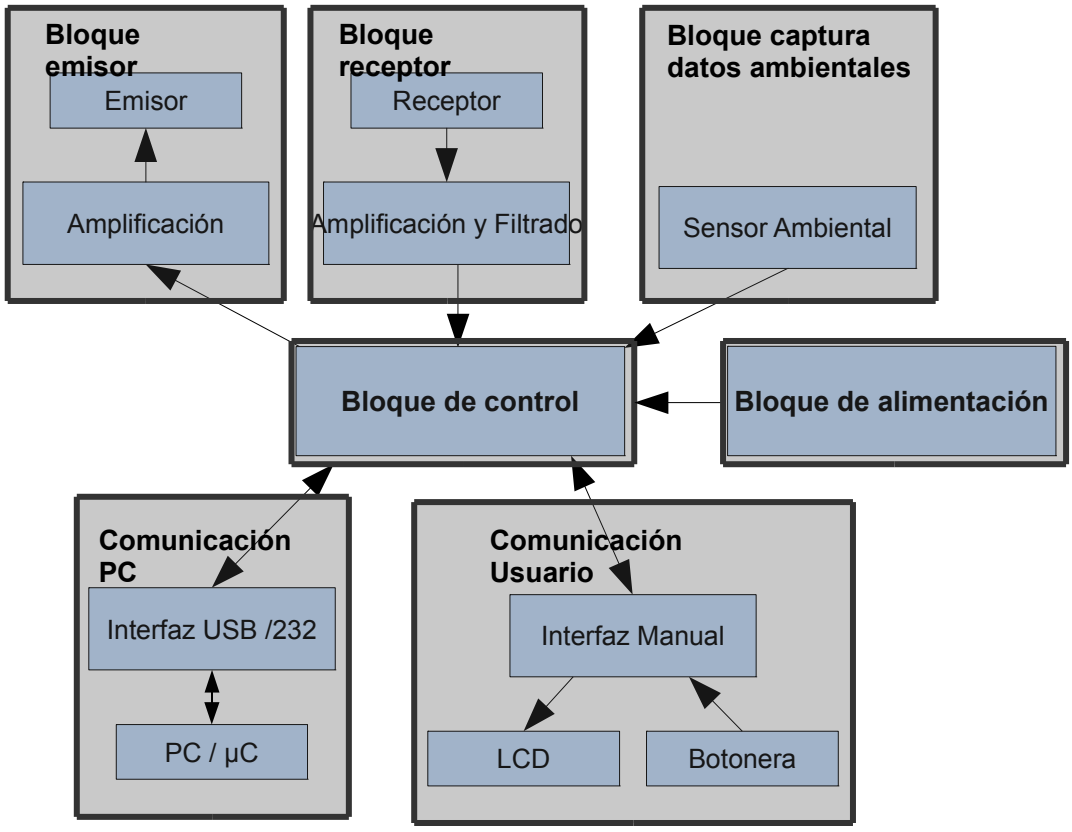
Será la unidad de control la que a partir del tiempo transcurrido entre que ha ordenado la emisión de los pulsos ultrasónicos y la recepción del eco estime la distancia a los objetos mediante la técnica conocida como Tiempo de vuelo o TOF (time of flight). Una vez haya realizado las mediciones correspondientes se encargará de comunicar dicho dato por cualquiera de los interfaces existentes. Dichos interfaces permiten al medidor de distancia comunicarse mediante los protocolos USB o RS-232 con otros dispositivos como PC's o microcontroladores o bien comunicarse directamente con el usuario mediante una pantalla LCD y un conjunto de botones que permitirán su uso como dispositivo autónomo de medición. Esta comunicación no solamente se efectúa para recibir y procesar peticiones de medición de distancia, sino que permite que nuestro medidor pueda ser calibrado, permitiendo configurar parámetros como el retardo asociado a la medición, los tiempos muertos entre la emisión y la recepción o la influencia de los factores ambientales. Además, la unidad de control procurará reducir en la medida de los posible los errores relacionados con el proceso de medición, bien sea mediante técnicas de corrección estadísticas o corrigiendo los errores debido a factores ambientales, fundamentalmente la temperatura.

Para ello la unidad de control monitoriza continuamente la temperatura ambiente gracias a un sensor analógico de temperatura y estima a partir de este valor la velocidad real del sonido. Se pretende con ello obtener un dispositivo que mantenga una tasa de error aceptable bajo diferentes condiciones de trabajo, y pueda usarse tanto en zonas interiores como al exterior.

3.2 – Diagrama de bloques del sistema.

En este capítulo explicaremos en detalle la solución propuesta para el diseño físico del dispositivo. Dado el gran número de componentes e integrados, dividiremos el dispositivo en bloques funcionales, explicando cada uno por separado y las interrelaciones entre ellos. Se estudiará por tanto cada bloque por separado, y se justificará el diseño y características de cada uno, mostrando así mismo el diagrama eléctrico asociado a cada unidad funcional. Podemos considerar pues nuestro sistema como el conjunto de bloques interrelacionados que se muestra en el diagrama adjunto, donde las flechas indican el sentido de flujo de datos o señales.

Figura 3.2.1: diagrama de bloques del medidor de distancias



Una breve descripción de los subsistemas involucrados sería:

- **Bloque Emisor:**

El bloque emisor se encarga de la emisión de las ondas ultrasónicas al medio. Recibe una una señal cuadrada lógica de 40KHz proveniente de la unidad de control, señal que será amplificada y transformada en un tren de pulsos cuadrados de 20Vpp que será enviada al transductor ultrasónico para emitir así así el tren de ondas ultrasónicas.

- **Bloque Receptor:**

El bloque receptor se encarga de la recepción, filtrado y amplificación del eco de retorno de la señal ultrasónica enviada por el bloque emisor. El receptor transforma los ecos de retorno ultrasónicos en señales de pocos mVpp, que serán amplificadas, filtradas y pasadas por un umbral de detección fijo que discrimina entre ecos auténticos y perturbaciones y ruido ambiente. Esta señal umbralizada será enviada a la unidad de control, enviando un “1” lógico si se ha recibido un eco válido, o un 0 en caso contrario.

- **Bloque de Alimentación**

El bloque de alimentación se encarga de proporcionar energía al resto de bloques, bien sea mediante baterías o usando el propio cable de conexión USB en caso de que éste esté conectado. El consumo del dispositivo es de 5 Voltios y cerca de 100 miliamperios, de manera que se podrá elegir entre alimentarlo directamente por USB, o a partir de 4 pilas AAA de 1.5 voltios y un regulador de tensión.

- **Interfaz de Comunicación con el Usuario**

Este bloque se encarga de permitir el uso del dispositivo de manera manual, mediante una botonera y un display de cristal líquido.

- **Interfaz Comunicación PC**

El bloque de comunicación con el PC (o microcontrolador) permite la comunicación del medidor de distancia con cualquier otro aparato que admita conexión tipo USB o RS-232. Para ello incorpora los conectores adecuados para establecer la comunicación así como los interfaces asociados a cada protocolo de comunicación.

- **Bloque de captura de datos ambientales**

Este bloque se encarga de obtener las características termodinámicas del medio en el que se encuentra el medidor. Constará de un sensor de temperatura y otro de humedad, ambos de salida analógica, que será enviada a la unidad de control para estimar correctamente en consecuencia la velocidad del sonido y limitar los errores asociados a la medición por tiempo de vuelo.

- **Unidad de control**

El bloque de control se encargará de coordinar el proceso completo de medición, conectándose en primer lugar con cualquiera de los interfaces disponibles y esperando alguna petición de medición por parte de estos. Si se solicita alguna petición de medición activará el bloque emisor y se situará a la espera de recibir el eco que le envíe el bloque receptor, determinando el tiempo de vuelo. Así mismo, se encarga de calcular la distancia a partir del tiempo de vuelo mediante la información de los sensores ambientales, y enviará dicha información por cualquiera de los interfaces activos, sea por USB, RS-232 o la interfaz manual.

3.3 – Descripción de los subsistemas.

3.3.1. Bloque emisor:

El bloque emisor se encarga del primer paso de la medición por tiempo de vuelo, esto es, la emisión del tren de pulsos ultrasónicos. Para ello emplea un emisor ultrasónico CEBEK C-7210, que tiene las siguientes características;

Especificaciones	Valor
Modelo	CEBEK C-7210
Nivel De Presión Sonora (SPL)	115dBmin (0dB=0.02mPa)
Tensión Máxima Entrada	20Vpp
Intensidad Máxima	25mA
Temperatura de uso	-40ºC a 80ºC
Directividad (caída de 6dB)	80º (cónico)
Frecuencia Nominal	40 ± 1 KHz

De todos estos valores, los parámetros más importantes para diseñar el subsistema emisor son la frecuencia nominal del transductor y las tensiones e intensidades máximas admisibles.

Por un lado se debe procurar que la frecuencia de trabajo se corresponda de manera precisa con la frecuencia nominal del emisor, esto es, 40KHz, ya que como se explicó en el capítulo anterior trabajar con valores alejados de la frecuencia nominal implica una atenuación elevada y por tanto una pérdida global del rendimiento del dispositivo medidor.

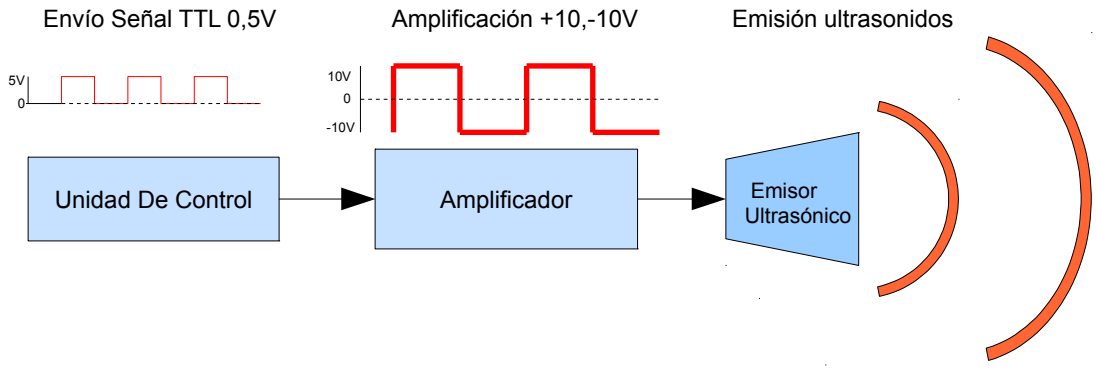
Por otro lado, la tensión máxima admisible a la entrada impone un límite máximo a la tensión de alimentación del transductor de 20Vpp. Dado que la potencia de la energía acústica emitida depende directamente de la tensión de la señal eléctrica con la que se excita el emisor, se debe procurar excitarlo con una señal lo más cercana posible a 20Vpp, con el fin de maximizar el alcance del medidor de distancia.

La intensidad máxima admisible, por último, impone una restricción a la corriente máxima que admite el transductor antes de que se produzca deterioro permanente en el material piezoléctrico.

En resumen, se debía implementar un susbistema que permitiese la creación de un tren variable de pulsos a 40KHz con una tensión lo más cercana posible a 20Vpp, que se usaría para alimentar el emisor de ultrasonidos. La solución adoptada consistió en dejar a cargo de la unidad de control la generación de una señal cuadrada con ciclo de carga de 50% y longitud variable, y de frecuencia 40KHz. De esta manera se garantiza que tanto la frecuencia como el número y ancho de los pulsos sea exacto, así como evitar el tener que generar esta señal mediante componentes electrónicos externos, tipo astables u osciladores, que encarecen el circuito y sobre todo son más proclives a perturbaciones y errores. El único inconveniente es que se obliga al microcontrolador de la Unidad De Control a ocupar ciclos de proceso en la generación de esta onda, si bien no supone apenas una carga de trabajo apreciable, puesto que realizará estas operaciones mediante interrupciones.

Ahora bien, como se ha indicado anteriormente para maximizar el alcance del medidor debíamos excitar el emisor con señales lo más cercanas posibles a los 20Vpp (el máximo permitido por el transductor) por lo que previamente a la etapa de envío de la señal ultrasónica se debía amplificar la señal lógica proveniente del microcontrolador, con una lógica de 5 Voltios, a una señal con tensiones pico a pico de 20V.

Figura 3.3.1: esquema de funcionamiento del subsistema emisor



Funcionamiento del subsistema emisor: la unidad de control envía una señal cuadrada de 40KHz y ciclo de trabajo de 50%, que es primero amplificada a una señal de 20Vpp y que posteriormente se envía al emisor de ultrasonidos que emitirá en consecuencia un tren de pulsos ultrasónicos a la máxima potencia posible.

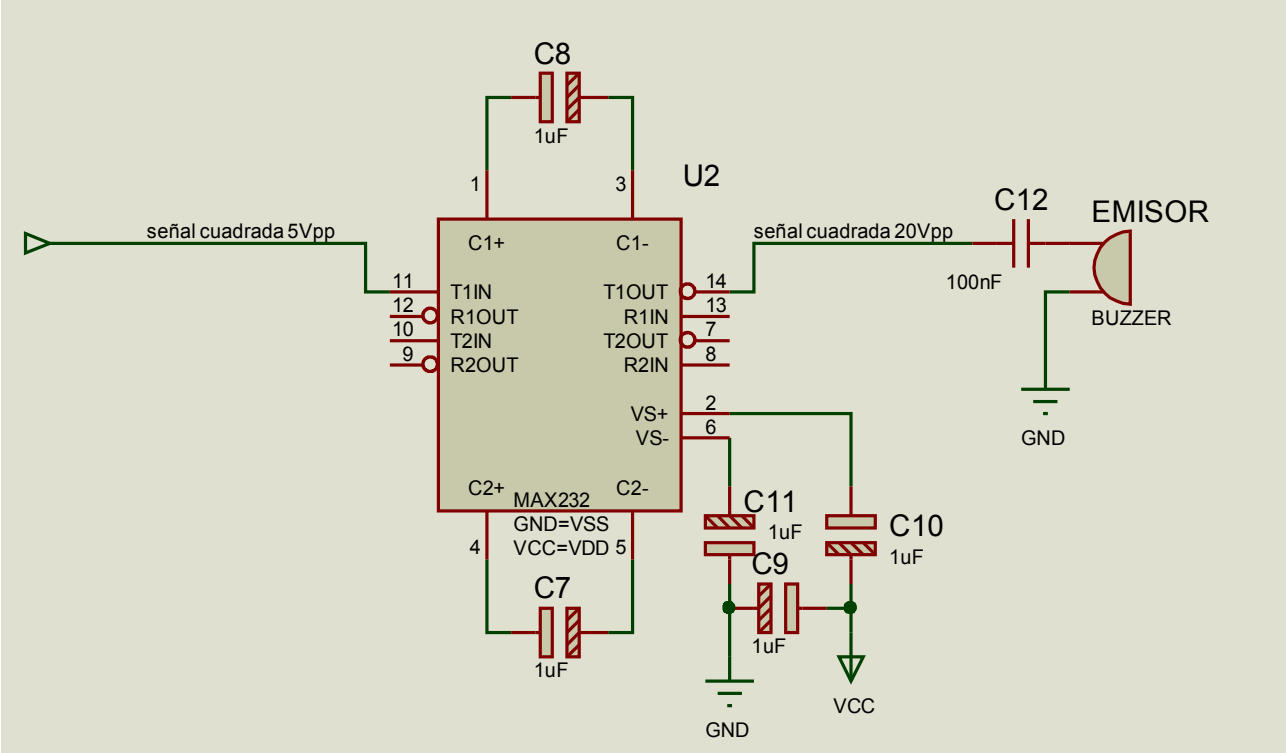
De todas las posibles opciones a nuestro alcance decidimos utilizar un conversor de niveles MAX232N a modo de amplificador de tensión. Realmente el integrado MAX232 se usa para poder comunicar dispositivos que operan con una lógica TTL (usada por los microcontroladores y en concreto por nuestra unidad de control) con otros dispositivos que trabajen con lógica RS232 (Puerto Serie). La razón es que un 0 lógico se corresponde en lógica TTL con una tensión de 0V, o +12 V en lógica 232, mientras que un 1 lógico, que en lógica TTL se corresponde con una tensión de 5V, pasan a ser -12V en lógica RS232. Ello implica que para poder comunicar dispositivos que trabajen con lógicas distintas debemos por tanto adaptar los niveles de tensión. El integrado MAX232N se encarga de esta conversión.

Podemos pues alimentar una de las entradas de este integrado de forma que la señal de 0-5V que genere la unidad de control sea convertida en una señal que trabaje en los niveles de tensión -12, 12V, obteniendo así una señal de 24Vpp. Aunque esta tensión es teóricamente mayor

a la máxima tensión admitida por el transductor, observamos que en la práctica los niveles de tensión de este circuito, cuando se construye según aparece en el esquemático, no alcanzan tensiones superiores a los 20Vpp. No superamos por tanto el límite de tensión máxima admisible a la entrada, pudiendo conectar la salida directamente al transductor. Además, la intensidad que recibe el transductor ultrasónico no supera los 15mA, con lo que estamos por debajo de los límites máximos permitidos. Además una gran ventaja que ofrece este integrado es que nos garantiza un retardo y distorsión de la señal mínimos, pues al estar su uso indicado para transmitir datos a velocidades cercanas a los 120Kbps, puede sin problemas trabajar con una señal cuadrada de baja frecuencia como la nuestra. Su consumo, por último es muy bajo, del orden de los 100mW

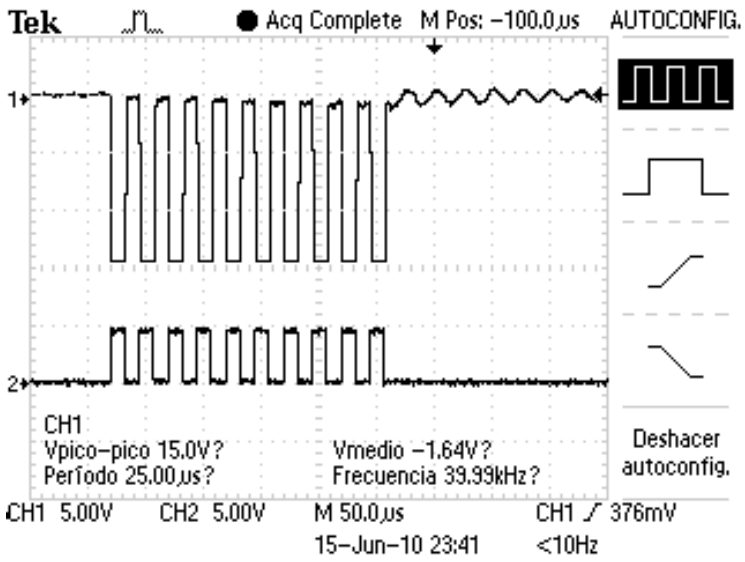
A continuación se muestra el subsistema electrónico del emisor, donde la señal generada por la unidad de control es recibida por el pin 11 del integrado MAX232N y convertida a los nuevos niveles de tensión por el pin 14, conectándose directamente al emisor ultrasónico.

Figura 3.3.2: diagrama esquemático del bloque emisor.



El condensador cerámico puesto en serie a la entrada del emisor (C12) se utiliza a modo de filtro para evitar el paso de corriente ante tensión continua. De esta manera evitamos el desgaste excesivo del transductor, ya que solo permitirá el paso de corriente cuando la señal que reciba el transductor sea una señal alterna, en nuestro caso, una señal cuadrada de 40KHz. El resto del tiempo el transductor estará en reposo. El resto de condensadores (C7-C11) son necesarios para generar y estabilizar los niveles de tensión. A continuación mostramos capturas de pantalla del osciloscopio donde podremos ver la forma de onda obtenida por el microcontrolador y cómo se amplifica por medio del conversor de niveles MAX232N.

Figura 3.3.3: forma de onda del tren de pulsos emisor



Tren de 10 pulsos de 40KHz generadas por el microcontrolador (abajo) y la posterior amplificación gracias al integrado MAX232N (arriba).

Vemos cómo la etapa amplificadora consigue generar un tren de ondas con un retardo inapreciable y conservando la forma de onda, además amplificar la onda unas 4 veces y obteniendo un pulso de ondas cuadradas de 18Vpp. Resulta interesante observar los pequeños picos de tensión que se producen en la etapa emisora tras la generación de los 10 pulsos ultrasónicos provenientes del microcontrolador.

Estos picos de tensión, con una frecuencia de 40 KHz, se deben al efecto de la resonancia del emisor ultrasónicos, de forma tal que se genera un tren de ondas parásito a la misma frecuencia y con valor de tensión pico a pico cercano a los 1.8 Voltios, que no obstante son atenuados rápidamente.

Esta solución por tanto se demostró muy satisfactoria, puesto que con un número pequeño de componentes (el integrado y 6 condensadores de pequeña capacidad) se consigue un circuito amplificador que genera una señal ultrasónica de alta calidad con la máxima potencia posible, usando muy pocos componentes y con un coste muy bajo (menos de 0.5 €). Así mismo solamente requiere de un pin digital configurado como salida digital.

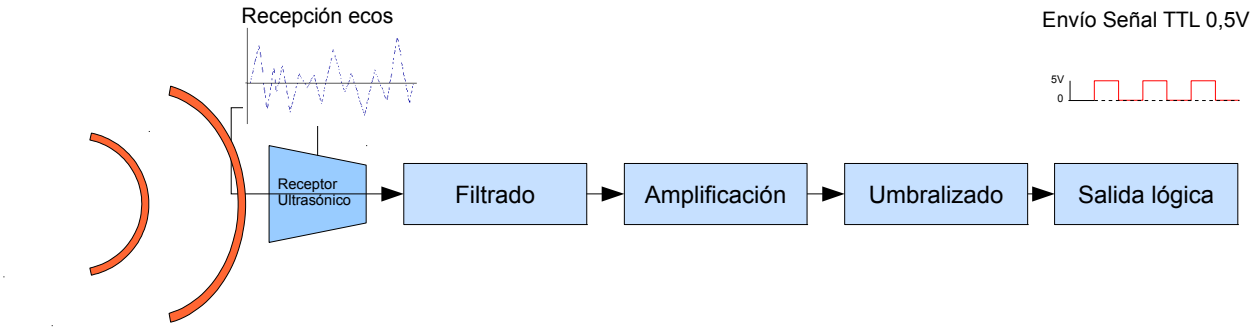
3.3.2 Bloque receptor:

3.3.2.A. Consideraciones iniciales.

El bloque receptor será el encargado de recibir los ecos de retorno que previamente ha emitido el emisor ultrasónico y discernir si realmente corresponden a ecos o son perturbaciones o ruidos. El sistema será capaz de transformar pequeñas variaciones de presión acústica en señales de muy bajo voltaje, que serán amplificadas, filtradas y mediante una técnica de umbralización determinar si corresponden a ecos o a ruidos externos al sistema, a fin de garantizar la estabilidad. Además, la salida de este subsistema estará directamente conectada a uno de los pines digitales de la unidad de control, por lo que resultará imprescindible que devuelva una salida lógica, en la que el 1 (con valores de tensión entre 3.3 y 5V) indique que se ha recibido un eco, y el 0 (con valores de tensión entre 0 y 2.5 V) indicará lo contrario.

Así, el cálculo del tiempo de vuelo por parte de la unidad de control se limitará en emitir el tren de pulsos ultrasónicos a la unidad emisora, activar un temporizador y permanecer a la espera hasta que el pin asociado a la unidad receptora se active. Este sistema estará por tanto dividido en varios subsistemas, los cuales serán el bloque receptor, de filtrado, amplificación y umbralizado, de acuerdo con el siguiente esquema:

Figura 3.3.4: diagrama de bloques de la unidad receptora.



3.3.2.B. Unidad Receptora.

El receptor ultrasónico será el encargado de transformar las señales acústicas en diferencia de potencial en bornas, y cumple las siguientes características:

Especificaciones	Valor
Modelo	CEBEK C-7210
Sensibilidad (SPL)	-64dBmin (0dB=1V/µbar)
Tensión Máxima Entrada	20Vpp
Intensidad Máxima	20mA
Temperatura de uso	-40ºC a 80ºC
Directividad (caída de 6dB)	80º (cónico)
Frecuencia Nominal	40 ± 1 KHz

El receptor deberá escogerse con la mayor sensibilidad posible y con una frecuencia nominal idéntica al del emisor. La sensibilidad determinará el valor de tensión ante un determinado nivel de presión sonoro, por lo que será crítica para poder tener una resolución y alcance elevados, mientras que la frecuencia nominal deberá escogerse idéntica al del emisor de ultrasonidos para evitar la atenuación debido al comportamiento como filtro paso banda del receptor (ver capítulo anterior). En nuestro caso elegimos una pareja emisora-receptora de ultrasonidos, con lo que la frecuencia nominal de ambos será la misma.

3.3.2.C. Etapa amplificadora; primera aproximación al problema.

Dado que nuestro medidor de distancia deberá ser capaz de captar ecos provenientes de objetos pequeños situados a distancias de como mínimo 3m, se deberá amplificar enormemente la señal recibida. Para calcular cuánto debe ser amplificada la señal recurrimos a cálculos teóricos que estimen la pérdida de energía acústica desde que se emite el tren de ondas hasta que el eco de retorno es captado por el receptor, para así dimensionar la etapa amplificadora de modo que compense esta atenuación.

Recurrimos por tanto al cálculo teórico del nivel de presión sonoro en función del nivel de presión sonoro del emisor, las características termodinámicas del medio, la distancia al objeto y la geometría del mismo, de acuerdo con la ecuación siguiente que calcula el nivel de presión del eco de retorno, EL(x) a partir de los parámetros anteriores;

$$EL(X)=SPL(Xo)-40\log_{10}(X/Xo)-2\alpha_f X+TS(X)$$

Consideramos el caso más desfavorable, donde la distancia entre el objeto y el sensor *X* era de más de 4 metros y el objeto era de pequeño tamaño y con una geometría genérica; un buen ejemplo puede ser el suponer una esfera de 6 cm de radio, que equivale a un valor de *TS(x)* de – 36dB. La atenuación debida a la humedad α_f se puede considerar constante y equivalente a – 1.3dB/m y por último el nivel de presión sonora SPL, dato especificado en la hoja de datos de emisor, es de 115dB para un X_o de 30cm. Incluyendo estos valores en la ecuación tenemos al fin, $EL(X)=115\,dB-40\log_{10}(400/30)\,dB-2(-1.3)*4\,dB-36\,dB=15\,dB$

Con estos valores tenemos que el nivel de eco de retorno, EL, tendrá una presión sonora para distancias de 3-4 m de 15dB. Ello implica por tanto que podemos esperar una atenuación máxima de cerca de 100dB entre la emisión y recepción del tren ultrasónico que se tendrá que corregir parcialmente en la etapa amplificadora. Así mismo comprobamos empíricamente que para poder captar ecos de retorno a distancias largas se requería una amplificación de la señal del receptor muy elevada, del orden de los 75 dB. Con estos datos se vio imprescindible dotar al bloque receptor de una etapa amplificadora, del orden de los 75dB, a fin de poder captar ecos de objetos pequeños a largas distancias.

En un primer estudio la implementación de la etapa amplificadora se realizó usando el integrado LM324, un conocido amplificador operacional cuádruple de propósito general, baja potencia y muy bajo coste. Sus especificaciones, aunque modestas, cumplían los requisitos exigidos para actuar como amplificador operacional en el rango de frecuencias y tensiones empleados por el circuito.

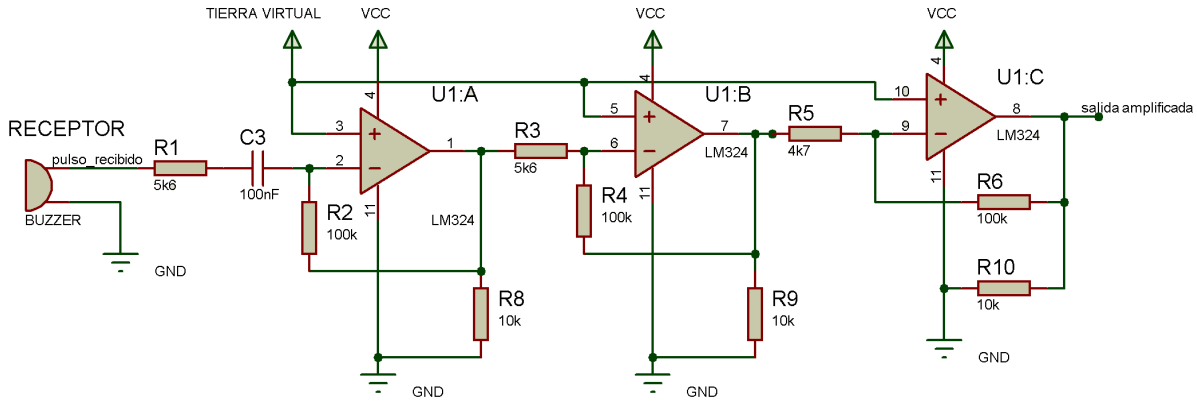
En primer lugar la frecuencia nominal de la señal a amplificar (40KHz) impone un límite a la ganancia máxima que puede otorgar el amplificador; en nuestro caso, nuestro amplificador tiene un producto de ganancia por ancho de banda (GBWP) de 1MHz, y dado que trabajaremos con una frecuencia única e igual a 40KHz, tenemos que la ganancia máxima será,

$$Ganancia = GWBP / f = 1000\,KHz / 40\,KHz = 25 \quad ,$$

con lo que como máximo cada amplificador operacional del LM324N nos puede amplificar la señal unas 25 veces, o lo que es lo mismo, unos 28dB. Como la ganancia total debe ser de cerca de 75 dB, se hará necesario por tanto utilizar 3 de los 4 amplificadores operacionales para aumentar la ganancia, quedando el último amplificador libre para ser utilizado como comparador Schmitt-Trigger e implementar así el proceso de umbralizado.

Un posible esquema eléctrico del amplificador sería el siguiente:

Figura 3.3.5: diagrama esquemático del subsistema amplificador.



En este circuito cada uno de los amplificadores operacionales actúa como amplificador en modo inversor, cumpliéndose para cada A.O. la siguiente relación,

$V_{out} = -V_{\text{in}} \frac{R_f}{R_{\text{in}}}$, que determina el valor de tensión de salida de cada amplificador operacional en función del valor de entrada y de las resistencias .

Aplicando esta ecuación a nuestro circuito tenemos lo siguiente:

- Ganancia en el amplificador operacional A: $R2/R1 = 100/5.6 = 17.86 V/V = 25.0\text{dB}$
- Ganancia en el amplificador operacional B: $R4/R3 = 100/5.6 = 17.86 V/V = 25.0\text{dB}$
- Ganancia en el amplificador operacional C: $R6/R5 = 100/4.7 = 21.28\text{V}/V = 26.6\text{dB}$

La ganancia total, por tanto, será de 76.6dB.

3.3.2.D. Necesidad de la etapa de filtrado.

Ahora bien, cuando se empleó el integrado LM324N como amplificador inversor, se observó que aparece un componente de ruido muy elevado; ello es debido por un lado a la elevada amplificación que se realiza de la señal original del receptor y por otro a que este integrado es relativamente sensible al ruido. La solución que se adoptó para solucionar este problema fue usar dos de los 3 amplificadores operacionales para implementar un filtro activo de paso banda estrecho y mantener el tercer A.O. en modo amplificador inversor a fin de aumentar aun más la ganancia.

La asignación de la ganancia total del filtro paso banda representaba una cuestión interesante, ya que por un lado la ganancia total de las 3 etapas amplificadoras se habían fijado cercanas a los 75dB, y por otra parte la ganancia máxima que otorga cada A.O. Era inferior a los 28dB. Ello en principio nos dejaba cierto margen para asignar la ganancia a cada una de las 3 etapas, aunque empíricamente se comprobó que no era aconsejable trabajar con ganancias cercanas al límite máximo permitido, ya que aumentaban enormemente los errores y la distorsión de la señal.

Ello implicaba en definitiva que la solución ideal pasaba por asignar la misma ganancia por etapa a cada amplificador operacional, lo que nos dejaba nuestro filtro paso-banda con una ganancia cercana a los 50dB, al estar constituido por dos amplificadores operacionales que otorgarán cada uno una ganancia de 25dB. De esta manera reducíamos el problema de aparición de distorsiones y errores y garantizábamos una respuesta similar en cada una de las 3 etapas de la amplificación.

La elección del orden del filtro se demostró evidente, puesto que por un lado interesaba usar el filtro activo de mayor orden que se pudiera implementar con dos A.O. usando el menor número de componentes y sin recurrir a elementos como bobinas, que encarecerían el precio del aparato. Por ello, la solución ideal fue utilizar un filtro de 4º orden mediante el diseño de dos filtros de 2º orden en cascada, uno por cada A.O.

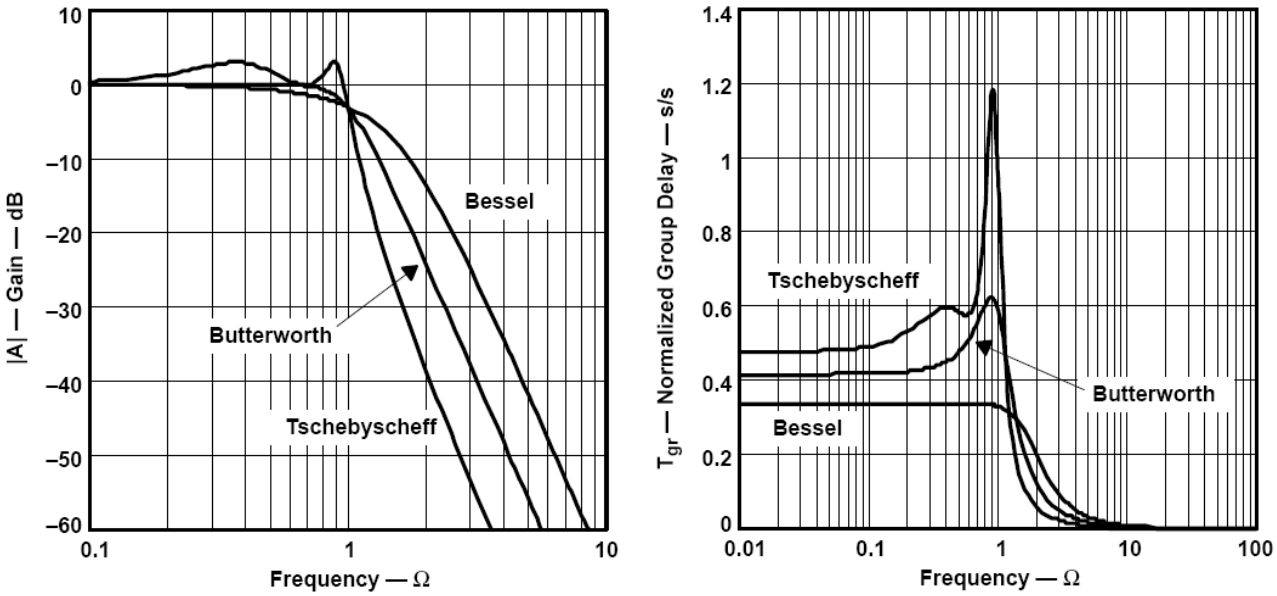
Por otro lado, la frecuencia de corte debía lógicamente ser idéntica a la frecuencia nominal del receptor, por lo que se estableció en 40KHz, mientras que el ancho de banda se tomó con una amplitud relativamente elevada, a fin de que la tolerancia de la frecuencia nominal del transductor (del orden de ± 1 KHz) no afectase sobremanera al filtro. Además así se permitía cierto margen para desviaciones en la frecuencia central del filtro debidas a la temperatura y sobre todo a la imposibilidad de obtener los valores exactos de las resistencias y capacitores que exige el modelo teórico.

Hay que recordar que aun más importante que mantener una ganancia constante está el hecho de evitar que debido a los factores externos comentados anteriormente la frecuencia central se aleje de la banda de paso del filtro y en consecuencia no se capte el eco que reciba el dispositivo; este problema fue estudiado en profundidad, y se consideró que con un ancho de banda de 8.4KHz se podían evitar las desviaciones anteriormente descritas, sin dejar demasiado margen a que se amplifiquen ruidos a diferente frecuencia.

3.3.2.E. Diseño del filtro

La elección del tipo de filtro representaba una decisión difícil; a partir de la respuesta en fase y frecuencia que nos otorga cada filtro, había que seleccionar el que mejor sirviese a nuestros propósitos. Esta viene a ser la respuesta en fase y en frecuencia normalizadas para los filtros de 4º orden candidatos:

Figura 3.3.6: respuesta normalizada en fase (izquierda) y ganancia (derecha)



- El filtro de Chebyschev es el que podía darnos una transición más abrupta en las proximidades de la frecuencia de corte, aunque se caracterizan por presentar un rizado considerable en los extremos de la banda de paso. Por ese motivo decidimos descartarlo como filtro a utilizar, ya que presentaba variaciones importantes de magnitud en el ancho de banda seleccionado, lo cual, unido a la enorme ganancia total del filtro, presentaba problemas de saturación de la señal.

- El filtro de Butterworth presentaba la ventaja de una respuesta plana o cuasi-plana en todo el ancho de banda seleccionado, unido a una transición relativamente abrupta en la zona de corte.

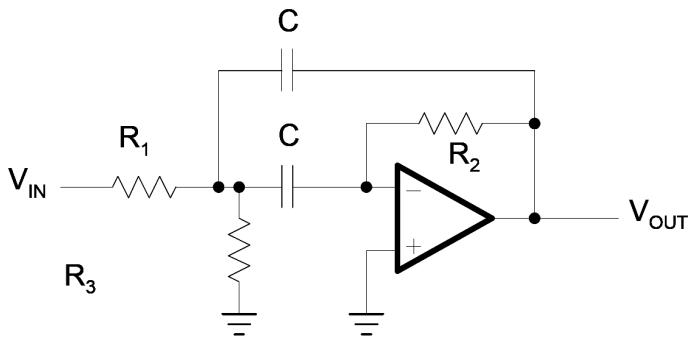
- Por último, el filtro de Bessel, si bien presenta una respuesta de ganancia en la banda de paso no tan plana como la de Butterworth, y una transición algo menos abrupta, presenta la interesante propiedad de tener un retardo de fase constante para todo el ancho de banda, además de ser el filtro que mejor responde ante una señal impulsional.

Al final nos decidimos por emplear un filtro de Bessel, ya que no nos pareció un factor decisivo la ganancia plana en frecuencia ni la diferencia en lo abrupto de la transición entre la zona de paso y la zona de corte, amén de que resultaba especialmente interesante el hecho de conseguir un retardo lineal en la fase y que no se distorsionara la forma de onda. Además, y más importante, este filtro es el que mejor respuesta en fase y ganancia ofrece frente ante una señal de tipo pulso, situación que se dará con muy alta probabilidad, pues la señal amplificar serán los eco de un trenes de 8-20 pulsos ultrasónicos, y generalmente la atenuación y distorsión debida a los rebotes y el paso por el medio hace que de manera efectiva lleguen menos de la mitad de dichos pulsos. En consecuencia al final se decidió implementar un filtro paso-banda que cumpliera los siguientes requisitos:

Especificaciones Del filtro Paso Banda	Valor
Tipo De Filtro	Bessel
Orden Del Filtro	4º Orden (4 polos)
Frecuencia Central	~40 ± 1 KHz
Ancho De Banda (caída de 3dB)	~8.4 KHz
Ganancia Total	~350 V/V || 50dB

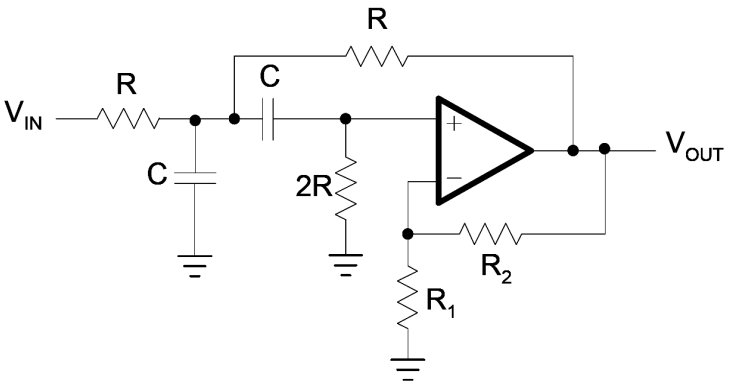
La última cuestión que se nos planteó fue decidir cuál de las diferentes topologías de filtro debíamos emplear para diseñar el esquema eléctrico del filtro de entre las dos topologías más comunes como son la Sallen-Key o la MFB (multiple feed-back).

- La topología Multiple Feedback o Rauch permite implementar un filtro de segundo orden utilizando 2 condensadores y 3 resistencias, de acuerdo con la siguiente configuración:



Y posee como principal ventaja una baja sensibilidad ante variaciones de los componentes, además de ser el filtro más adecuado cuando el factor de ganancia y el de calidad sean altos.

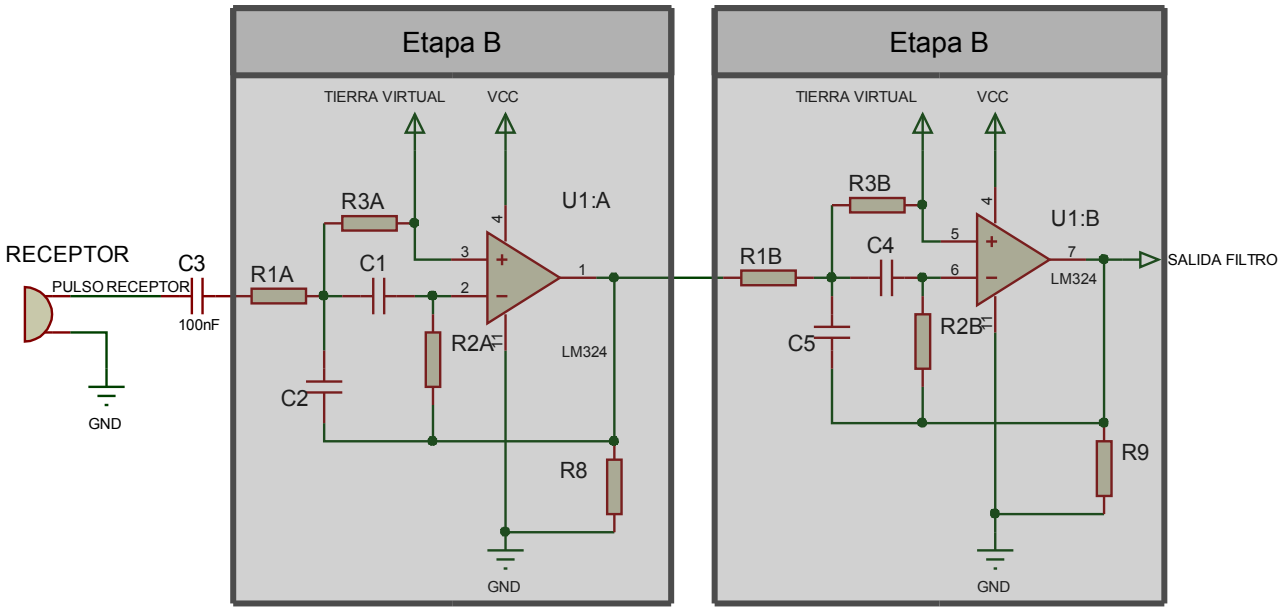
- La topología Salen Key permite por su parte implementar un filtro de segundo orden con un solo operacional usando solamente 2 condensadores y 5 resistencias, de acuerdo a una configuración como la siguiente:



Esta topología resulta la más adecuada cuando se requiere un valor de ganancia muy preciso y cuando el filtro tenga un factor de calidad bajo (Q<3) y una ganancia cercana a la unidad.

Una vez considerados los pros y contras de cada topología nos decantamos por emplear la topología ***Multiple Feed Back*** (MFB), fundamentalmente debido a la alta ganancia y alto factor de calidad que permite el filtro y para garantizar una sensibilidad pequeña ante las tolerancias de los componentes. El esquema del circuito resultante es:

Figura 3.3.7: esquema eléctrico del filtro paso banda



3.3.2.F. Implementación del filtro

Resultó algo complicado encontrar los componentes estándar que mejor implementaran el filtro, ya se debía conseguir una combinación de componentes que cumpliese con sus especificaciones y que además tuvieran en cuenta ciertas consideraciones en cuanto a criterios de consumo, respuesta e inmunidad al ruido. Por ejemplo, en el caso de las resistencias se debían seleccionar valores comprendidos entre los 1kΩ y los 100kΩ; resistencias con valores muy bajos implican un alto consumo de potencia al provocar la circulación de demasiada corriente , mientras que resistencias demasiado elevadas añaden demasiado ruido al sistema, ya que la relación entre el ruido inherente al sistema y la magnitud de las intensidades asociadas sería demasiado pequeña. En el caso de los capacitores, por otro lado, se debía escoger un valor que estuviese comprendido entre los 100pF y los 100nF, para evitar respectivamente demasiada sensibilidad ante el ruido y una respuesta ante cambios de señal demasiado lenta.

Los componentes que mejor se ajustaban a todas estas condiciones fueron:

Figura 3.3.8: lista de componentes del filtro paso banda

Componentes	Etapa A	Etapa B
R1	2.7kΩ	2.2kΩ
R2	100kΩ	82kΩ
R3	3.3kΩ	2.2kΩ
C	330pF	330p

Las funciones de transferencia asociadas a cada filtro son las siguientes;

- Filtro A:

$$F_A(s)=\frac{-R2\,R3\,C\,w_{1n}}{(R1+R3)\left(\frac{R1\,R2\,R3\,C^2 w_{1n}^2}{R1+R3}+2s\frac{C\,w_{1n}\,R1\,R3}{R1+R3}+s^2\right)} \quad \text{, donde} \quad w_{1n}=\frac{1}{C}\sqrt{\frac{R1+R3}{R1*R2*R3}}$$

- Filtro B:

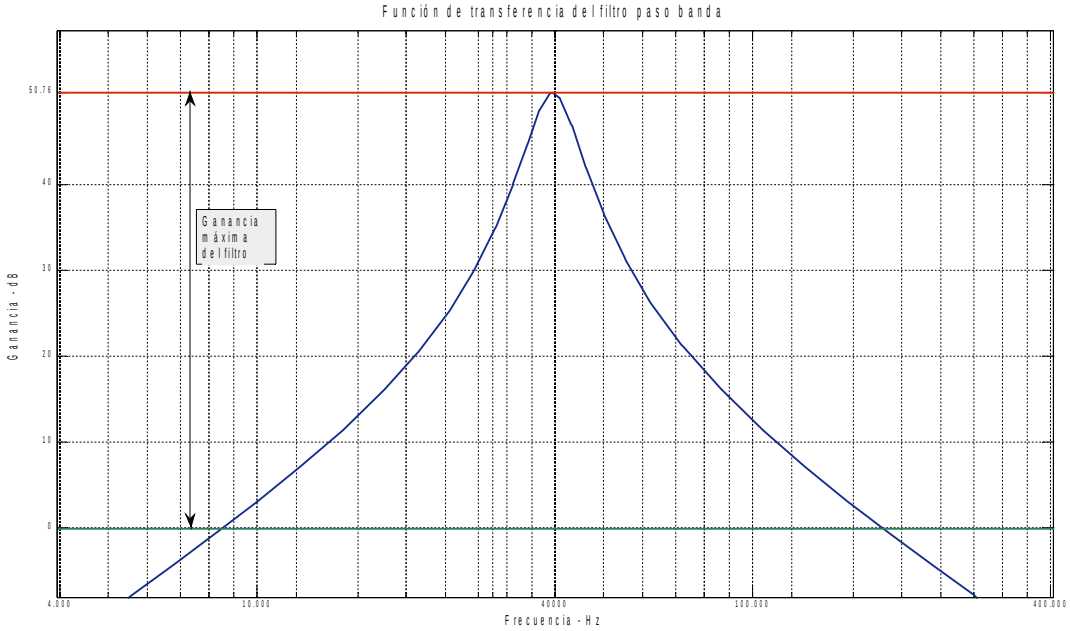
$$F_B(s)=\frac{-R5\,R6\,C\,w_{2n}}{(R4+R6)\left(\frac{R4\,R5\,R6\,C^2 w_{2n}^2}{R4+R6}+2s\frac{C\,w_{2n}\,R4\,R6}{R4+R6}+s^2\right)} \quad \text{, donde} \quad w_{2n}=\frac{1}{C}\sqrt{\frac{R4+R6}{R4*R5*R6}}$$

Uniendo ambos filtros y sustituyendo los valores de los componentes, tenemos;

$$F_{Total}=F_A(s)*F_B(s)=\frac{19.48\,s^2}{s^4+0.4754\,S^3+2.056\,s^2+0.4754\,s+1}$$

cuya gráfica la podemos ver a continuación;

Figura 3.3.9 diagrama de respuesta en frecuencia de la etapa de filtrado

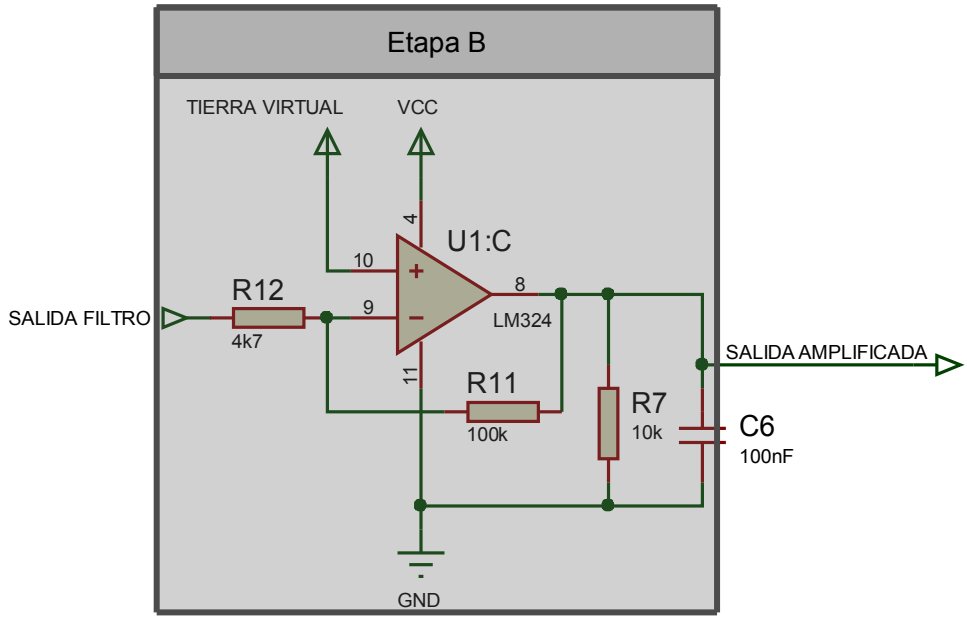


Podemos observar en el diagrama la ganancia total del filtro, que es de 50.76 dB, muy cercano al valor de ganancia elegido de 50dB. Así mismo se observa una pequeña desviación de la frecuencia central del filtro respecto a la ideal; si la frecuencia ideal era la frecuencia nominal del dispositivo, unos 40KHz, observamos que a la hora de implementar el filtro con valores reales está se desvía ligeramente respecto del valor teórico, siendo la frecuencia real 39.5 KHz, que en cualquier caso está comprendida dentro de la tolerancia asociada a la unidad receptora. Conseguimos en definitiva una buena solución al problema del filtrado, empleando 2 de los 4 amplificadores operacionales en implementar un filtro paso banda que además nos amplifique la señal.

3.3.2.G. Etapa final de amplificación

Cuando abordamos el problema de la amplificación consideramos que la mejor opción era reservar uno de los amplificadores operacionales para implementar una etapa de amplificación de la señal de eco. De esta manera, la señal de tensión que sale del filtro es nuevamente amplificada por medio de un A. O. en modo inversor, de acuerdo con el siguiente esquema eléctrico;

Figura 3.3.10 diagrama eléctrico de la última etapa de amplificación



Para conseguir una amplificación cercana a los 25 dB consideramos que la mejor opción era recurrir a emplear dos resistencias de 100k y 4.7k, que amplifican la señal por un factor R11/R12 = 100/4.7 =21.28 V/V = 26.6 dB. Así, la suma de los efectos del filtro y el amplificador se traducen en una ganancia total de 50.74 + 26.6 dB = 77.34 dB.

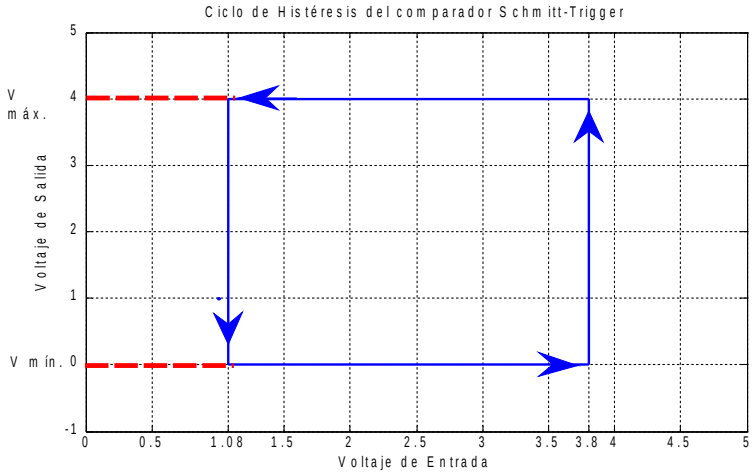
3.3.2.H. Etapa de umbralizado

Una vez se consiguió diseñar una etapa adecuada de amplificación y filtrado, faltaba diseñar una última etapa que se encargase de distinguir entre ruidos ajenos al sistema y los ecos de retorno. Pensamos que la solución más efectiva y más simple consistía en fijar un umbral de tensión fijo a partir del cual las señales que los superen pasarían a ser consideradas como ecos de nuestro dispositivo, y devuelva un 1 lógico en caso afirmativo.

Un estudio de la respuesta ante ecos a diversa distancia nos dio que para que el dispositivo detecte objetos a distancias cercanas a los 3 metros bastaba con que la señal tratada tuviera una amplitud de 1.35 voltios pico a pico; cualquier valor superior a este debía activar una 1 lógico. El uso de la tierra virtual hace que las señales de salida de los amplificadores tengan su referencia puesta a 2.5 voltios, de modo que siempre que un eco supere los 3.8 voltios, la salida pasará a tomar un 1 lógico.

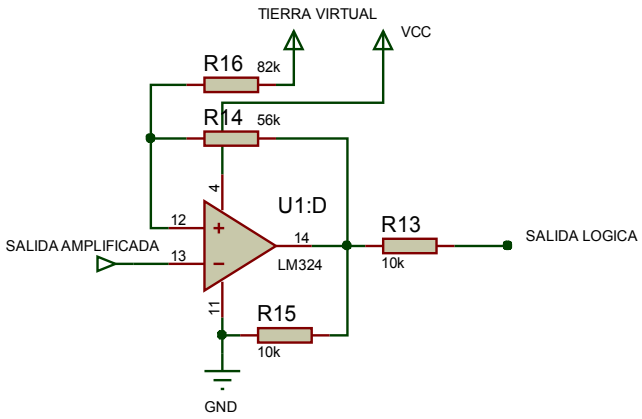
Además tuvimos en cuenta el carácter ruidoso de las señales con las que trabajaba, de manera que optamos por implementar un comparador con ciclo de histéresis con el fin de que el cambio de estado no se produzca siempre que se supera o se baje de este umbral de tensión, sino que para que la salida vuelva a valer 0 el voltaje de la señal de eco debía ser inferior a los 1.1 voltios. Con estos datos decidimos diseñar un comparador con el siguiente ciclo de histéresis;

Figura 3.3.11 ciclo de histéresis de la etapa de umbralizado



El circuito que implementa este ciclo de histéresis es el siguiente;

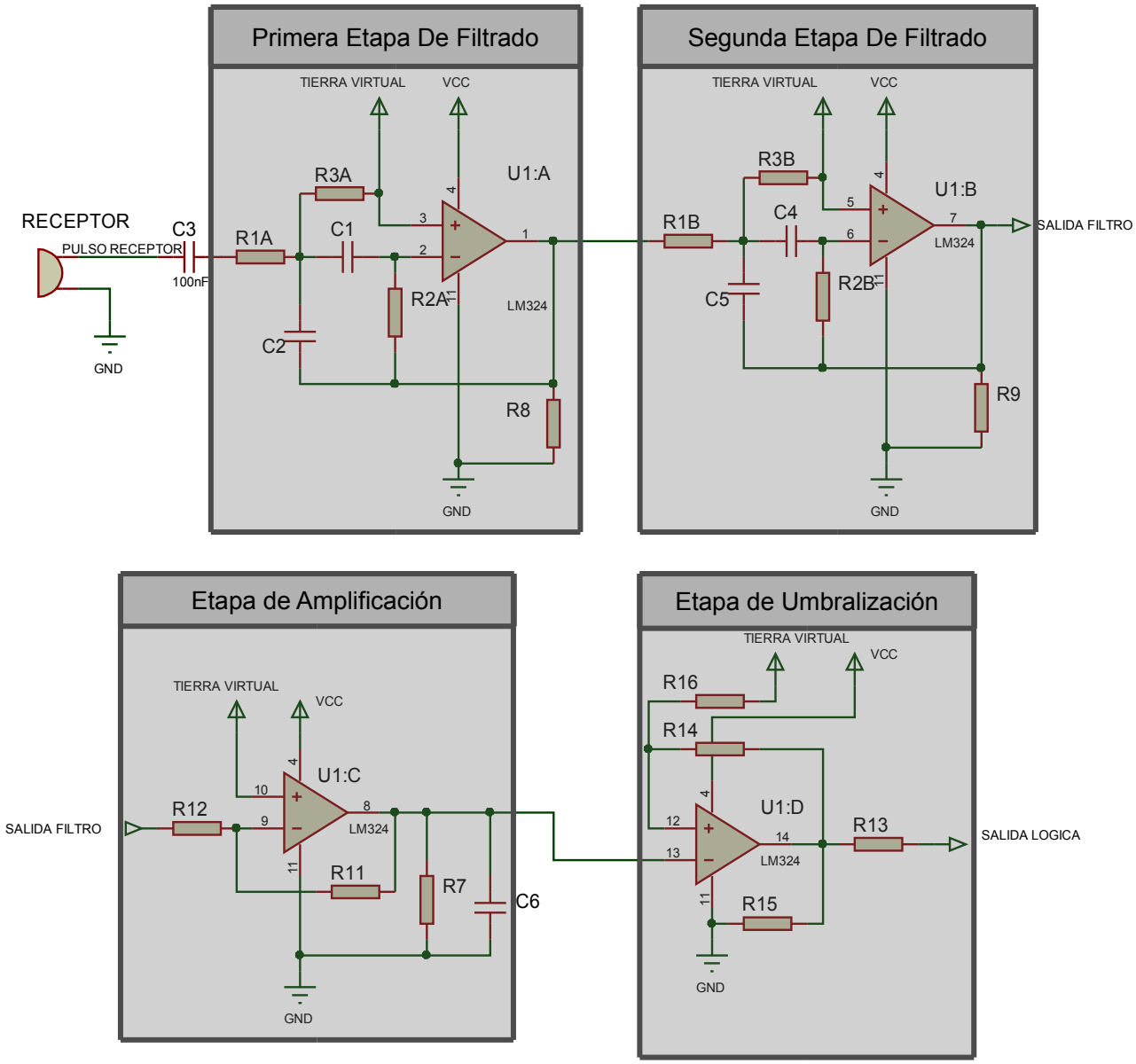
Figura 3.3.12 diagrama eléctrico del comparador Schmitt-Trigger



3.3.2.I. Esquema completo

El sistema de recepción, en definitiva, tiene el esquema siguiente;

Figura 3.3.13 diagrama eléctrico completo del sistema del bloque receptor



Existen ciertos aspectos que merecen la pena destacar acerca de la solución adoptada:

En primer lugar, debemos explicar por qué se ha puesto un condensador en serie entre la salida del receptor y la entrada a la primera etapa del filtro (condensador C3, de 100nF). Su objetivo es el de actuar como capacitor de desacoplo, de forma que actúe como filtro de corriente continua; de esta manera, al amplificador operacional solamente le llegará una señal de corriente alterna que será la que amplifique. Elegir el valor de este condensador no fue un asunto crítico, limitándose a seleccionar uno que estuviera en el rango de los 100 pF a los 100nF.

En segundo lugar, resulta interesante señalar la función del condensador situado en paralelo entre la salida de la etapa amplificadora y tierra (condensador C6, de 100nF). En un principio durante las primeras pruebas realizadas se observó que cuando el dispositivo debía realizar mediciones a objetos situados a escasa distancia, menores de 35 cm, se entraba en resonancia y las mediciones se volvían completamente erróneas. La razón de la resonancia es debido a que la etapa emisora y receptora trabajan ambas a la misma frecuencia nominal, de modo que la activación del pulso ultrasónico saturaba el subsistema receptor.

Este problema se vio solucionado añadiendo el condensador justo antes de la etapa de umbralizado, para disminuir así el tiempo de resonancia. Con ello hemos conseguido que la distancia mínima de detección se reduzca a unos 5 cm, una gran mejora y un valor próximo al que manejan los sensores comerciales.

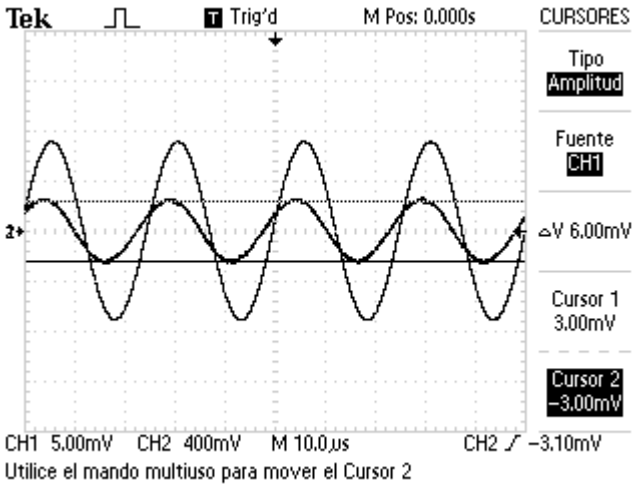
Por ultimo resta por explicar qué función realizan las resistencias R7, R8, R9 y R15 puestas en paralelo entre tierra y la salida de cada amplificador operacional; su función es la de actuar como resistencias de pull-down para garantizar la estabilidad de la salida y de tener una referencia de tierra cuando no exista tensión de salida.

3.3.2.J. Resultados del osciloscopio

Procedemos ahora a mostrar capturas de pantalla del osciloscopio, donde veremos cómo actúa el filtro amplificador así como la fase de umbralizado.

Mostramos en primer lugar cómo actúa la etapa de filtrado y amplificación, que recordemos se diseñó para obtener un filtrado de la señal en base a un filtro de Bessel de 4º orden y centrado en una frecuencia central de 39.7 KHz, consiguiendo además una amplificación de la de la señal de 50.76dB, o lo que es lo mismo, una amplificación de la señal recibida por el receptor ultrasónico de cerca de 345 veces el valor inicial.

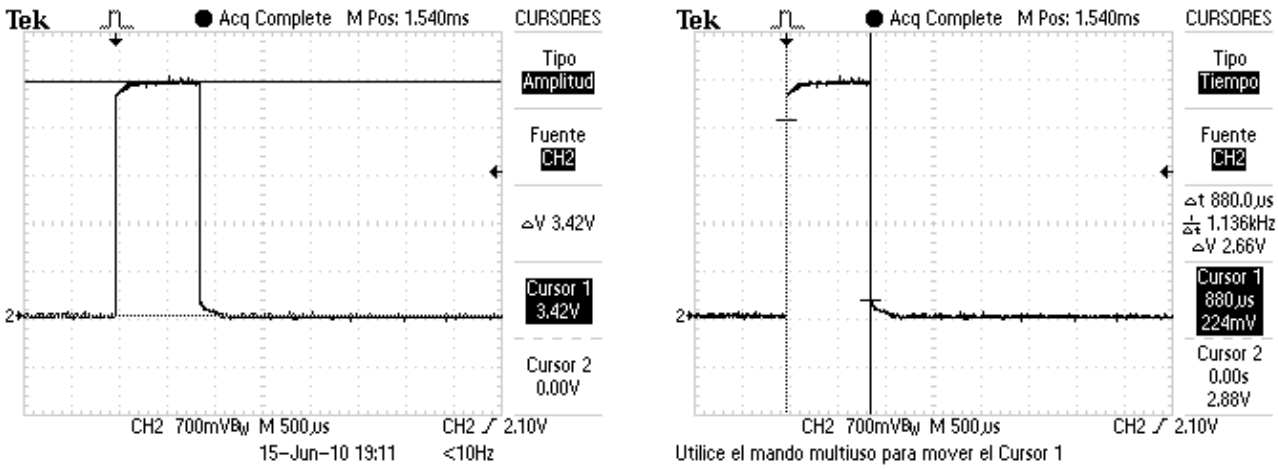
Figura 3.3.14 etapa de filtrado + amplificación



Podemos observar aquí una señal proveniente del receptor ultrasónico de 6mVpp, y cómo se produce la amplificación hasta alcanzar una amplitud de 1440mVpp; ello se traduce en una amplificación del orden de 240V/V, o lo que es lo mismo, unos 47.6 dB, cercano al valor teórico de 50.76dB. Hay que recordar que el amplificador que se está usando corresponde un modelo de uso general, y que no es el más adecuado para tratar con ondas de relativamente alta frecuencia como en nuestro proyecto.

Aun así, obtener una ganancia de 47.6 dB es un resultado más que considerable, ya que esta ganancia sumada a los 26dB que nos da la última etapa amplificadora nos da una ganancia total de unos 73.6 dB, o lo que es lo mismo una amplificación de la señal de entrada por un factor de 4700 V/V. Mostramos por último un par de resultados de la etapa de umbralizado, para poder obtener así la forma de onda de la señal de salida del bloque receptor.

Figura 3.3.15 captura de la señal de salida del bloque receptor



Ejemplos del tratamiento de las señales recogidas por el receptor ultrasónico; la señal proveniente del receptor, ruidosa y de pocos mV se convierte gracias a los procesos de amplificación filtrado y umbralizado en una señal lógica que determina si se ha recibido un eco o no

Como vemos, la salida del bloque amplificador corresponde a un pulso de una amplitud cercana a los 3.36 V, y cuyo ancho será variable dependiendo fundamentalmente de la distancia al objeto; mientras más alejado esté el objeto a medir, menor será el tramo de onda que al rebotar supere el umbral mínimo de tensión. El ancho del pulso, por otra parte, es del orden de los 800 µs, lo bastante grande como para que sea captado por el microcontrolador como un 1 lógico.

En resumen, observamos que el bloque receptor implementado se comporta de manera fiel al modelo diseñado, permitiéndonos filtrar y amplificar la señal para posteriormente, mediante un proceso de umbralizado, obtener una señal lógica que indique que se ha recibido eco de retorno. Como veremos con más detalla en el capítulo 5, con este diseño hemos conseguido una sensibilidad suficiente como para poder medir distancias de hasta 4.5 metros, cumpliendo con creces los objetivos iniciales de este proyecto.

3.3.3 Bloque de alimentación:

El bloque de alimentación será el encargado de suministrar la corriente adecuada al resto de dispositivos. Las especificaciones de potencia son bastante modestas, necesitando el medidor una tensión de funcionamiento de 5V y una corriente de 100mA, que se traduce en un consumo de potencia de 0.5 W. Se decidió diseñar un sistema de alimentación dual, en el que además de recibir corriente por medio de baterías, pudiese utilizar la propia conexión USB para funcionar, cambiando entre uno u otro método de alimentación por medio de un interruptor.

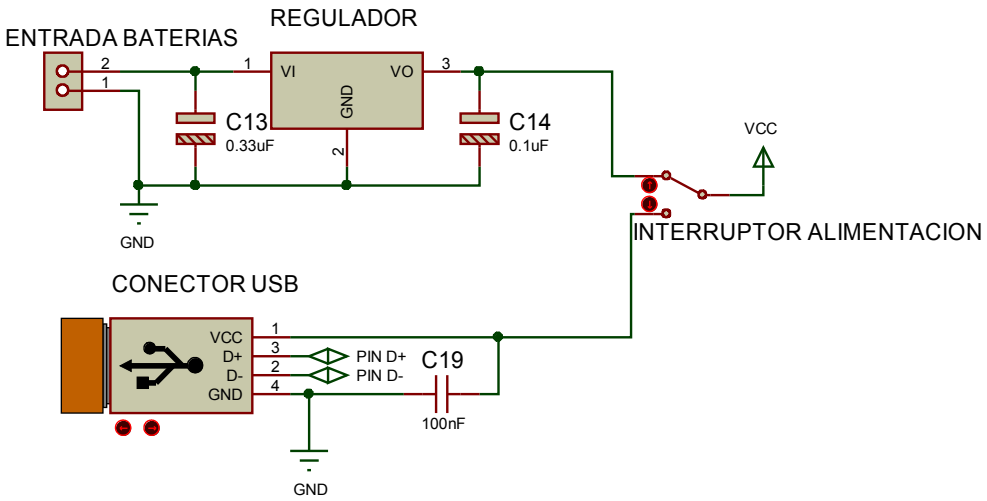
Para poder utilizar la conexión USB como suministrador de corriente debíamos tener en cuenta las especificaciones de potencia de esta especificación, que distingue entre 2 tipos de métodos de alimentación:

- Alimentación desde el bus USB. Permiten hasta 500mA de corriente, alimentados a 5V, pudiendo ser toda o parte de la corriente que precise el dispositivo suministrada vía USB. Durante el inicio y hasta que sea reconocido el dispositivo, no más de 100mA podrán ser empleados por el periférico, exigiéndose además que los periféricos que empleen este modo de alimentación puedan funcionar en un rango de tensiones entre 4.7 y 5.25 V.

- Alimentación desde el propio dispositivo. El suministro de corriente se realiza por medio de una fuente externa, aunque en casos de pérdida de potencia se puede solicitar hasta 100mA de corriente desde la conexión USB.

La alimentación por USB nos pareció una solución obvia, ya que el medidor puede comunicarse con cualquier otro dispositivo mediante una conexión USB estándar. Como los requisitos de consumo que exige el protocolo estándar USB entran dentro de lo que necesita el medidor, esto es, un voltaje de 5 V y una potencia menor a 2.5 W, se podría así alimentar directamente el dispositivo sin ningún tipo de regulación. Además, ya que se pretendía que fuese también usado como dispositivo manual de medición, se decidió incluir una segunda toma de alimentación mediante pilas. Basta con incluir 4 pilas AA de 1.5 voltios y un regulador de tensión para obtener un voltaje constante de 5V para que podamos proporcionar un número elevado de horas de autonomía.

Figura 3.3.16 diagrama eléctrico del bloque de alimentación



Los condensadores C13 y C14 asociados al regulador son los recomendados por la hoja de especificaciones del mismo y se utilizan para estabilizar la tensión y disminuir el rizado. El condensador C19 se coloca así mismo para estabilizar el voltaje proveniente de la conexión USB

Este diseño se demostró como una solución muy efectiva por lo reducido de su coste y la alta eficiencia y autonomía que proporciona al aparato medidor de distancias.

3.3.4 Bloque de captura de datos ambientales:

3.3.4.A. Influencia de los sensores ambientales en los errores de medida

La captura de datos ambientales resulta imprescindible para calcular con exactitud la velocidad del sonido del medio y en consecuencia realizar las mediciones con mayor precisión. Las características de los sensores ambientales debían elegirse teniendo en cuenta las prestaciones de la unidad de control en cuanto a velocidad de cálculo y captura de datos, así como la resolución a la hora de realizar internamente los cálculos.

En un primer diseño se consideró incluir tanto sensores higrométricos como de temperatura para caracterizar con la mayor exactitud posible las condiciones termodinámicas del medio que rodea al medidor, ya que como vimos en el capítulo 2 los factores que determinan la velocidad del sonido en el aire eran fundamentalmente la temperatura y la humedad relativa. Sin embargo, un estudio detallado de la ecuación

$v_s = 331.58 + 0.62T + 0.015\,HR$ nos permite observar que la contribución de la humedad relativa a la velocidad del sonido es del orden de $0.015 \cdot \Delta HR / 331.58$; considerando el caso más extremo en el que la variación de la humedad relativa sea del 100%, nos deja que los cambios en la velocidad del sonido serán como mucho del orden de $0.015 \cdot 100 / 331.58 = 0.45\,\%$.

Teniendo en cuenta que la distancia es directamente proporcional a la velocidad del sonido, comprobamos que como mucho la contribución de la humedad relativa en los errores de precisión sería pues del orden de 0.45 %. Así que nos plantea la cuestión de si merecía la pena el coste adicional que supone un sensor de humedad para corregir este error, o basarnos únicamente en la temperatura para estimar las condiciones termodinámicas del aire.

Había que tener en cuenta así mismo el precio típico de los sensores de humedad disponibles en el mercado, que ronda los 8 euros; un precio muy alto en comparación con el resto de componentes aquí utilizados y que haría que de utilizarlo el coste se encareciese en más de un 50%.

La necesidad del sensor de temperatura sin embargo estaba fuera de toda duda, pues su influencia es mucho mayor; al igual que hicimos con la humedad relativa, podemos calcular la contribución de la temperatura en la velocidad del sonido, que estará en torno a $0.62/331.58 \cdot 1000 = 1.87 mm/ºC$ por cada metro medido. A título comparativo podemos observar que una diferencia de temperatura moderada de 15ºC afecta a la velocidad del sonido en un 3%, unas 7 veces mayor. Su coste, así mismo, es del orden de los 0.3 €, unas 25 veces más barato. Con todos estos datos, decidimos finalmente que el coste del sensor higrométrico no compensaba la mejora de precisión, así que al final la unidad de captura de datos ambientales se limita a incluir un sensor de temperatura. Decidimos también incluir la posibilidad de corregir por software el efecto de la humedad en la ecuación, mediante un parámetro configurable desde cualquiera de los interfaces de comunicación disponibles.

3.3.4.B. Características del sensor de temperatura

Al final nos decidimos por utilizar como sensores analógicos de temperatura el termistor MCP9701A, cuyas principales especificaciones vienen reflejadas en la siguiente tabla;

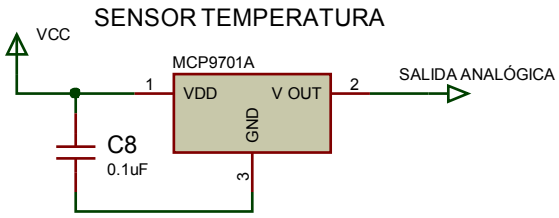
Figura 3.3.17: Características del sensor analógico MCP9701A

Parámetro	Valor
Precisión Típica	±1ºC
Consumo	0.03mW
Rango de Funcionamiento	-40 - 150ºC
Tensión de funcionamiento	3.1- 5.5 V
Coeficiente De Temperatura	19.53mV/ºC
Tensión a 0ºC	400mV
Encapsulado	TO-92
Rango	-20.48 ºC – 150 ºC

Este integrado nos ofrecía una respuesta bastante lineal en un rango amplio de temperaturas, con una precisión típica de ± 1 ºC en buena parte de su rango de funcionamiento, y por un precio bastante bajo. Además este sensor está especialmente diseñado para trabajar con los conversores analógico/digitales de los microcontroladores, ya que la conversión de analógico a digital de este coeficiente de temperatura se traduce en 1 bit/ºC si usamos una resolución de 8 bits, o 4 bits/ºC si usamos una resolución de 10 bit/ºC, lo que nos da una resolución de 0.25ºC/bit, más que aceptable para realizar mediciones de temperatura.

El encapsulado TO-92 resulta ideal por su sencillez (consta de solo 3 pines; alimentación VDD, tierra GND, y la salida analógica VOUT) y es el más adecuado para realizar mediciones de la temperatura del aire. Este sensor no requiere de circuitería externa, si bien se recomienda incluir un condensador de desacoplo entre el pin de alimentación y el de tierra, cuyo valor recomendado por el fabricante es de 0.1 a 1µF. El diagrama esquemático asociado tiene pues la siguiente estructura;

Figura 3.3.18: Diagrama eléctrico de la unidad de captura de datos ambientales



3.3.4.C. Relación entre la tensión del sensor ambiental y la temperatura

La ecuación que permite obtener la relación entre la tensión y la temperatura viene definida por la ecuación siguiente, $V = 0.400 + 19.53\text{e-}3 * T$, donde V es la tensión en escala 0-5 Voltios a la salida del sensor MCP9701A y T es la temperatura en grados centígrados. Como trabajamos con un conversor A/D con 10 bits de precisión, debemos cambiar la escala para usar la escala de tensión de 0 a 2^{10} , lo que da como resultado que la temperatura depende de la tensión (en escala de 10 bits) de acuerdo con la ecuación, $T = 0.25V - 20.48$, donde

nuevamente T se expresa en grados centígrados y la tensión en escala de 10 bits.

Dado que nuestro microcontrolador no trabaja en coma flotante y queremos una precisión mínima de ¼ de grado, multiplicamos por 100 la ecuación anterior, teniendo al fin la ecuación $T_{100}=25\,V-2048$, con T medida en centésimas de grado y V medida en escala de 10 bits (de 0 a 1024).

Esta última ecuación será precisamente la empleada por la unidad de control para obtener la temperatura en centésimas de grado a partir del valor de tensión proveniente del sensor de temperatura, permitiéndonos obtener valores de temperatura con una resolución de un cuarto de grado en un formato de 16 bits con signo (hay que recordar que los límites de temperatura que soporta el sensor oscilan entre los -20.48 ºC y los 150.00 ºC).

3.3.4.D. Relación entre la tensión del sensor ambiental y la velocidad del sonido

Por otra parte, la dependencia de la velocidad del sonido con respecto de la humedad y la temperatura viene dada, como vimos antes, por la ecuación,

$$v_s=331.58+0.62\,T+0.015\,HR$$, c

on V_s en m/s, T en ^{o}C y la humedad relativa en escala porcentual 0-100% . Consideramos en primer lugar, como en el caso anterior, el digitalizado de la tensión proveniente del sensor de temperatura y su conversión a una escala de 10 bits. Así mismo trabajaremos con una variación de la humedad ambiental reflejada por una variable de 8 bits de longitud, donde 0 corresponde a una humedad relativa del 0% mientras que la humedad máxima de 100% vendrá reflejada por el valor 255; de esta manera la ecuación queda finalmente;

$$V_s=318.88+0.155\,V+0.005882\,HR$$,

donde la velocidad del sonido V_s está expresada en m/s, V es el voltaje de salida del sensor de temperatura en escala de 10 bits y HR es la humedad relativa en escala de 8 bits.

Como en el caso anterior evitamos los decimales multiplicando por 100 los dos miembros de la ecuación y convirtiendo en fracciones los decimales, tenemos que la velocidad del sonido, en cm/s es:

$$V_s = 31888 + (V * 31)/2 + (HR * 3)/5$$

donde la tensión V mide el valor de tensión a la salida del sensor MCP9701A en escala de 10 bits y la humedad relativa está expresada en escala de 8 bits. Esta última ecuación es la que estará incluida en la unidad de control para obtener la velocidad del sonido a partir de la temperatura ambiente.

3.3.4.E. Corrección de errores de deriva del sensor de temperatura

Si se quiere corregir posibles errores de deriva debidos a mala calibración del sensor de temperatura podemos añadir el efectos del error de deriva en las ecuaciones anteriormente descritas incluyendo el término T_{100_deriva} que contrarreste tal efecto, de forma que la temperatura real, en centésimas de grado, queda $T_{100\,real} = T_{100} + T_{100\,deriva}$, medido como siempre en centésimas de grado.

Así pues la temperatura real queda de acuerdo con la ecuación $T_{100\,real} = 25\,V - 2048 + T_{100\,deriva}$,mientras que la velocidad del sonido real, por su parte, queda de la forma, $V_{s\,real} = 31888 + (V * 31)/2 + (HR * 3)/5 + (T_{100\,deriva} * 5)/8$.

El parámetro de corrección del error de deriva será configurable por el usuario desde los interfaces de uso mediante RS-232 y/o USB, permitiendo así una mayor exactitud del medidor de distancias.

3.3.5 Interfaz de comunicación manual:

3.3.5.A. Interfaz de control

El objetivo de la interfaz de comunicación manual es el de permitir al usuario tener un acceso a las funciones principales del medidor de distancia sin depender de una conexión con un PC o un dispositivo similar. La idea de partida era maximizar la simplicidad de manejo del medidor y hacer completamente transparente el uso de esta interfaz a cualquier usuario.

Nuestra primera decisión fue no incluir opciones de calibración ni de corrección estadística en esta interfaz; pensamos que lo más adecuado sería que estas opciones fuesen configuradas desde el interfaz de comunicación con el PC, de modo que cuando se usara el modo manual se operase con los parámetros de trabajo almacenados en la memoria permanente del microcontrolador. Si el usuario deseara modificar algún parámetro de configuración o estadístico simplemente deberá reconectarlo a un PC o similar y cargar la nueva configuración, funcionando a partir de ese momento con la nueva configuración guardada, y permaneciendo en memoria cuando se apague el dispositivo.

Esta decisión no solamente responde a un deseo de simplicidad, sino que realmente la mayoría de estos parámetros no necesitan ser modificados durante una misma sesión de uso, ya que muchos de ellos se usan para una configuración “fina” del dispositivo; algunos de ellos como la corrección del tiempo muerto entre recepción y emisión o el error de deriva solamente precisarán ser modificados en casos muy especiales en los que el dispositivo vaya a trabajar en condiciones muy diversas de funcionamiento.

Otros parámetros como la configuración del número de muestreos afectan de manera secundaria al funcionamiento del dispositivo; cuando se usa en modo manual, realizar un mayor número de mediciones y obtener su valor de mediana redunda siempre en una mayor precisión en los resultados, y además aquí el inconveniente asociado a los retardos entre diferentes conjuntos de mediciones no resulta crítico, puesto que su uso manual implica una baja frecuencia de

medición. Es aconsejable de hecho cambiar modificar este valor antes de utilizar el aparato como dispositivo manual e incluir un valor elevado.

La segunda decisión fue seleccionar, de entre todos los modos posibles de medición de distancia (medición única, lote de medidas o medición con filtro de mediana) el que mejor se adecuase a las condiciones de uso que tendrá el aparato en modo manual. Había que tener en cuenta que su uso como dispositivo manual implica una gran inestabilidad del dispositivo y movimientos y giros continuos, por lo que obviamente la opción elegida fue incluir la opción de sets de múltiples con filtro de mediana, para suavizar estos errores y garantizar una medición robusta.

Así pues decidimos finalmente utilizar un número de opciones muy reducidas, reduciéndolas únicamente a **dos** opciones de manejo; realización de mediciones de distancia y captura de la temperatura ambiente. La medición de distancia es al fin y al cabo es el propósito principal del dispositivo, mientras que mostrar la temperatura ambiente es un dato interesante para mostrar al futuro usuario de nuestro aparato. Decidimos emplear un botón para activar las órdenes de medición y otro para activar las órdenes de lectura de temperatura. Por último debíamos incluir un interruptor que permitiera el cambio de modo de funcionamiento manual a automático. Será preciso por tanto emplear 3 pines configurados como entradas digitales de la unidad de control para la comunicación entre la interfaz de control manual y la unidad de control.

3.3.5.B. Interfaz de visualización

Una vez decididos los comandos disponibles para el usuario restaba por implementar la interfaz de visualización que muestre al usuario el resultado de la medición. La opción más económica y que mejor se ajustaba a nuestros propósitos era incluir un LCD de pequeño tamaño, de 16x1 caracteres, que nos permitía mostrar los datos requeridos así como mensajes de estado. Hay que tener en cuenta que tanto la temperatura como la distancia precisan de 5 dígitos para ser mostrados por pantalla, por lo que pueden mostrarse de sobra en un display de 16 caracteres.

Nos decidimos por un LCD de estas dimensiones que estuviese controlado por el integrado hd44780, que en su configuración más sencilla permite controlar displays gráficos con 4 pines de datos y dos señales de control. Así pues el control del display gráfico precisa en total de 6 pines digitales de salida de la unidad de control.

3.3.5.C. Esquema completo

La lista completa de botones y pulsadores y se reduce por tanto a 3, y las acciones asociadas serán las siguientes;

Figura 3.3.19: Comandos de la interfaz manual y acciones asociadas

Elemento de la interfaz manual	Acción
Botón de medición	Mientras está pulsado, el dispositivo realiza continuamente mediciones con filtrado de mediana. Mostrará por el LCD la mediana de los valores muestreados, así como el número de medidas fallidas.
Botón de temperatura	Mientras esté pulsado, el dispositivo muestra realiza continuamente mediciones de la temperatura ambiente y las muestra por el LCD
Interruptor de modo manual/ automático	Cambia entre el modo manual y el modo automático.

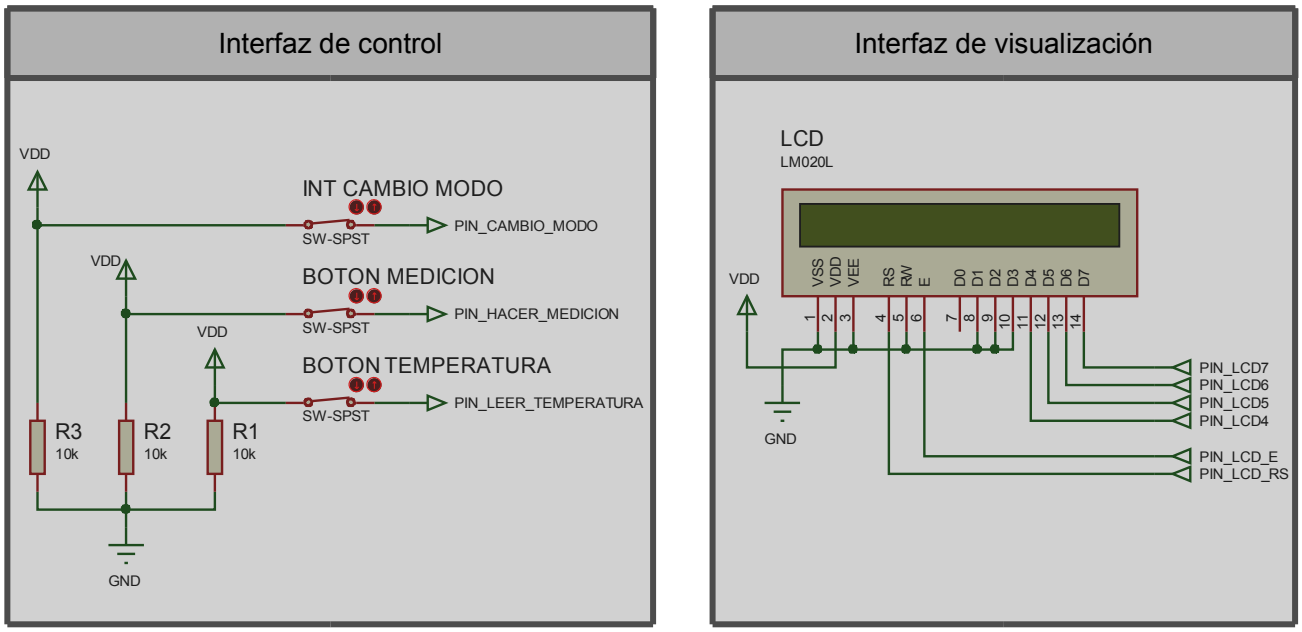
La lista de pines necesarios viene así mismo indicada en el cuadro siguiente;

Figura 3.3.20: lista de pines usados por la interfaz manual

Nombre	Tipo	Función
PIN_LCD7 PIN_LCD6 PIN_LCD5 PIN_LCD4	Entradas	Bus de datos del LCD.
PIN_LCD_E	Entrada	Señal de reloj /Enable del LCD; E=1 habilita LCD, E=0 deshabilita LCD.
PIN_LCD_RW	Entrada	Selección del registro; RS=1 datos, RS=0 instrucciones.
PIN LEER TEMPERATURA	Salida	Envía orden de lectura de la temperatura ambiente
PIN HACER MEDICION	Salida	Envía orden de medición de distancia
PIN CAMBIO MODO	Salida	Envía orden de cambio modo manual /automático

Por último, el esquema eléctrico completo es;

Figura 3.3.21: diagrama eléctrico de la interfaz de usuario



3.3.6 Interfaz de comunicación con el PC:

En este apartado nos centraremos únicamente en explicar cómo esta diseñado el interfaz de comunicación con el PC a nivel de hardware, dejando la interfaz de software para el capítulo siguiente. Aquí por tanto explicaremos brevemente los protocolos de comunicación sobre los que están basados los interfaces que permiten conectar el medidor con un PC, interfaces que serán también válidos para comunicar nuestro aparato con prácticamente cualquier dispositivo microcontrolado.

Se decidió que los interfaces de hardware que poseyera nuestro medidor de distancia fuesen mediante USB y RS-232. Con el primero de ellos se pretendía que pudiese ser conectado con cualquier computadora e incluso con microcontroladores de alta gama, mientras que el RS-232 se incluyó para que pudiera conectarse con aparatos más a antiguos o de uso industrial, así como con microcontroladores de la gama media-baja. A continuación explicaremos brevemente cada uno de los interfaces de hardware incluidos y el esquema eléctrico que los implementa.

3.3.6.A. Puerto Serie RS-232

El estándar RS-232 designa una norma para el intercambio de datos digitales entre diferentes dispositivos mediante la transmisión bit a bit de la información, a diferencia de la comunicación en paralelo donde un número determinado de bits son enviados simultáneamente. Aunque a nivel lógico existen otros estándares que también envían el flujo de datos en serie, generalmente la especificación puerto serie se refiere al estándar RS-232. Este estándar, si bien está ya obsoleto en el campo de los ordenadores personales y en aplicaciones de uso doméstico y empresarial, habiendo sido mayoritariamente desplazado por el estándar USB, se sigue usando con frecuencia en numerosos aparatos de uso industrial y electrónico como por ejemplo muchos microcontroladores de gama media-baja (8-16 bits). El módulo que se encarga de la comunicación por puerto serie se denomina USART, acrónimo de ***universal asynchronous receiver/transmitter*** , y está presente en la práctica mayoría de microcontroladores existentes.

Resulta por tanto interesante habilitar una vía comunicación por RS-232 con nuestro medidor de distancia y permitir así la interacción de nuestro aparato con microcontroladores de similares prestaciones al que emplea la unidad de control, el pic18f4550. Ello permite que nuestro dispositivo pueda formar parte de aplicaciones de robótica a pequeña escala, además de permitir la comunicación con dispositivos industriales como PLC's o similares.

El estándar RS-232 describe un método de comunicación donde la información es enviada bit a bit por medio de un canal lógico. Para ello la información debe ser dividida en palabras de determinada longitud, comprendidas normalmente entre los 5 a 8 bits. Esta información puede ser transmitida siguiendo una señal de reloj, denominándose entonces comunicación síncrona, o sin ayuda de una señal externa, o de forma asíncrona. El primero de los casos permite transferencias a velocidades más elevadas y menor tasa de error, si bien tienen un coste más alto. El estándar RS-232, en sus últimas versiones, permite tasas de transferencia relativamente altas, del orden de los 1.5 Mbps. En nuestro proyecto trabajaremos a 115200 baudios, en modo asíncrono.

En el modo asíncrono se deben incluir los denominados bits de inicio y de parada(entre 1 y 2), para indicar al dispositivo que está a la escucha que se van enviar un nuevo grupo de bits. Adicionalmente se puede incluir bits para detección de errores, como los bits de paridad.

Es importante señalar que para que se pueda producir una correcta comunicación entre dos dispositivos se deben configurar ambos para poder leer y escribir a una misma tasa de transferencia así como conocer el número de bits de datos, los bits de parada y, en caso de existir, el tipo de bit de paridad incluido. La configuración con la que trabajamos consta de 8 bits de datos, un bit de parada y sin bit de paridad; esta configuración es denominada también 8N1, y representa el tipo de conexión más común por este puerto.

Los ceros y unos lógicos corresponden con diferentes niveles de voltaje, de acuerdo con las siguientes especificaciones;

Figura 3.3.22: niveles de voltaje del estándar RS-232

Nivel lógico	Nivel de Tensión del Transmisor	Nivel de Tensión del Receptor
0	+5..+15 V	+3.. +25 V
1	-15..-5 V	-25.. -3 V

Mientras mayor sea en magnitud los niveles de tensión menores serán los efectos del ruido y mayor será por tanto la calidad de la transmisión. Estos valores de tensión vendrán determinados por la fuente de potencia asociada a cada dispositivo. Como puede observarse estos valores de tensión están completamente alejados de los niveles lógicos utilizado por los circuitos integrados, por lo que se precisan conversores de tensión para pasar de la lógica de los circuitos integrados o TTL, que en nuestro caso corresponde a 0 V para un 0 lógico y 5V para un 1 lógico. En nuestro proyecto decidimos utilizar un conocido conversor de niveles entre TTL y RS-232, el MAX232N, que nos permite por un coste muy bajo realizar esta conversión.

Hay que recordar que este conversor contiene dos canales para realizar dos conversiones independientes entre RS-232 y TTL, y que uno de estos canales se utilizó en la etapa amplificadora para cuadruplicar la señal de entrada al emisor ultrasónico. Ello es explicó en el apartado 3.3.1.

El módulo USART de la unidad de control será el encargado de establecer la comunicación entre cualquier dispositivo y el propio microcontrolador, permitiéndonos incluir este protocolo de comunicación y añadir un interfaz de software para su conexión con computadoras o microcontroladores

3.3.6.B. Estándar USB

La especificación USB es un estándar que en su origen fue diseñado para unificar todos los interfaces y conectores asociados a los periféricos de los PC (RS 232, puerto paralelo, PS-2...), simplificando con ello el software asociado a cada dispositivo, así como aumentando el ancho de banda para nuevos periféricos. Funciona no solamente como interfaz de comunicación entre dispositivos, sino también como interfaz de suministración y distribución de energía, de modo que gestiona la energía que los periféricos precisan, garantizando un consumo adecuado y protegiéndolo ante consumos excesivos y cortocircuitos.

En primer lugar, el hecho de que la interfaz USB pueda funcionar como distribuidora y administradora de energía permite que periféricos con bajo consumo puedan alimentarse directamente del cable de conexión USB. Como ya explicamos en el apartado anterior cuando abordamos el bloque de alimentación, los bajos requisitos energéticos de nuestro medidor de distancia nos permitieron utilizar esta opción ahorrando consumo de baterías mientras está conectado mediante USB.

La topología USB tiene un diseño asimétrico, en el que existe una multitud de puertos por los que los periféricos se conectan a un servidor central (generalmente un PC), de acuerdo con una topología en estrella de múltiples niveles. Los diferentes periféricos no pueden ser conectados entre sí, ya que únicamente el servidor central es quien puede controlar al resto de periféricos.

El estándar USB permite a los periféricos el poder desenchufarse y enchufarse sin tener que desconectar o reiniciar el servidor Este proceso, denominado “enumeración”, implica que cada vez que un periférico sea conectado a un puerto USB, el servidor le solicite información para que le indique qué driver debería instalar para poder comunicarse correctamente. A cada periférico se le asigna una sola dirección durante la enumeración para ser usada por las transferencias en tiempo de ejecución. Durante este tiempo, el PC (servidor) inicia el proceso de comunicación mediante transacciones con los periféricos específicos para posteriormente aceptar cada periférico su transacción y responder en adelante de acuerdo con ella, haciendo transparente al usuario la instalación de nuevos dispositivos, y evitando tener que configurar puertos, IRQ's y demás.

A partir de ese momento los periféricos responderán para controlar las transacciones que, por ejemplo, pidan información detallada sobre el dispositivo y su configuración, o información sobre la gestión de energía, etc... El periférico envía y recibe datos al / desde el servidor usando el formato de datos standard USB. Así, desde el punto de vista del periférico, todos los periféricos USB son esclavos que obedecen a un protocolo determinado. Deben reaccionar para responder las transacciones enviadas desde el PC servidor; en caso de no reaccionar a alguna transacción (sea por un bloqueo a nivel de ejecución o a un error en la transmisión) el servidor considera que el periférico ha desaparecido y lo borra de la lista de dispositivos enumerados, debiendo desenchufar el periférico y volverlo a conectar para recomenzar el proceso de enumeración y asignarle una nueva dirección.

Esto fue un factor importante a tener en cuenta , puesto que según esto no podíamos tener nuestra unidad de control bloqueada en ningún instante, lo que se traduce en limitar en lo posible el uso de interrupciones en nuestro microcontrolador y de rutinas bloqueantes. Pero por otro lado debíamos realizar el proceso de medición del tiempo de vuelo lo más preciso posible, sin que sea viera sometido a retardos asociados a los procesos de recepción y respuesta de transacciones por parte del servidor. Será en el siguiente capítulo cuando detallaremos la solución adoptada, aunque podemos adelantar que optamos por una solución de compromiso en la que apuramos los tiempos en los que dejábamos a la unidad de control bloqueada sin que por ello perjudicase el proceso de transacciones, a costa de aumentar en gran medida el buffer de memoria asociado al USB.

La instalación del driver correspondiente se realiza mediante una combinación de parámetros denominados PID/VID (identificador de producto e identificador de comerciante, respectivamente); gracias a ello se identifica el tipo de periférico que se ha conectado y se podrá gestionar el servidor la potencia que precisa, el tipo de transacciones que permitirán la comunicación, la tasa de transferencia, etc... El VID es suministrado por los desarrolladores del estándar USB bajo licencia de pago, si bien existen VID gratuitos para fines no comerciales. En nuestro caso utilizaremos la combinación VID/PID suministrada por Microchip, que otorga una licencia de uso PID/VID gratuita para fines no comerciales en aquellos dispositivos que comercializa bajo el estándar USB.

El estándar USB soporta 4 diferentes tasas de transferencia

- *SuperSpeed* (USB 3.0) : a 4.8 Gigabits por segundo
- *High-speed* (USB 2.0) : a 480 megabits por segundo
- *Full-speed* : a 12 megabits por segundo
- *Low-speed* : a 1.5 megabits por segundo

Nuestro dispositivo funcionará bajo el modo USB 2.0 full-speed, el cual soporta cuatro tipos de transferencia de datos; Control, Bulk, Interrupción e Isócrona.

- Control : las transferencias de Control se usan normalmente para comunicaciones de estatus y comando (durante el proceso de enumeración, por ejemplo) y se realizan en ráfagas no periódicas y de tipo bidireccional.

- Bulk : las transferencias de tipo Bulk también se realizan en ráfagas, pero estas son de un tamaño mucho mayor, ocupando normalmente todo el ancho de banda que en ese momento esté disponible. Este tipo de transferencia utilizar el método de detección de error vía CRC, garantizando el envío y se utilizan para grandes transferencias de datos, aunque no garantiza una latencia o ancho de banda mínimos. Son además de tipo unidireccional.

- Interrupción: este tipo de transacciones se utiliza cuando se requiere una respuesta rápida con una latencia mínima garantizada. Hay que aclarar sin embargo que el concepto de "interrupción" está referido a cómo actúa esta transacción a nivel de periférico, enviando una señal de interrupción que será gestionada por el servidor, según una frecuencia de lectura fija. Si una transacción ha resultado fallida se detecta el error y se intentará establecer la transacción en la siguiente lectura por parte del host. Son también de tipo unidireccional.

- Isócronas: estas transacciones se utilizan para transmisión de datos donde sea importante asegurar un ancho de banda y una latencia mínima, a costa de no permitir el reenvío de paquetes corruptos ni garantizar su envío, si bien incluye detección de errores por CRC.

El proceso de enumeración se realizará mediante transferencias de control mientras que la transferencia de datos se realizará mediante bulk e interrupciones.

3.3.6.C. Interfaz de comunicación mediante USB-CDC

La especificación USB define *clases* que identifican el tipo de periférico dependiendo de la función que realiza y permiten configurar el host en consecuencia; audio, video, impresora, imágenes o dispositivos de comunicación o control como módems o adaptadores de ethernet. Esta última clase es llamada *Communication Device Class* o CDC, y será la clase bajo la cual funcione nuestro dispositivo. El objetivo de esta clase es la de servir de soporte a cualquier periférico que opere bajo los más diversos protocolos de comunicación, sean orientados a las telecomunicaciones como podrían ser modems analógicos, ISDN o teléfonos digitales, o orientados a redes, como modems ADSL, adaptadores o hubs Ethernet, etc...

Uno de los protocolos de comunicación que puede implementar es el protocolo RS232, de manera que empleando esta clase de podrá comunicar nuestra unidad de control con nuestro servidor utilizando una conexión USB, en la que el PC actuaría de host y nuestro medidor como periférico esclavo, comunicándose entre ellos mediante un puerto serie virtual o Virtual COM. Así, durante el proceso de enumeración se reconoce el dispositivo como uno asociado a la Communication Class Device; configurando además de manera adecuada las transacciones para emular el puerto RS-232, tenemos que al final del proceso de enumeración se crea un nuevo puerto COM virtual.

Utilizar la clase CDC para implementar un puerto serie virtual permite que cualquier software del lado del host que esté capacitado para comunicarse por RS-232 también podrá comunicarse usando el puerto serie virtual, sin ningún cambio en su programación más que especificar el número de puerto COM virtual. Las ventajas de ello son obvias; podemos crear una única interfaz de usuario que funcione de modo idéntico para una conexión RS232 o COM virtual, y lo que es más importante, podemos utilizar esta única interfaz de usuario para comunicar nuestro medidor con dispositivos tan dispares como PC u otros microcontroladores de gama media-baja.

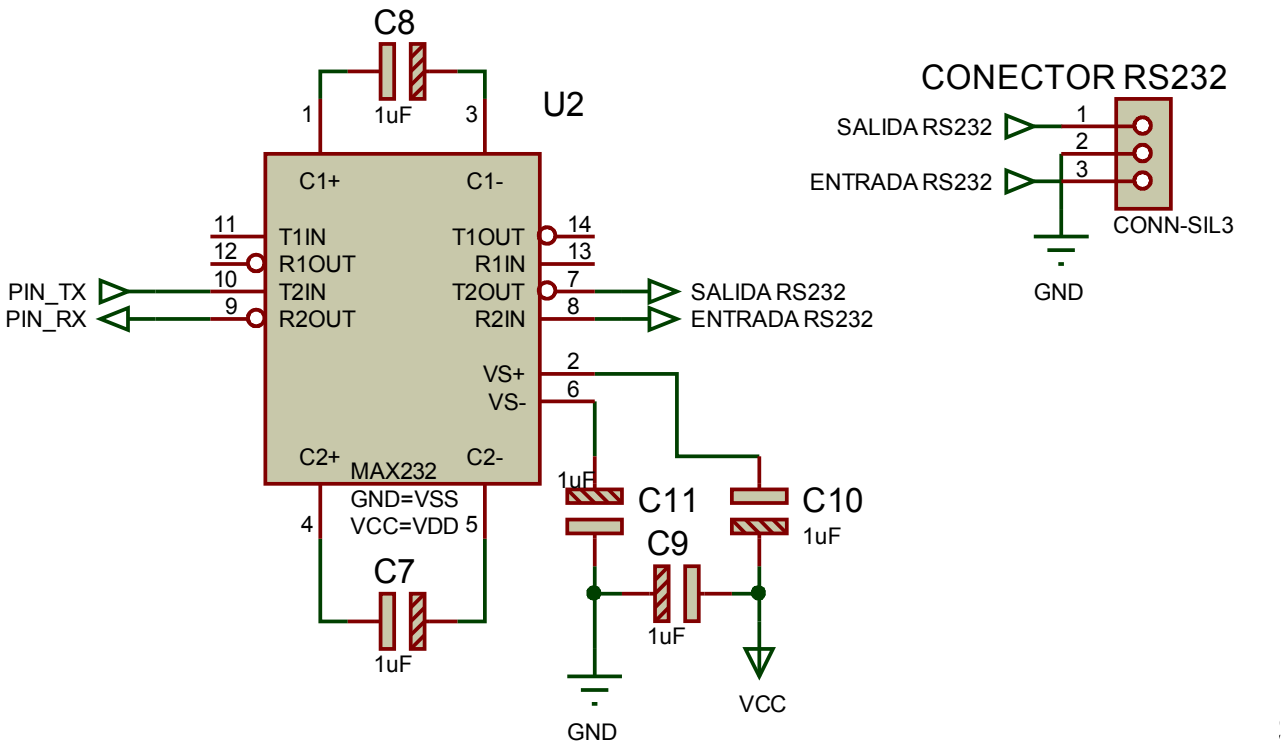
El microcontrolador dispone de un periférico integrado que se encarga de la comunicación mediante USB, permitiendo así añadir esta interfaz de comunicación. Se requerirán dos pines digitales concretos, D+ y D-, para realizar esta comunicación. En el apartado siguiente nos centraremos en el esquema eléctrico asociado a la comunicación USB y RS-232.

3.3.6.D. Implementación de las conexiones USB y RS-232

- Comunicación por RS-232

Como ya vimos al estudiar la etapa de amplificación de la señal emisora, los condensadores C7 a C11 son requeridos por el integrado para realizar la conversión de niveles de tensión de lógica TTL a lógica RS-232. Los pines pin_rx y pin_tx están asociados respectivamente a la recepción y la emisión de la comunicación serie y operan a niveles TTL, estando por tanto directamente conectados al periférico integrado en la unidad de control. Por otro lado, los pines entrada_rs232 y salida_rs232 serán los asociados a la recepción y transmisión del puerto serie y con lógica RS-232. La conexión RS-232 usa el número mínimo de pines posibles, limitándose a incluir aparte de los pines de recepción y transmisión una referencia de tierra.

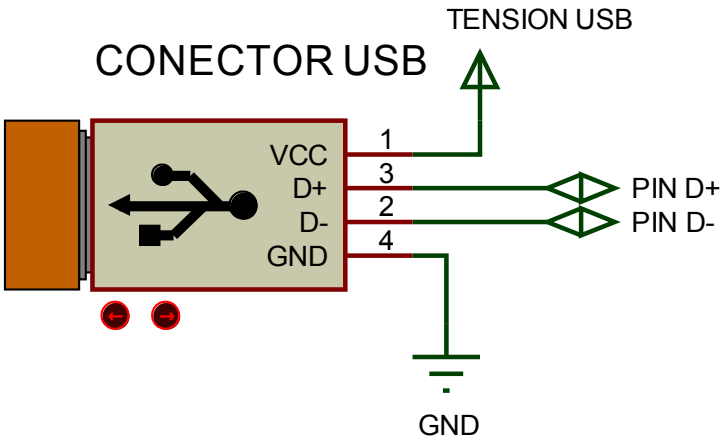
Figura 3.3.23: diagrama eléctrico de la conexión RS-232



- Comunicación por USB

En este caso el circuito resulta mucho más sencillo, pues no es necesario adaptar las señales provenientes del conector USB antes de llevarla a la unidad de control. Así pues únicamente se requería comunicar los pines D- y D+ directamente con el periférico integrado dentro de nuestro microcontrolador. Se ha decidido

Figura 3.3.24: diagrama eléctrico de la conexión USB



3.3.7 Unidad de control:

3.3.7.A. Elección del microcontrolador

La unidad de control es la que dirige todo el proceso de medición, captura de datos ambientales y comunicación con el exterior. Si en los anteriores apartados hemos visto cómo funcionaban los diferentes sistemas pertenecientes al medidor, ahora veremos cómo estos bloques se comunican con la unidad de control así como los componentes y circuitos asociados a este subsistema. La solución por la que optamos pasó por emplear un microcontrolador en el que estuvieran incluidos todos los periféricos mencionados en los apartados anteriores; interfaces USB y RS-232 para comunicación, unidad de cálculo y memoria para incluir el programa y realizar las mediciones, conversores analógico digital para la captura de datos ambientales y un número moderadamente elevado de entradas y salidas digitales.

De esta manera con un único integrado tendríamos incluido casi todo lo necesario para que la unidad de control realice todas las tareas que se le requieran, con el consiguiente ahorro en coste, diseño y espacio. Así pues la unidad de control quedará reducida al microcontrolador que será quien efectúe todas las tareas de comunicación y cálculo así como por algunos componentes discretos que precisa el microcontrolador para funcionar, como relojes, capacitores y resistencias, y que no suelen estar incluidos dentro del microcontrolado bien sea por una cuestión de economía, ahorro de espacio o para permitir una mayor versatilidad en el uso del integrado.

La cuestión más importante recaía pues en elegir el microcontrolador que más se adecuase a nuestro requerimientos en cuanto a velocidad de cálculo, ancho de palabra, número de pines analógicos y digitales libres, interfaces de comunicación disponibles..., ya que sería determinante tanto en el diseño del hardware como del software asociado.

En primer lugar debíamos estimar los requisitos mínimos que precisaba un dispositivo como el nuestro para poder así cribar los que tengan unas características inferiores a las deseadas. También consideramos muy importante la disponibilidad de entornos de desarrollo y lenguajes de programación atractivos para la programación de los diferentes microcontroladores, el soporte que ofrecen, y por supuesto, el precio. La lista de requisitos se detalla a continuación;

Figura 3.3.25: requisitos mínimos necesarios

	Requisitos mínimos	**Aclaraciones**
Número de E/S digitales	20 pines E/S	9 para interfaz manual , 4 para interfaz de comunicación con el PC, 5 para unidad de control, 2 para emisión/recepción ultrasonidos
Número de entradas analógicas	1 pin	1 entrada analógica para el sensor de temperatura.
Interfaces de comunicación	USB Full Speed, USART	USB Full Speed para implementar USB-CDC USART para implementar RS-232
Memoria de programa	24Kbytes	-
Memoria de datos	1Kbyte	-
Velocidad (MIPS)	12MIPS	-

Teniendo en cuenta estos requerimientos así como las consideraciones anteriores, nos decidimos primeramente por seleccionar microcontroladores de la familia Microchip. La razón que nos llevó a ello fue en primer lugar la enorme cantidad de información, ejemplos y tutoriales disponible para los integrados de esta familia, sobre todo en el rango de los microcontroladores de 8 bits, así como la política de envío de samples (muestras) gratuitos. Prueba de ellos es que por ejemplo pudimos conseguir de manera gratuita tanto el sensor de temperatura como el propio microcontrolador.

Esta empresa posee un surtido muy amplio de microcontroladores que van desde los 4 a los 32 bits, siendo líder mundial en el sector de los microcontroladores de 8 bits, con una enorme representación en el sector de consumo y automovilístico. Consideramos que utilizar microcontroladores de la gama de 16 o 32 bits no sería necesario y supondría un coste adicional y una complejidad excesiva en cuanto a diseño y programación, con lo que al final decidimos seleccionar el integrado más adecuado de la familia de los 8 bits, que es precisamente la familia más conocida y extendida de esta compañía.

Efectuamos una búsqueda de acuerdo con los requisitos arriba expuestos y obtuvimos una lista con todos los integrados que superaban los requisitos arriba expuestos, todos ellos pertenecientes a la gama mejorada 18f;

Figura 3.3.26: comparativa entre diferentes microcontroladores

	Requisitos mínimos	**PIC18f2455**	**PIC18f2550**	**PIC18f4455**	**PIC18f4550**
Número de E/S digitales	20	24	24	35	35
Número de entradas analógicas	1	10	10	10	10
Interfaces de comunicación	USB Full Speed, USART	USB Full Speed, USART	USB Full Speed, USART	USB Full Speed, USART	USB Full Speed, USART
Memoria de programa	24Kbytes	24Kbytes	32Kbytes	24Kbytes	32Kbytes
Memoria de datos	1Kbyte	2Kbytes	2Kbytes	2Kbytes	2Kbytes
Velocidad (MIPS)	12MIPS	12 MIPS	12 MIPS	12 MIPS	12 MIPS
Precio	-	3.30 €	3.44 €	3.51 €	3.65 €

Como vemos, los únicos criterios verdaderamente significativos son el tamaño de memoria de programa, datos y el número de pines E/S disponibles. Teniendo en cuenta el carácter experimental de este proyecto y la escasa diferencia de precio entre uno y otro microcontrolador, decidimos trabajar con el que mejores prestaciones de memoria y pines E/S nos daba, el PIC18f4550, para poder trabajar con mayor comodidad y permitir futuras mejoras en el diseño del medidor. Una vez elegido este microcontrolador ya podíamos diseñar el resto de la unidad de control, seleccionando los componentes asociados y efectuando las conexiones con el resto de bloques funcionales.

3.3.7.B. Esquema eléctrico de la unidad de control

Como ya señalamos anteriormente la unidad de control estaba formada por el microcontrolador y cierto número de componentes discretos asociados. En primer lugar debíamos incluir el oscilador externo, que en nuestro caso estará formado por un reloj de oscilador de cuarzo a 12 Mhz y dos condensadores cerámicos que garantizan una frecuencia estable. Aunque teníamos la opción de utilizar el oscilador interno del que dispone nuestro pic18f4550, este no era tan exacto como uno externo y estaba más sujeto a perturbaciones, por lo que decidimos utilizar uno externo.

Así mismo en esta familia de PIC's el módulo USB precisa de un condensador de 47uF para funcionar, ya que este modulo funciona internamente a 3.3 voltios y el condensador es necesario para realizar la regulación de tensión. Este condensador no se incluye dentro de estos microcontroladores integrarlo dentro del periférico encarecería enormemente el precio final y ocuparía muchísimo espacio.

Por otra parte decidimos incluir un puerto de programación en nuestra placa de circuito para futuras revisiones del firmware del programa y hacer más sencillo su testeo. De esta manera podemos reprogramar nuestro medidor siempre que nos haga falta sin necesidad de extraer físicamente el microcontrolador. Este puerto se conoce como ICSP (In-Circuit Serial Programming) y precisa únicamente de dos pines dedicados de nuestro microcontrolador, por lo que podemos permitirnos su uso.

Se decidió también incluir un pulsador que tuviera la función de forzar un reset externo; esta opción es muy útil ya que permite eliminar posibles bloqueos o funcionamientos anómalos sin tener que recurrir a desconectar la alimentación del medidor. El microcontrolador, por último, necesita como el resto de integrados ser alimentado por una corriente de 5V y disponer de una conexión a tierra para cerrar el circuito.

Se adjunta a continuación un diagrama de bloques con todas las conexiones entre la unidad de control y el resto de bloques funcionales, así como las conexiones entre el microcontrolador y los componentes discretos que precisa para su funcionamiento.

Figura 3.3.27: conexiones entre la unidad de control y el resto de bloques funcionales

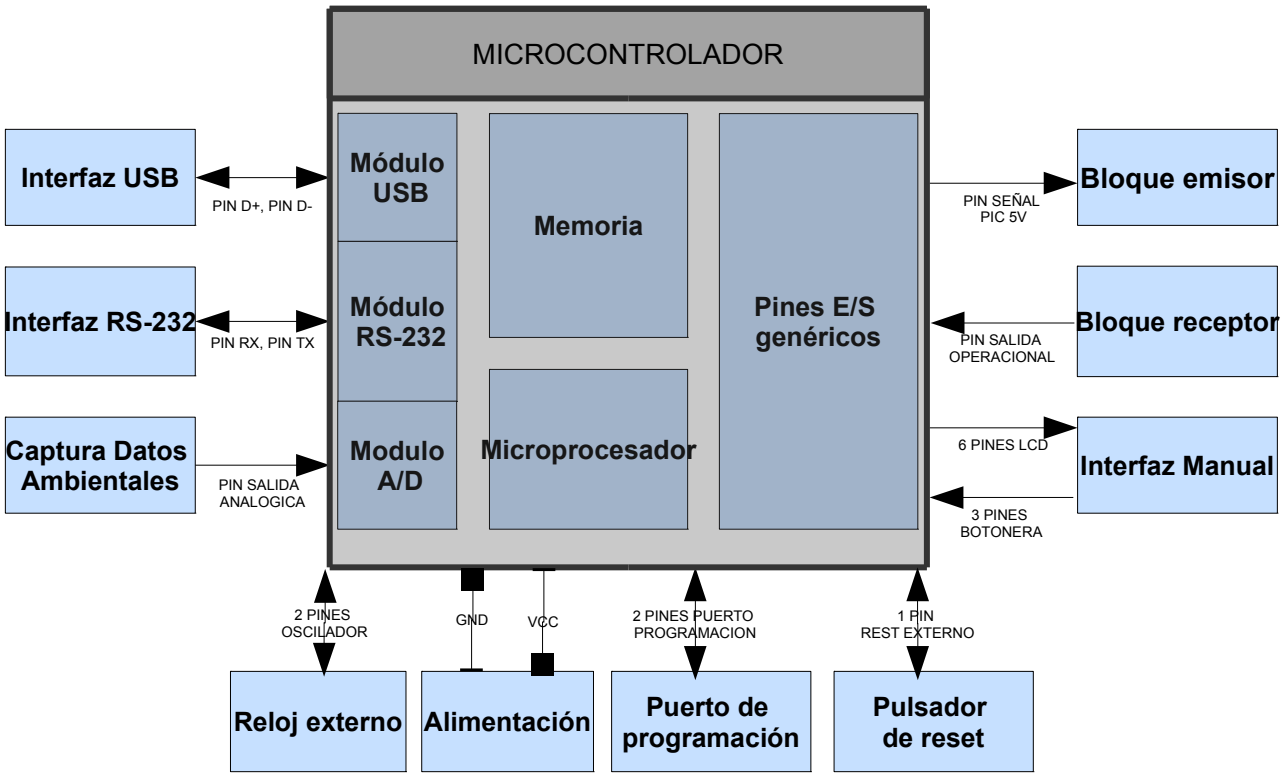


Diagrama funcional de las conexiones entre bloques, así como de las conexiones entre el microcontrolador y los componentes discreto que conforman el bloque de control; reloj externo puerto de programación y el PIC18F4550.

Por último, el esquema eléctrico de la unidad de control será el siguiente;

Figura 3.3.28: diagrama eléctrico de la unidad de control

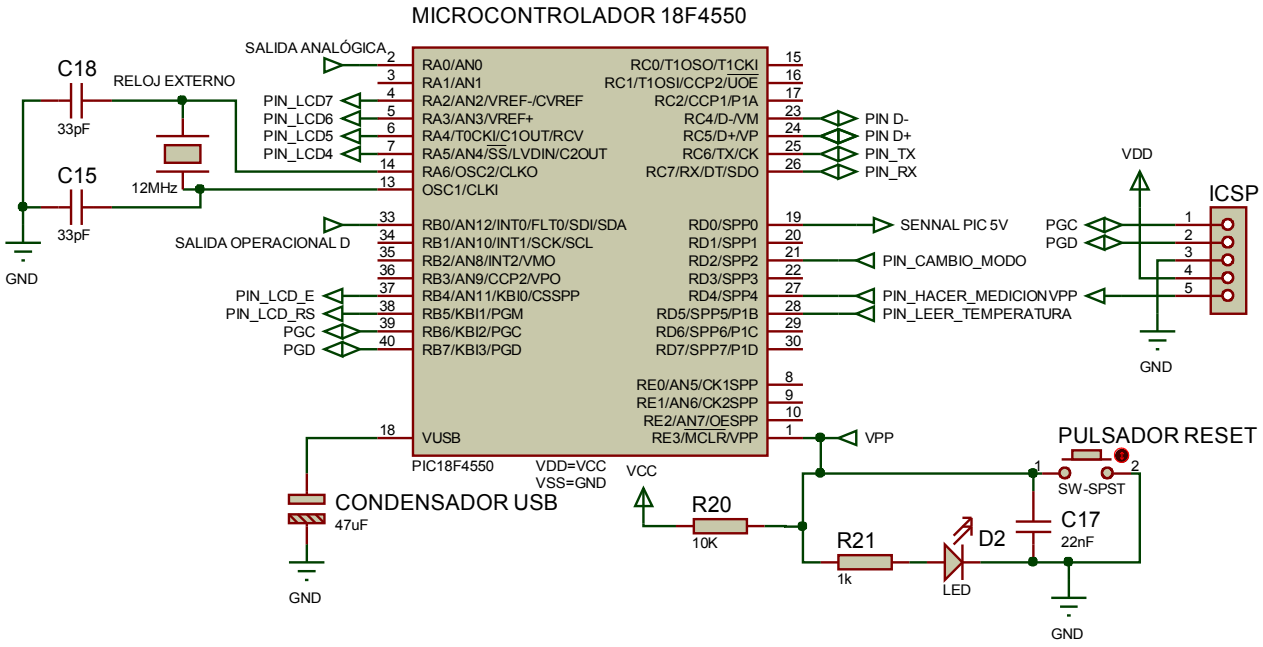


Diagrama eléctrico de la unidad de control formado por el microcontrolador y los componentes asociados; reloj externo, condensador USB, pulsador de reset y puerto de programación ICSP. Aparecen así mismo las conexiones con el resto de bloques funcionales y las tomas de alimentación.

En este esquema podemos ver las diferentes conexiones entre el microcontrolador y el resto de bloques funcionales, tanto de pines genéricos de entrada/salida como de pines dedicados a la comunicación con otros dispositivos, al oscilador externo, etc... Se ha añadido en led de estado asociado al pulsador de reset, de modo que el led estará encendido cuando reciba tensión desde cualquiera de las fuentes de alimentación. Si por el contrario no está conectado, el led se mostrará apagado, indicando así si está o no conectado a alguna fuente de tensión.

Con este último bloque termina al fin la descripción física del medidor de distancias. En el capítulo de *Anexos* se adjuntan el diagrama esquemático completo del medidor de distancias así como el PCB del mismo, realizado según la tecnología *through hole.*

4- DESCRIPCIÓN DEL FIRMWARE

En el capítulo 3 hemos explicado con detalle todo el diseño del dispositivo desde un punto de vista puramente físico, justificando la solución adoptada y describiendo cada uno de los bloques funcionales en los que se divide y los circuitos eléctricos que los implementan. En este capítulo nos centraremos en explicar el firmware que permite controlar nuestro medidor de distancias y que se encuentra integrado en la unidad de control.

Empezaremos en primer lugar por aportar un enfoque eminentemente práctico, al explicar la interfaz de usuario que engloba los protocolos USB y RS-232, así como la interfaz de usuario manual; ello permite utilizar el dispositivo sin necesidad de mayores conocimientos del código del programa. Una vez se conoce las funcionalidades del medidor, entraremos en detalle en explicar el firmware de nuestro medidor de distancias.

Continuaremos posteriormente por la parte de programación propiamente dicha, comenzando por justificar la elección del lenguaje de programación y el compilador empleado de entre la enorme gama de opciones disponible, así como el entorno de desarrollo utilizado, pasando posteriormente a detallar las principales características del lenguaje y el compilador empleado.

Ya por último nos centraremos en explicar en detalle el *firmware* (programa) incluido dentro del microcontrolador junto con todo lo que está con ello relacionado; gestión datos de entrada/salida desde los diferentes protocolos disponibles, uso de librerías, implementación de las diferentes técnicas de medición etc...

4.1 – Interfaz de usuario del medidor de distancia

En este apartado nos centraremos en explicar desde un punto de vista eminentemente práctico las funcionalidades que ofrece nuestro medidor de distancia mostrando las diferentes opciones de medición que ofrece, así como las diferentes posibilidades de uso, sea como periférico mediante RS-232 o USB, o su uso manual si se quiere usar de manera autónoma, las técnicas de corrección de errores de que dispone así como el interfaz de usuario para los 3 protocolos de comunicación de que dispone; puerto serie, RS-232 y manual.

4.1.1. Opciones de medición del dispositivo medidor de distancias

Detallamos aquí las diferentes opciones de medición que ofrece nuestro medidor, explicando las diferentes técnicas de medición así como las diferentes herramientas de que dispone el dispositivo para reducir en la medida de lo posible los errores asociados con la medición de distancias. Comenzamos explicando las 3 técnicas de que dispone nuestro medidor para calcular la distancia por tiempo de vuelo; medición simple, medición múltiple y medición múltiple con cálculo de mediana.

En la medición simple, el dispositivo calcula el tiempo de vuelo mediante el envío de un único tren de pulsos, obteniendo la distancia posteriormente a partir de la información termodinámica del entorno. Esta técnica permite calcular de manera rápida la distancia a un objeto, aunque obviamente estará sujeto a una incertidumbre relativamente alta (valor típico de ± 12 mm) debido a la imprecisión de la medición del tiempo de vuelo así como de la información ambiental. Resulta por ello más aconsejable realizar un conjunto de mediciones y trabajar con un conjunto amplio de medidas que nos permita obtener una información más fiable. Estas técnicas son las que hemos denominado *medicion_multiple*, y consisten en ejecución de un número variable de mediciones y su posterior almacenamiento en la memoria interna del microcontrolador. El número de mediciones es un parámetro interno del sistema que puede ser modificado a voluntad, pudiendo trabajar con hasta 64 mediciones por proceso.

El microcontrolador tiene una cierta capacidad de obtención de datos estadísticos, pudiendo por ejemplo devolver el valor de mediana del conjunto de resultados válidos, esto es, distintos de 0; este proceso es el que se conoce como medicion_multiple_media, y ofrecerá unos resultados mucho más acordes con los reales. Hay que señalar que en su modo manual de funcionamiento, el medidor de distancias realiza mediciones en modo múltiple con filtro de mediana, para garantizar una medición robusta cuando se usa como dispositivo de medición portátil.

Aun así es posible que se quiera trabajar con la serie completa de resultados de la medición, bien sea porque no interese obtener el filtro de mediana del conjunto de los valores o porque se quieran obtener información estadística adicional; para permitir esto se ha incluido el modo medicion_multiple_batch, el cual realiza un conjunto de mediciones sin tratamiento estadístico adicional, enviando el array de datos en bruto mediante RS-232 o USB.

4.1.2. Opciones de corrección de errores de medición

Como vimos en el capítulo 2, la velocidad del sonido depende en gran medida de la temperatura del medio, y en bastante menor medida de la humedad relativa. La inclusión de un sensor de temperatura se hacía crucial para obtener unos resultados precisos y consistentes en un rango amplio de condiciones ambientales. Se decidió así mismo prescindir de un sensor de humedad, debido a la poca influencia que tenía la velocidad del sonido frente a ello, optando por incluir de forma externa la humedad relativa por medio de un parámetro configurable por el usuario.

Así mismo, decidimos incluir un factor de corrección ligado a la medición de temperatura, de nombre “deriva_temperatura”, que permite una corrección de la temperatura obtenida por el sensor con una precisión de 0.25 ºC, así como un último factor de corrección, este ligado a los retrasos en la estimación del tiempo de vuelo.

Si recuperamos la ecuación termodinámica del sonido e introducimos el valor de la velocidad del sonido dentro de la ecuación que permite obtener la distancia, tenemos pues que

$$d = \frac{v_s * t}{2} = \frac{(331.58 + 0.62\,T + 0.015\,HR) * t}{2} \quad ,$$

donde v_s es la velocidad de propagación del sonido en m/s, T es la temperatura en ^{o}C, HR es la humedad relativa en escala porcentual 0-100% y t el tiempo en segundos.

Si así mismo añadimos los factores de corrección ligados a la temperatura y al retardo del tiempo de vuelo, tenemos que la ecuación queda

$d_{real} = d + \frac{(0.62\,T_{deriva} + 0.015\,HR) * t}{2} + d_{retardo}$, donde d es el valor de distancia teórico (sin corrección de errores), T_{deriva} es la deriva de temperatura en ^{o}C y $d_{retardo}$ es el retardo expresado en metros.

Se puede ver que a partir de esta ecuación que relaciona la distancia teórica obtenida con la distancia real podemos reducir enormemente los errores de medición, modificando la deriva de temperatura, la humedad relativa y el retardo asociado. Todos estos parámetros, como hemos dicho son perfectamente accesibles para el usuario en modo lectura y escritura cuando el medidor está funcionando como periférico mediante puerto serie o USB. Cuando el dispositivo funciona de modo autónomo, el usuario no será capaz de modificar o acceder a estos valores, debiendo emplear los interfaces anteriormente señalados para calibrar el dispositivo.

Como ya explicamos anteriormente, preferimos simplificar enormemente el uso del medidor de distancia en modo manual, limitándolo a realizar mediciones y lecturas de temperatura, y sacrificando como contrapartida las capacidades de calibración en este modo. Creímos que merecía la pena un diseño mucho más sencillo a incluir una interfaz completa en el modo manual, que permita la edición y al acceso a todas estas variables internas, algo que hubiera complicado y encarecido en gran manera el coste. Además, veremos más adelante que se puede cambiar en caliente del modo de funcionamiento manual a los modos USB o RS-232 y viceversa, facilitando enormemente la calibración del dispositivo.

4.1.3. Otras opciones de configuración

Además de poder modificar los parámetros ligados a la corrección de errores de medición, existe la posibilidad de acceder a parámetros que permiten mejorar la resolución, el alcance o la precisión de las medidas. Nos estamos refiriendo a los parámetros que determinan el número de pulsos del tren ultrasónico, el número de mediciones a efectuar cada vez que se solicita un proceso de medición múltiple y el lapso mínimo entre la emisión y recepción de ultrasonidos. La correcta asignación de estos parámetros es algo bastante importante, puesto que la modificación de estas variables conlleva una serie de ventajas y desventajas.

El **número de pulsos** determina la longitud del tren ultrasónico, y si bien un número alto de pulsos permite mejorar enormemente el alcance, hasta los 4-5 metros de máximo, existe una importante pérdida de resolución asociada a este aumento de longitud del tren de ondas. Recordemos que la velocidad del sonido la podemos fijar en torno a los 340 m/s. Como nuestra frecuencia de trabajo es de 40KHz, tenemos que la longitud total del tren de ondas, para un número *n* de pulsos, es de $\frac{340\,m/s}{40\mathrm{KHz}} n = 0.85\,cm * número\ de\ pulsos$

Hay que considerar que la resolución de la medición está íntimamente ligada a la longitud de onda con la que se hace trabajar el dispositivo; uniendo esto al hecho de que realmente no podemos saber cuál de los diferentes pulsos es el que ha retornado y ha sido considerado como eco, nos da una incertidumbre bastante importante a la hora de estimar la resolución de la medición. Para trenes de onda relativamente pequeños, de hasta 10 pulsos por tren de onda, podemos asumir que el eco de retorno se sitúa típicamente entre la mitad y el primer cuartil de los trenes de onda, con lo que la imprecisión ligada a la captura del umbral mínimo de eco es bastante baja, de en torno a 0.25 cm. Cuando la señal de retorno es mucho más larga y la distancia es elevada, en cambio, entran en juego factores como la débil recepción de la señal de retorno o el gran número de pulsos que pueden superar el umbral mínimo de ecos de retorno.

La correcta asignación del número de pulsos dependerá enormemente de las condiciones de trabajo; tamaño de los objetos a medir, rugosidad, distancia típica, etc... por lo que lo más recomendable es adoptar una actitud conservadora; trabajar con un número pequeño de pulsos para posteriormente ir aumentando si se observa este número que resulta insuficiente para distancias más largas u objetos más pequeños.

El **número de medidas** determina por otra parte cuántas mediciones se deberán realizar cada vez que se solicita un proceso de medición múltiple. En su uso como dispositivo manual, emplear un número elevado de mediciones permite una gran precisión en los resultados y no supone una desventaja importante, puesto que la frecuencia de medición no representa aquí un problema serio. Sin embargo, en aplicaciones donde sea crítico el tiempo de respuesta, un número elevado de muestras por cada conjunto de mediciones supone retrasos considerables.

En su caso más desfavorable, en el que el tiempo de vuelo para cada medida individual es máximo (no se recibe eco de retorno), tenemos que cada medición individual supone un retraso de unos $t = 2 * d / v_s = 35.3\, ms$, para una distancia máxima establecida en d=6 metros y una velocidad del sonido de $v_s = 340\, m/s$. Si consideramos un set de medidas elevado, de por ejemplo 50 mediciones, tenemos que el retraso entre el comienzo del proceso de medición hasta que se envían los resultado es de 1.8 segundos, un retraso considerable en aplicaciones donde se requiera una respuesta rápida del sistema.

Habrá que tener esto en cuenta cuando el medidor sea usado como periférico en aplicaciones donde el tiempo de respuesta sea crítico, debiendo trabajar con conjuntos de mediciones mucho más modestos para cumplir las expectativas de velocidad en la medición. La correcta asignación de este parámetro, al igual que el anterior, depende exclusivamente de las condiciones de trabajo, por lo que se ha decidido permitir su configuración a fin de hacer más versátil nuestro proyecto. Por defecto, y considerando que su uso principal como prototipo estará destinado al testeo y robustez de las mediciones, el número de mediciones está asignado por defecto a 64, el máximo admisible por la memoria del microcontrolador.

El **lapso mínimo** determina por último el tiempo de espera entre la emisión del tren de pulsos ultrasónicos y el comienzo de la recepción de los mismos. La existencia de este retardo responde a un problema que se encuentra en muchas aplicaciones de sensores de ultrasonidos, y que se denomina *resonancia;* básicamente lo que sucede es que como tanto el emisor como el receptor trabajan a la misma frecuencia, el envío por parte del emisor del tren de pulsos ultrasónicos provoca en el receptor la inmediata excitación del receptor ultrasónico, corriendo el riesgo de que esta excitación supere el umbral de eco de retorno y se obtengan resultados de distancia falseados y de valores lógicamente mucho más bajos del valor que correspondería de haber recibido verdaderamente un eco de retorno.

Existen muchas posibles soluciones a este problema, que involucran el uso de material aislante sonoro entre el receptor y el emisor, aumentar la separación entre los mismos o usar transductores ultrasónicos con una mucho mayor directividad, para minimizar la transmisión de ondas sonoras en direcciones preferentes.

En nuestro caso, para el circuito PCB desarrollado tenemos que basta con imponer un lapso mínimo de 400us para eliminar completamente el eco espurio debido a la resonancia; ello equivale a que el dispositivo medidor no pueda realizar mediciones de objetos situados a distancias menores a $d = \frac{v_s * t}{2} = 340\, m/s * 400.10^6\, s/2 = 6.8\, cm$;

esta distancia mínima que nos imponte el medidor de distancia nos pareció adecuada, y más si se considera que realmente, a distancias menores a los 6.8 cm la longitud del tren de ondas alcanza el mismo tamaño que la distancia entre el propio dispositivo y el objeto a medir, con lo que los datos se verían enormemente distorsionados.

En cualquier caso, y si el montaje del dispositivo permitiera atenuar la resonancia, se podría modificar este valor para obtener una distancia mínima medible aún más pequeña, o en el caso contrario en el que la resonancia fuera demasiado elevada se podría aumentar el valor de espera hasta alcanzar límites tolerables.

4.1.4. interfaz de usuario para la comunicación USB-CDC y RS-232.

Como ya vimos en el apartado ***3.3.6*** el uso de la interfaz USB- CDC tenía como principal ventaja que para el usuario del dispositivo tanto uno como otro protocolo funcionan de idéntica manera, esto es, ambos se comportan como un puerto serie RS-232 al uso, por más que uno sea realmente un puerto serie y otro un puerto USB bajo el protocolo CDC.

Ello permite emplear una misma interfaz de usuario para el usuario, haciendo transparente para el usuario la comunicación con el medidor de distancia sea cual sea el modo de conexión. Recordemos que a nivel de usuario la configuración de una comunicación USB-CDC implica en nuestro caso la creación de un puerto COM virtual, con características idénticas al que tendría un verdadero puerto COM asociado a la conexión RS-232. Cualquier programa que permita trabajar con nuestro aparato bajo un puerto determinado COM real podrá trabajar con el puerto COM virtual sin más que cambiar el número asociado al puerto COM.

En los anexos se incluirá un script de matlab que permite trabajar con el medidor de distancias, permitiendo leer y configurar todos los parámetros y realizar todos los tipos de medición. Incluimos por tanto la interfaz de uso, debiéndose tener en cuenta que esta interfaz es común para ambos protocolos de comunicación.

4.1.4.A. Opciones de medición bajo USB y RS-232

Aunque aparezcan 6 opciones de medición, en realidad se corresponden con las 3 técnicas de medición vistas anteriormente, estando cada una de ellas disponibles con dos posibles formatos de salida de datos; formato numérico big-endian o formato string. El tamaño de los datos de salida será variable, dependiendo tanto del número de medidas por proceso como de si la salida de los datos se ha especificado en uno u otro formato.

Figura 4.1.1: interfaz de uso para RS-232 y USB; opciones de medición

Identificador primario	Id. adicionales	Acción
M	**Ninguno**	Calcula el tiempo de vuelo del pulso ultrasónico. Devuelve por USB/RS-232 el carácter identificador "**M**" seguido de la **distancia** en forma de cadena de caracteres numéricos.
m	**Ninguno**	Calcula el tiempo de vuelo del pulso ultrasónico. Devuelve por USB/RS-232 el carácter identificador "**m**" seguido de la **distancia** en formato de 16 bits *big endian*
N	**Ninguno**	Efectúa un lote de mediciones, cuyo número estará determinado por la variable *numero_medidas*. Devuelve por USB/RS-232 el número de medidas erróneas (sin eco de retorno), seguido por **la mediana** del conjunto de mediciones, ambos datos en formato de cadena de caracteres numéricos, precedido ambos por el carácter identificador "**N**".
n	**Ninguno**	Efectúa un lote de mediciones, cuyo número estará determinado por la variable *numero_medidas*. Devuelve por USB/RS-232 el número de medidas erróneas (sin eco de retorno), seguido por **la mediana** del conjunto de mediciones, ambos datos en formato de 16 bits *big endian,* precedido ambos por el carácter identificador "**n**"
B	**Ninguno**	Efectúa un lote de mediciones, cuyo número estará determinado por la variable *numero_medidas*. Devuelve por el puerto serie el **conjunto completo** de medidas, en formato de cadena de caracteres, precedidos por el carácter identificador "**B**"
b	**Ninguno**	Efectúa un lote de mediciones, cuyo número estará determinado por la variable *numero_medidas*. Devuelve por el puerto serie **el conjunto completo** de medidas, en formato de 16 bits *big endian*, precedidos por el carácter identificador "**b**"

Para realizar un proceso de medición, como podemos ver, bastará con enviar el byte correspondiente y esperar al retorno de datos desde el dispositivo.

4.1.4.B. Opciones de lectura del interfaz de usuario bajo RS-232 y USB

En el caso de requerir la lectura de algún parámetro, se exige el envío de dos bytes; una para indicar lectura, llamado identificador primario, y otro para concretar qué parámetro leer, de nombre éste último identificador secundario. Se ha decidido fijar un tiempo de espera máximo entre la recepción de el byte primario y el secundario de 5 milésimas de segundo, lo que permite velocidades de transmisión mínimas de 2400 baudios. Si en ese intervalo de tiempo no se recibe un byte, o este byte no se corresponde con ninguna de las secuencias de medición preestablecidas, se saldrá automáticamente del menú de lectura sin que llegue a producirse ninguna lectura.

Figura 4.1.2: interfaz de uso para RS-232 y USB; opciones de lectura

Id. primario	Id. 2º	Petición	Formato de salida
R	**L**	Lectura del tiempo máximo de espera (en decenas de microsegundo) entre la emisión y recepción de ultrasonidos.	Envío de la secuencia **RL[byte]**, donde en [byte] se envía el ***lapso máximo***
R	**D**	Lectura del retardo asociado a las mediciones de distancia (en mm).	Envío de la secuencia **RD[byte1][Byte2]**, donde en [byte1][Byte2] se envía el ***retardo*** en formato *big-endian*
R	**N**	Lectura del número de pulsos ultrasónicos enviados por el emisor.	Envío de la secuencia **RN[byte]**, donde en [byte] se envía el ***numero de pulsos***
R	**Z**	Lectura de la deriva asociada a le medición de temperatura (en centésimas de ºC).	Envío de la secuencia **RZ[byte1][Byte2]**, donde en [byte1][Byte2] se envía la ***deriva de temperatura*** en formato *big-endian*
R	**H**	Lectura de la humedad relativa (en escala 0-255).	Envío de la secuencia **RH[byte]**, donde en [byte] se envía la humedad relativa
R	**K**	Lectura de del número de medidas a realizar cuando se selecciona un método de medición múltiple.	Envío de la secuencia **RK[byte]**, donde en [byte] se envía el ***número de medidas***
R	**T**	Lectura del valor actual de temperatura (centésimas de ºC).	Envío de la secuencia **RT[byte1][Byte2]**, donde en [byte1][Byte2] se envía la ***temperatura*** en formato *big-endian*
R	**V**	Lectura del valor actual de la velocidad del sonido cm/s).	Envío de la secuencia **RV[byte1][Byte2]**, donde en [byte1][Byte2] se envía la ***velocidad del sonido*** en formato *big-endian*

Dado que la comunicación es en modo serie con 8 bits de datos, los datos que aparecen como de 16 bits se enviarán al pic en una secuencia de dos bytes en formato big endian (primero el más significativo).

4.1.4.C. Opciones de configuración del interfaz bajo RS-232 y USB.

Aquí, además de requerir los bytes identificadores primarios y secundarios para poder seleccionar un parámetro interno concreto, se requieren de entre 1 y 2 bytes para actualizar así el valor de la variable.

Figura 4.1.3: interfaz de uso para RS-232 y USB; opciones de configuración

Id. primario	Id. 2º	Bytes de datos adicionales	Acción de escritura
S	**L**	**[Byte]**	Configuración del tiempo máximo de espera (en decenas de microsegundo) entre la emisión y recepción de ultrasonidos. Si se asigna un lapso máximo igual a 0, se asignará al lapso máximo el valor por defecto, 400 us.
S	**D**	**[Byte1][Byte2]**	Configuración del retardo asociado a las mediciones de distancia (en mm). Si el valor introducido es igual a 0, se asignará al retardo el valor por defecto, 25 mm
S	**N**	**[Byte]**	Configuración del número de pulsos ultrasónicos enviados por el emisor. Si el número de pulsos recibidos es 0 o mayor que 128, se asignará el número de pulsos por defecto, 8 pulsos.
S	**Z**	**[Byte1][Byte2]**	Configuración de la deriva asociada a le medición de temperatura (en centésimas de ºC). Si la deriva de Tª supera los ±25.00 ºC, el valor que se asignará por defecto a la deriva de temperatura, 0.00ºC
S	**H**	**[Byte]**	Configuración de la humedad relativa (en escala 0-255).
S	**K**	**[Byte]**	Configuración del número de medidas a realizar cuando se selecciona un método de medición múltiple. Si el número de medidas recibido es 0 o mayor que el valor máximo admisible, asignará el valor por defecto,64

Se debe cumplir lo mismo que en el caso anterior, esto es, un lapso máximo de tiempo entre los diferentes bytes que forman parte de la secuencia de configuración de 5 ms, o lo que es lo mismo, la secuencia de configuración debe enviarse como en el caso anterior a velocidades superiores a los 2400 baudios.

Lo que se ha visto hasta ahora han sido las diferentes secuencias válidas para realizar acciones de lectura, escritura o medición. Además del envío de datos por los puertos RS-232 y USB-CDC, el display LCD también muestra las acciones de lectura, medición y escritura, pudiendo servir para testear el buen funcionamiento de la comunicación puerto serie-USB. En el apartado 4.4.6 se muestra una lista detallada de todas las cadenas de caracteres que se muestran por el LCD en función de cada petición posible de las que pertenecientes tanto a los dos interfaces que acabamos de explicar, RS-232 y USB, como del interfaz manual.

4.1.5. Uso manual del dispositivo.

En los apartados anteriores hemos examinado las diferentes opciones relativas a las diferentes opciones de medición, presentación de los resultados así como los diferentes parámetros internos que pueden ser leídos o editados. En este apartado nos centraremos en la interfaz de uso manual, esto eso, cuando se activa su uso como dispositivo autónomo y se cancelan las conexiones por RS-232 o USB.

Bajo el modo de funcionamiento manual existen solo 3 tipos de acciones que podrá realizar el medidor de distancia; cambiar entre el modo manual de funcionamiento al modo automático (RS-232/USB), lectura de temperatura o proceso de medición múltiple con filtro de mediana. Existen por tanto en el circuito 3 tipos de actuadores que implementan estas acciones; un pulsador para la lectura de temperatura, otro pulsador para la realización de mediciones y un interruptor para implementar el cambio de modo manual/automático.

Figura 4.1.5: Comandos de la interfaz manual y acciones asociadas

Elemento de la interfaz manual	Acción
Botón de medición	Mientras está pulsado, el dispositivo realiza continuamente mediciones con filtrado de mediana. Mostrará por el LCD la mediana de los valores muestreados, así como el número de medidas fallidas.
Botón de temperatura	Mientras esté pulsado, el dispositivo muestra realiza continuamente mediciones de la temperatura ambiente y las muestra por el LCD
Interruptor de modo manual/ automático	Cambia entre el modo manual y el modo automático. El LCD refleja el cambio entre uno u otro modo de funcionamiento.

En su modo manual, el LCD se convierte en el único modo de visualización de que dispone el medidor de distancia. Incluimos aquí la lista de estados posibles del medidor y las cadenas de caracteres asociadas que se muestran por LCD:

Figura 4.6 : lista de cadenas de caracteres del LCD en función del estado del medidor en su modo manual.

Estado del medidor	Pantalla LCD	
Inicio:buscando interfaces válidos	Conectando...	Cuando encuentra una conexión válida, el mensaje pasa a ser: Hecho.
Conexión efectuada con el modo manual	Modo manual	
Medición de distancia en modo manual	D(XXX) =YYYY mm	YYYY=valor de mediana del array de distancias. XXX=número de medidas erróneas (valores nulos) obtenidos.
Lectura de temperatura en modo manual	Temp(oC)=XXX.XX	XXX.XX = temperatura/100

4.1.6. Cambio de las interfaces de comunicación.

Antes de finalizar este apartado debemos explicar brevemente cómo se realiza el cambio de la interfaz de comunicación con la que nos comunicaremos con el dispositivo.

Aunque en el apartado 4.4 nos centraremos en cómo está implementado a nivel interno el gestor de cambio de interfaces de comunicación, aquí daremos una explicación desde el punto de vista del usuario, mostrando qué hay que hacer para hacer que nuestro medidor de distancia pueda funcionar tanto bajo USB como RS-232 o en modo manual, cambiando así mismo de interfaz de comunicación sin necesidad de reiniciar el medidor de distancia.

Al inicio, el dispositivo intentará conectarse al primer interfaz de comunicación disponible, sea USB, RS-232 o manual. En el caso del interfaz USB, el medidor solamente se conectará a este protocolo cuando el proceso de configuración del mismo haya finalizado. Esto puede verse en el display LCD, mostrando en todo momento el estado actual del medidor y si está o no conectado mediante USB.

En el caso del interfaz RS-232, el medidor no activará esta interfaz de comunicación a menos que desde le puerto serie se le envíe la llamada señal de activación del puerto serie, que tiene por defecto el valor “y”; así, para poder activar la comunicación entre cualquier dispositivo y nuestro medidor, el dispositivo deberá enviar continuamente la señal de activación del puerto serie, a fin de que nuestro medidor la reconozca y active en consecuencia el puerto serie. Puede ocurrir que, estando el dispositivo ya conectado mediante USB se active la comunicación mediante RS-232; esta es una opción que hemos permitido explícitamente para poder permitir un modo de funcionamiento mixto quedando a responsabilidad del usuario el combinar correctamente las dos conexiones de forma simultánea.

Por último, hay que explicar el cambio al modo manual. Este cambio se realiza con el *interruptor de modo manual/automático*, de modo que cuando el interruptor pasa a modo manual automáticamente pasan a desconectarse los interfaces de comunicación RS-232 o USB, respondiendo solo a los botones de la interfaz manual. Para volver a usar los interfaces de comunicación Usb o RS-232 se deberá apagar el *interruptor de modo manual/automático*, recomenzando el proceso de búsqueda de interfaces de comunicación válidas. La forma más sencilla de comprobar el cambio de interfaces de comunicación será como ya hemos dicho observando si el cambio de interfaz se refleja en las pantallas LCD; adjuntamos una lista de las cadenas de caracteres que se muestran en el LCD asociadas a cambios del interfaz de comunicación.

Figura 4.1.7: lista de las cadenas de texto del LCD asociadas a cambios de la interfaz de comunicación.

Acción del medidor	Pantalla LCD	
Inicio:buscando interfaces válidos	Conectando...	Cuando encuentra una conexión válida, el mensaje pasa a ser: Hecho.
Conexión efectuada con el interfaz USB	Conexión USB	
Conexión efectuada con el interfaz RS-232	Conexión RS-232	
Conexión efectuada con los interfaces USB + RS-232	Conexión USB+232	
Conexión efectuada con el modo manual	Modo manual	

Con lo explicado en este capítulo se tienen los suficientes conocimientos a nivel de usuario como para poder utilizar el aparato desarrollado bajo todos los modos de funcionamiento, accediendo a los parámetros internos y pudiéndolos modificar. En el apartado siguiente ofreceremos una explicación detallada del firmware, explicando a nivel funcional cada una de las opciones de uso y configuración señaladas en este apartado.

4.2 – Elección del lenguaje de programación y del compilador.

El microcontrolador elegido, miembro de la familia 18F, forma parte de la gama de microcontroladores de 8 bits desarrollada por la empresa Microchip, que viene siendo durante años líder mundial en esta franja de productos. Dada su enorme popularidad existen multitud de compiladores y lenguajes de programación que nos permiten la creación de aplicaciones para estos microcontroladores de 8 bits, de uso tanto gratuito como de pago, y bajo licencias de uso y explotación libre o propietario.

La primera elección fue decantarse por algún lenguaje alto nivel para realizar nuestra aplicación, dada la enorme complejidad que conlleva un programa de estas características, y más en un caso como el nuestro en el que se debe configurar un interfaz de comunicaciones USB mediante CDC. Habría resultado una tarea no solo enormemente compleja sino en parte inútil, ya que la mayoría de los lenguajes de programación de alto nivel incluyen en sus librerías estándar este tipo de interfaces, reduciendo el esfuerzo de programación al uso de unas pocas funciones. Por ello, y porque el objetivo de nuestro proyecto no está centrado en el desarrollo de software sino que tiene un carácter multidisciplinar, decidimos ahorrar esfuerzo y tiempo de programación y emplear alguno de los lenguajes de programación de alto nivel disponibles.

Habiendo decidido ya el uso de un lenguaje de alto nivel, como C, Pascal o Basic, decidimos por decantarnos por usar un lenguaje de programación con un compilador asociado que fuese gratuito y operase bajo licencia libre en su sentido más amplio posible. En primer lugar porque opinamos que proyectos como este, de carácter eminentemente divulgativo y realizados en el mundo universitario deben emplear programas y plataformas de código libre, por lo que

decidimos actuar en consecuencia y realizar nuestro proyecto usando herramientas de código libre. En segundo lugar porque el lenguaje de programación elegido, así como el compilador, nos parecieron unas herramientas excelentes e incluso superiores a algunas de las alternativas de pago, y deseábamos emplear este proyecto como plataforma para popularizar su uso.

Así pues, nos decantamos por un lenguaje de programación llamado **jalv2** ,basado levemente en Pascal y que forma parte de los lenguajes de alto nivel, y cuyo compilador permite trabajar bajo una licencia de código abierto. Este compilador está diseñado para optimizar el tamaño del código resultante de nuestro programa, algo especialmente útil en el campo de la programación de microcontroladores, donde el espacio disponible para almacenar el programa es un factor determinante.

El principal atractivo que vimos a este lenguaje de programación fueron sus librerías, que permiten manejar la gran mayoría de periféricos con los que deberá interaccionar generalmente un microcontrolador; gracias a ellas podemos utilizar todos los protocolos de comunicación incluidos en los microcontroladores de la gama de 8 bits, tales como CAN, USB, RS-232, SPI, I2C, así como los periféricos internos como los conversores analógico-digitales, PWM, etc... Dispone así mismo de un gran número de librerías dedicadas a controlar los más diversos periféricos externos, tales como sensores de temperatura, humedad, CO2, pantallas de cristal líquido, motores y dispositivos de almacenamiento de memoria.

Otro aspecto importante a destacar es que a pesar de no ser un compilador soportado por la empresa *Microchip*, fabricante de los microcontroladores de 8 bits como los de nuestro proyecto, el compilador a día de hoy permite trabajar con prácticamente todos los microcontroladores que tiene microchip en el mercado. Lógicamente, el microcontrolador ***pic18f4550*** usado en este proyecto está plenamente soportado por el compilador, así como el resto de modelos de la misma sub-familia de 8 bits del tipo 18f.

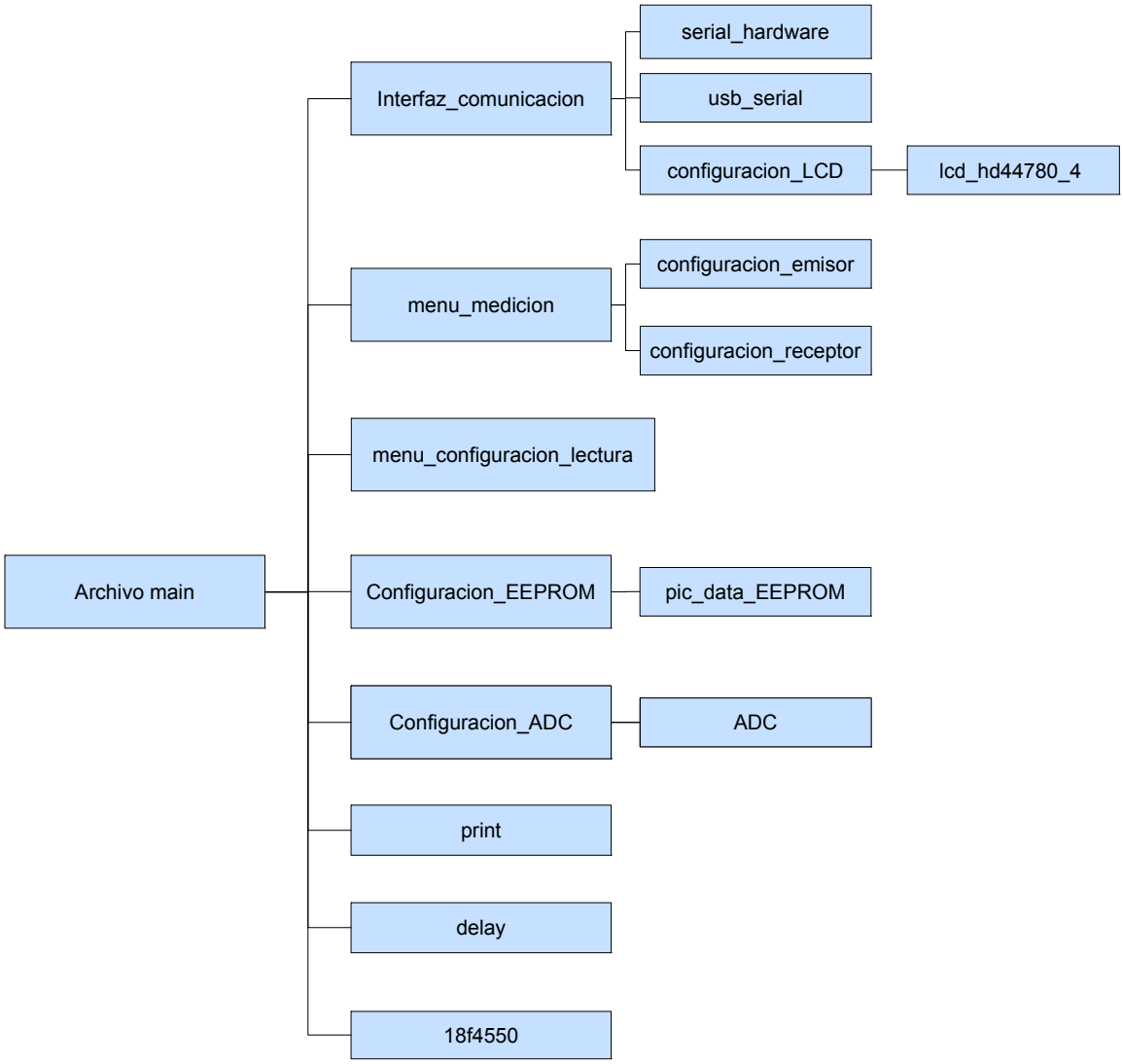
Por último, la licencia de uso y explotación del compilador y de las librerías para jalv2 responden a la licencia freeBSD, que permiten no solamente su uso gratuito sino su posible comercialización sin liberar el código fuente de la aplicación; esta licencia, mucho más permisiva que licencias similares como la GPL, permite por tanto una enorme libertad a la hora de realizar

proyectos comerciales con aplicaciones que usen este lenguaje.

Estas fueron por tanto las razones que nos llevaron a elegir este lenguaje de programación para nuestro proyecto. A día de hoy existe un único compilador para el lenguaje de programación ***jalv2***, llamado ***jalv2 compiler*** y que está siendo desarrollado por un grupo diferente del que diseñó el lenguaje de programación jalv2. Así mismo existe una activa comunidad encargada de las librerías adicionales para jalv2, las cuales están reunidas bajo el nombre de ***jallib*** y que facilitan el uso de todos los periféricos comentados anteriormente. A fecha de hoy ambas comunidades han sacado la versión **2.4.n** del compilador ***jalv2 compiler*** y la versión ***0.5*** de ***jallib***, siendo éstas las versiones con las que ha sido desarrollada la aplicación.

4.3 - Descripción general de las librerías

Figura 4.3.1: distribución jerárquica de las librerías



El fichero principal de nuestro código fuente tiene como nombre main.jal, y en él se declaran el resto de ficheros y librerías estándar que precisa el programa, que tienen una distribución como la que aparece en el diagrama:

- **Archivo main.jal:**

Este es el fichero principal de nuestro programa; en él se declaran el resto de librerías, se configuran los fusibles y los parámetros principales del microcontrolador y se define el funcionamiento de la rutina principal.

- **Archivo delay.jal:**

Esta librería forma parte del conjunto de librerías estándar de jalv2 y contiene funciones que permiten establecer retardos precisos de microsegundos, milisegundos o segundos

- **Archivo print.jal:**

Esta librería se encarga de dar formato a los datos que reciben periféricos internos o externos como USB, LCD o RS-232, sea en binario, string, hexadecimal, decimal, etc..., y forma así mismo parte del conjunto de librerías estándar.

- **Archivo configuracion_EEPROM.jal:**

Esta librería definida por el usuario declara las posiciones de memoria EEPROM en las que estarán almacenadas las variables globales principales (temperatura, número de pulsos, retardo, velocidad del sonido) , y a continuación llama a la librería estándar **pic_data_eeprom.jal.**

- **Archivo pic_data_eeprom.jal:**

Esta librería forma parte del conjunto de librerías estándar y se encarga de definir las posiciones de memoria de la eeprom, así como de incluir funciones de escritura y lectura de la EEPROM.

- **Archivo configuracion_ADC.jal:**

Esta librería definida por el usuario se encarga de configurar el periférico interno ADC (conversor analógico-digital) así como de definir funciones encargadas de los cálculos matemáticos para obtener la temperatura y la velocidad del sonido a partir de la información proveniente de los sensores analógicos. Para realizar todo esto se sirve de la librería estándar **ADC**, incializando posteriormente el conversor analógico-digital.

- **Archivo adc.jal**

Esta librería es llamada por **configuracion_ADC.jal** y forma parte del conjunto de librerías estándar. Su función es la de configurar el conversor analógico digital, incluyendo funciones y variables que permiten inicializar el módulo ADC, establecer la precisión de la conversión, especificar los pines analógicos a utilizar e incluye así mismo funciones de lectura de los datos analógicos en diferentes formatos.

- **Archivo interfaz_comunicacion.jal:**

Bajo esta librería definida por el usuario se agrupa todo lo relativo a los interfaces de comunicación, tanto USB como RS232 como a la interfaz manual. Por lo tanto, bajo esta librería se agruparán todas las funciones, variables, constantes y librerías asociadas a la comunicación con el puerto serie, puerto USB mediante CDC, configuración de la botonera externa y configuración y envío de datos al LCD. En primer lugar se configura la interfaz manual, definiendo y configurando los pines digitales asociados a la botonera de la interfaz manual (medicion de distancia, lectura de temperatura y activación de la interfaz manual), para posteriormente configurar el periférico externo lcd, mediante la llamada a la librería **configuracion_LCD.jal**. Posteriormente se encarga de la configuración del interfaz USB y RS232, que como explicamos anteriormente trabajan de manera similar, siendo por lo tanto las funciones encargadas de la lectura y escritura las mismas para trabajar con el puerto serie como para USB.

Se incluyen también funciones que permiten la correcta selección de la interfaz de comunicación al inicio (manual, RS232, USB o mixto), además de funciones que permiten el cambio de la interfaz de comunicación, bajo demanda del usuario y/o dependiendo del estado de los puertos de comunicación.

Esta librería se encarga por último de configurar el temporizador *timer_usb* y las interrupciones asociadas encargadas del refresco de la señal USB así como del envío y recepción de datos, sea por puerto serie o por USB. Para realizar todas estas funciones, esta librería precisa a su vez llamar a las librerías estándar **usb_serial.jal** y **serial_hardware.jal** para configurar respectivamente los periféricos internos USB y USART.

- **Archivo configuracion_LCD.jal**

Esta librería es llamada por la librería definida por el usuario **interfaz_comunicacion.jal** , y es así mismo una librería definida por el usuario, cuya función es la de configurar los pines digitales encargados de la comunicación con el display de cristal líquido (LCD), de implementar las funciones de escritura sobre el display así como de incluir las cadenas de caracteres que se mostrarán en la pantalla del LCD. Para realizar todo esto se vale de la librería estándar **lcd_hd44780_4.jal**.

- **Archivo lcd_hd44780_4.jal**

Esta librería forma parte del conjunto de librerías estándar, y se encarga del control de LCD alfanuméricos que usen el protocolo basado en el popular chip hd44780. Esta librería en concreto trabaja en modo de puerto paralelo con 4 bits, e incluye todas las funciones necesarias para la configuración y uso del display.

- **Archivo usb_serial.jal**

Esta librería es llamada a su vez por la librería definida por el usuario **interfaz_comunicacion.jal** , y pertenece al conjunto de librerías estándar. Concretamente se encarga de la configuración del periférico interno USB, que controla de acuerdo con el protocolo CDC. Incluye todas las funciones de escritura, lectura, configuración, estado e inicialización necesarias para configurar y comunicarse con el periférico.

- **Archivo serial_hardware.jal**

Como en el caso anterior, esta librería es llamada a su vez por la librería definida por el usuario **interfaz_comunicacion.jal** , perteneciendo así mismo al conjunto de librerías estándar. Esta en concreto se encarga de la configuración y manejo del puerto RS-232 en modo asíncrono, incluyendo funciones que permiten su configuración así como funciones de entrada y salida de datos.

- **Archivo menu_medicion**

Esta librería agrupa todas las funciones, constantes y variables necesarias para implementar el proceso de medición, incluyendo así mismo las funciones encargadas del proceso de emisión como de recepción de ultrasonidos, incluidas dentro de las librerías

configuracion_emision_ultrasonidos.jal y **configuracion_recepcion_ultrasonidos.jal**. En lo que el proceso de medición se refiere, se incluyen aquí las funciones encargadas de la obtención del tiempo de vuelo así como las que a partir del tiempo de vuelo permiten obtener la distancia al objeto, teniendo en cuenta las características termodinámicas del medio. Aquí se encuentran por tanto las funciones encargadas de realizar una única medida así como las que realizan un conjunto preestablecido de mediciones y obtienen la mediana del conjunto de medidas.

- **Archivo configuracion_emision_ultrasonidos.jal**

Esta librería, llamada a su vez por la librería menu_medicion.jal, es una librería definida por el usuario que se encarga de configurar correctamente las subrutinas encargadas de enviar al emisor ultrasónico un pulso cuadrado con una frecuencia y número de pulsos específicos. Para ello esta librería se encarga de configurar el temporizador timer_pwm y de definir las interrupciones asociadas a ello así como de especificar la frecuencia y número exacto de pulsos, además de configurar correctamente el pin digital de salida encargado del envío de la señal cuadrada.

- **Archivo configuracion_recepcion_ultrasonidos.jal**

Esta librería es también llamada como en el caso anterior por la librería menu_medicion.jal, y es así mismo una librería definida por el usuario que se encarga del proceso de recepción del eco ultrasónico, encargándose por tanto de la configuración del timer asociado a la medicion del tiempo de vuelo (TOF), de nombre timer_lapso_maximo. Por ello, se encarga también de implementar las interrupciones, funciones, variables y constantes que permiten obtener el TOF. Como hemos dicho, esta librería hace uso del temporizador timer_lapso_maximo, que sirve tanto para obtener el tiempo de vuelo como para abortar el proceso de recepción de la señal ultrasónica en caso de que el lapso de tiempo transcurrido sea demasiado largo. También requiere de una interrupción externa, de nombre int_RECEPTOR, que permite parar el temporizador cuando efectivamente se ha recibido un eco; por ello, esta librería también configura tanto el pin digital de entrada encargado de la recepción del eco como de la propia interrupción externa asociada a dicho pin.

- **Archivo menu_configuracion_lectura**

Esta librería agrupa todas las subrutinas encargadas tanto de configurar los parámetros principales del dispositivo, como el número de pulsos, retardo mínimo entre pulsos, error de deriva de distancia, etc... como las subrutinas encargadas de leer estos mismos parámetros.

- **Archivo 18f4550**

Esta librería forma parte del conjunto de librerías estándar, y resulta fundamental para el funcionamiento del programa. Se encarga de definir entre otras cosas los registros, pines y puertos específicos del microcontrolador pic18f4550, permitiendo así usar librerías genéricas para toda la familia de microcontroladores.

4.4 – Descripción funcional del firmware

La aplicación de software que debía ser incluida en el medidor de distancia tenía que encargarse tanto de las tareas de medición propiamente dichas como de la de establecer comunicación con el exterior a través de cualquiera de los interfaces de comunicación existentes, ya sea para recibir o enviar peticiones de medición y captura de datos ambientales o para configurar y calibrar el aparato. Pasamos pues a describir el firmware desarrollado, primero de manera general desde un punto de vista funcional, para entrar después en detalle y pasar posteriormente a incluir el código fuente del programa. Así, a nivel funcional podemos distinguir entre cinco diferentes bloques; medición de distancia, obtención de datos ambientales, comunicación con el usuario por medio de USB o RS-232, modo de funcionamiento manual y gestión de cambio de interfaces. Pasamos a continuación a describir cada uno de los bloques nombrados

4.4.1 Captura de datos ambientales:

Esta parte del firmware agrupa las librerías, procedimientos, variables, constantes y funciones encargados del proceso de la captura de los parámetros ambientales (temperatura y humedad) y la obtención de la velocidad del sonido a partir de estos datos, y lo encontramos perfectamente encapsulado dentro de la librería ***configuracion_ADC.jal***.

Como hemos explicado anteriormente, el valor de temperatura se obtiene a partir de la tensión de salida del sensor MCP9701A mediante el uso del módulo conversor analógico digital (llamado para abreviar A/D) del microcontrolador PIC18f4550. Así pues, se precisa activar y configurar el módulo A/D, así como configurar la entrada del microcontrolador asociada al sensor de temperatura como entrada analógica. De ello se encarga la librería ***adc.jal***, una librería estándar de jalv2 que permite la configuración del módulo A/D, permitiendo su uso con unas pocas funciones. En lo que respecta a la configuración, el módulo A/D y el canal analógico asociado al sensor han sido configurado con los siguientes parámetros:

Figura 4.4.1: configuración del módulo ADC

parámetro	valor
resolución	Alta (10 bits)
Número de canales analógicos	1
Tensión de referencia externa	NO
Valor de la tensión de referencia.	5 Voltios (interna)
Canal asociado al sensor MCP9701A	Canal AN0/RA0

Las funciones incluidas en esta librería permiten obtener la temperatura y la velocidad del sonido a partir del valor de tensión proveniente del sensor de temperatura, la humedad relativa y el error de deriva de la temperatura. Como hemos visto en el capítulo anterior, el valor de humedad relativa no se obtiene por medio de ningún sensor asociado al medidor de distancia, sino que será un parámetro configurable por el usuario desde cualquiera de los interfaces de comunicación y que mejorará la precisión de las mediciones. Lo mismo sucederá con la corrección de la deriva de la temperatura, que también es un parámetro configurable por medio de los

interfaces RS-232 y USB.

Recordemos que la ecuación que relaciona el valor de tensión con la temperatura es: $T_{100}=25\,V-2048$, con T medida en centésimas de grado y V medida en escala de 10 bits (de 0 a 1024). Así mismo, la dependencia de la velocidad del sonido con respecto de la humedad y la temperatura viene dada, como vimos antes, por la ecuación

$V_s=31888+(V*31)/2+(HR*3)/5$, donde la velocidad del sonido V_s está expresada en cm/s, V es el voltaje de salida del sensor de temperatura en escala de 10 bits y HR es la humedad relativa en escala de 8 bits. Así mismo, la temperatura real venía dada, corrigiendo el efecto de deriva del sensor de temperatura, por la ecuación $T_{100\,\text{real}}=25\,V-2048+T_{100\,\text{deriva}}$, mientras que la velocidad del sonido real, por su parte, quedaba de la forma, $V_{s\,\text{real}}=31888+(V*31)/2+(HR*3)/5+(T_{100\,\text{deriva}}*5)/8$.

Partiendo de estas ecuaciones se implementaron las funciones que aparecen en la siguiente tabla:

Figura 4.4.2: funciones asociadas al conversor analógico digital

Nombre de la función	Acción
convertir_ADC_temperatura	Convierte el valor de tensión digitalizado proveniente del sensor de temperatura en unidades de temperatura(centésimas de ºC), de acuerdo con la ecuación $T_{100}=25\,V-2048$
convertir_ADC_vel_sonido	Convierte el valor de tensión digitalizado proveniente del sensor de temperatura en unidades de velocidad (cm/s) , de acuerdo con la ecuación $V_s=31888+(V*31)/2$
obtener_temperatura_cent	Calcula el valor real de la temperatura, corrigiendo el efecto del error de deriva de la temperatura; $T_{100\,\text{real}}=T_{100}+T_{100\,\text{deriva}}$
obtener_vel_sonido_cent	Calcula el valor real de la velocidad del sonido, corrigiendo el efecto del error de deriva de la temperatura y añadiendo el efecto de la humedad relativa; $V_{s\,\text{real}}=V_s+(HR*3)/5+(T_{100\,\text{deriva}}*5)/8$

4.4.2 Medición de distancias:

El proceso de medición de distancias viene implementado por la librería **menu_medicion.jal** y sus librerías asociadas **configuracion_emision_ ultrasonidos** y **configuracion_recepcion_ultrasonidos.** Al referirnos al proceso de medición de distancia estamos considerando tanto el proceso de emisión del tren de pulsos, su recepción, así como el conjunto de funciones que permiten obtener la distancia, valiéndonos de la información de los sensores así como de los errores de deriva de los mismos.

4.4.2.A. Emisión de ultrasonidos

En lo que al proceso de emisión respecta, el tren de ondas ultrasónicas es generado por medio una interrupción asociada a un temporizador, lo que nos permite generar un tren de pulsos con una frecuencia exacta, sin correr el riesgo de distorsiones de la frecuencia debido a procesos bloqueantes.

Para obtener un tren de ondas con una longitud exacta, nos valemos así mismo de la interrupción asociada a dicho contador, la cual activa una subrutina que cuenta el número de veces que se envía un pulso ultrasónico y finalizando la emisión cuando se llega al número de pulsos deseados.

Los principales parámetros que definen el proceso de emisión vienen reflejadas en la siguiente tabla:

Figura 4.4.3: principales registros, funciones y parámetros que determinan el proceso de emisión

Registro	Alias	Valor	Descripción
TIMER 0	int_PWM	-	Temporizador de 8-16 bits con preescaler configurable y con interrupción por desbordamiento asociado.
CONSTANTE	frec_pwm_real	63	Permite generar el tren de pulsos a una frecuencia de 39.825KHz, el valor que mejor se ajusta a las características de los transductores y del filtro.
t0con	registro_timer_PWM	0B0_1001_000	Registro de configuración del t***imer 0***; se usará en modo de 8 bits sin preescaler y asociado al reloj interno.
intcon_tmr0if	flag_int_PWM	bit	Flag asociado a la interrupción por desbordamiento del ***timer0***.
intcon_tmr0ie	habilitar_int_PWM	bit	Máscara de la interrupción t***imer0***.
tmr0	valor_timer_PWM	word	Valor del temporizador ***timer 0***. Para poder obtener el tren de pulsos a la frecuencia justa, tras cada desbordamiento debe cambiarse su valor a ***frec_pwm_real***.
Variable	numero_pulsos	Byte. Configurable por el usuario	Determina el número de cambios de flancos de subida-bajada que deberán ser enviados. El número de pulsos realmente se corresponde con la mitad de este valor
Variable	contadorPWM	Byte	Contador que almacena el número de cambios de flanco generados. Cuando esta variable alcance el valor ***numero_pulsos***, se finalizará el envío
Subrutina – interrupción.	interrupcion_emision()	-	Función asociada a la interrupción ***int_PWM***, que permite actualizar ***contadorPWM*** y actualizar ***valor_timer_PWM*** a la frecuencia determinada por ***frec_pwm_real***
pin_d0	pinPWM	-	Pin asociado a la emisión del tren de ondas ultrasónicas; deberá se configurado por tanto como pin digital de salida.

4.4.2.B. Recepción de ultrasonidos

El proceso de recepción de ultrasonidos precisa de una gran precisión y un retardo mínimo y de valor fijo, ya que de su precisión y resolución dependerá la exactitud de la medición del tiempo de vuelo. Es por ello por lo que la recepción de ultrasonidos está asociada a una interrupción externa asociada al pin de entrada del receptor ultrasónico, garantizando así un retardo mínimo y de valor constante, ya que se evita la distorsión en la medición del tiempo de vuelo debido a cualquier rutina bloqueante.

Para calcular el tiempo de vuelo deberá así mismo intervenir otro temporizador, que sea activado cuando se comience a emitir el tren de ondas ultrasónicas y que se pare en el momento en el que se recibe un eco de retorno.

El proceso de recepción por tanto se encarga de activar la interrupción asociada al receptor de ultrasonidos y de parar el temporizador asociado a la medición del tiempo de vuelo. Así mismo, para evitar entrar en posibles bucles sin fin, se debe añadir una condición de salida en caso de que el tiempo de espera del eco sea demasiado elevado y poder dejar paso a otras sub-rutinas. Se decidió establecer un tiempo de espera máximo de 35.29 *ms*, que equivale a una distancia máxima de unos 6 metros entre el medidor y un objeto.

A continuación mostramos una tabla donde aparecen con detalle las principales interrupciones, variables y registros asociados al proceso de recepción:

Figura 4.4.4: principales registros, funciones y parámetros que determinan el proceso de recepción

Registro	Alias	Valor	Descripción
TIMER 1	int_lapso_maximo	-	Temporizador de 16 bits con preescaler configurable y con interrupción por desbordamiento asociado.
CONSTANTE	numero_ciclos_espera	12595	Establece el tiempo máximo de espera; equivale a una espera de 35.9 ms o una distancia máxima de de 6 metros
t1con	registro_timer_PWM	0b00_11_000_0	Registro de configuración del t***imer 1***; se usará como reloj con preescaler de 1:8, asociado al reloj interno.

PIR1_TMR1IF	flag_int_lapso_maximo	bit	Flag asociado a la interrupción por desbordamiento del ***timer 1***.
PIE1_TMR1IE	habilitar_int_lapso_maximo	bit	Máscara de la interrupción t***imer 1***.
tmr1	valor_timer_lapso_maximo	word	Valor del temporizador ***timer 1***. Para poder establecer el tiempo de espera deseado, tras cada desbordamiento debe cambiarse su valor a ***numero_ciclos_espera***.
Variable	lapso_transcurrido	Word	Almacena el lapso de tiempo transcurrido, medido en ciclos del ***timer 1***
Variable	medicionpwm	byte	Indica el fin del proceso de recepción, sea por recepción del eco o por exceder el tiempo máximo admisible.
pin_b0	Pinreceptor	-	Pin de entrada de la señal del módulo de recepción de ultrasonidos. Deberá ser configurado como entrada digital, y llevará asociado una interrupción externa por flanco de subida
intcon_int0	int_receptor	-	Interrupción asociada a pinreceptor por flanco de subida.
intcon_int0if	flag_int_receptor	bit	Flag asociado al cambio de flanco de subida de pinreceptor
intcon_int0ie	habilitar_int_receptor	bit	Máscara de la interrupción ***int_receptor***.
Subrutina - interrupción	interrupcion_recepcion()	-	Función asociada a la interrupción ***int_receptor***, que calcula el tiempo de vuelo medido en ciclos de reloj timer_PWM. Así mismo actualiza la variable ***medicionpwm*** indicando el fin de la recepción
Subrutina - interrupción	interrupcion_lapso _maximo()	-	Función asociada a la interrupción ***int_lapso_maximo***, permite salir del proceso de espera de eco, indicando el fin de la recepción mediante la actualización de la variable ***medicionpwm***

4.4.2.C. Medición de distancias

El proceso de cálculo de distancia involucra la emisión de ultrasonidos, la recepción del eco de la señal ultrasónica y el cálculo del tiempo de vuelo (TOF) entre ambos procesos. Gracias a demás a la información termodinámica del medio, se podrá calcular finalmente la distancia entre el medidor y el objeto. El proceso a seguir consiste en emitir el tren de pulsos, esperar a la recepción del mismo, calcular el tiempo de vuelo y a partir de ahí y con la información que se dispone sobre las características del medio y de los errores de deriva de los sensores, obtener la distancia al objeto.

Hay que señalar que entre la emisión y la recepción hay que esperar un cierto lapso de tiempo, denominado en el programa ***lapso_minimo***, para permitir que los efectos de la resonancia del emisor no afecten al receptor de ultrasonidos y no resulte en una medición falsa de pocos centímetros. Este parámetro puede ser ajustado desde el menú de configuración mediante el interfaz RS232 o USB.

Se han implementado diferentes estrategias de medición, que incluyen por un lado el realizar una única medición del tiempo de vuelo y obtener la distancia a partir de ese único dato así como la realización de un lote de mediciones por tiempo de vuelo. En este último caso existen también las opciones de enviar el lote completo de mediciones de distancia asociados o bien seleccionar de entre todos los tiempos de vuelo aquel que corresponda con la mediana de todo el conjunto de valores, a fin de garantizar una mayor robustez a la hora de obtener la distancia. Hay que recordar que en el modo manual, la única de las opciones de medición de distancia responden a este último tipo. Al igual que en los procesos anteriores, incluiremos en detalle los registros, funciones, variables y constantes más importantes asociados al proceso de medición:

Figura 4.4.5: variables y constantes principales que determinan el proceso de medición

Registro	Alias	Valor	Descripción
VARIABLE	lapso_minimo	Byte, configurable por el usuario	Lapso mínimo de tiempo, medido en decenas de µs, que hay que esperar entre la emisión y la el comienzo del proceso de recepción .
VARIABLE	vel_sonido	Word	Registra la velocidad del sonido en cm/s.
VARIABLE	retardo	Word, configurable por el usuario	Establece el error de deriva en milímetros asociado a la distancia obtenida.
VARIABLE	temperatura	Word	Registra la temperatura en centésimas de ºC
VARIABLE	deriva_temperatura	Sword, configurable por el usuario	Establece el error de deriva asociado al sensor de temperatura en centésimas de ºC
VARIABLE	Humedad relativa	Byte, configurable por el usuario	Establece el valor de humedad relativa en una escala de 0(0%) a 255 (100%)
CONSTANTE	ciclos_decima_segundo	300000	Constante que permite pasar de unidades de ciclos de timer_lapso_maximo a décimas de segundo.
VARIABLE (externa)	numero_medidas	Byte, configurable por el usuario	Establece el número de mediciones que se realizarán de la distancia al objeto
VARIABLE (externa)	set_medida	Array Word[64]	Array donde se almacenan las mediciones de distancia al objeto calculadas.

A continuación se mostrarán las funciones asociadas al proceso de medición, detallando su funcionamiento:

Figura 4.4.6: funciones y procedimientos que implementan el proceso de medición

Nombre	Acción
calcular_TOF	Realiza el proceso de cálculo de tiempo de vuelo, realizando la secuencia emisión de pulsos – espera – recepción de eco. El tiempo de vuelo viene definido en ciclos de timer_lapso_maximo.
obtencion_distancia	Realiza el cálculo de la distancia (mm) del objeto al medidor a partir del tiempo de vuelo medido en ciclos de timer_lapso_maximo, a partir de la ecuación $d = v_s * TOF / 2$.
menu_medicion_simple	Realiza el proceso completo de medición ; cálculo de tiempo de vuelo (***calcular_TOF***), refresco de la velocidad del sonido (***obtener_vel_sonido_cent***) y obtención de la distancia en mm (***obtencion_distancia***). Solo realiza un muestreo.
menu_medicion_multiple_batch	Realiza el proceso completo de medición para un conjunto de mediciones determinadas por el parámetro ***numero_medidas***. El refresco de la velocidad del sonido se realiza una única vez para ahorrar tiempo, antes de que se realice el bloque de mediciones. Los datos de las mediciones se almacenan en la matriz ***set_medida***.
menu_medicion_multiple_media	Al igual que la función anterior, realiza un conjunto de mediciones, seleccionando únicamente el valor de mediana del conjunto de medidas así como el número de medidas nulas (sin eco). Será el único modo de medición disponible en el modo de funcionamiento manual, al ser el más robusto.
ordena_burbuja	Función encargada de ordenar de menor a mayor un array de tamaño variable de tipo word. Se usa para realizar el proceso de cálculo de mediana del conjunto de medidas realizadas. Usa para ello el algoritmo de ordenación conocido como “burbuja”

4.4.3 Comunicación con el usuario mediante USB y RS-232

Esta parte del programa agrupa las librerías, procedimientos, variables, constantes y funciones encargados del proceso de comunicación con el usuario a través de los periféricos USB o USART de que dispone la unidad de control, encontrándose dentro de la librería ***interfaz_comunicación.jal*** y las librerías asociadas. Antes de entrar en detalle, conviene explicar la problemática asociada al uso de la comunicación mediante USB en el dispositivo medidor de distancia.

4.4.3.A Problemática asociada al protocolo USB-CDC: uso de la interrupción *int_USB*

A diferencia de la comunicación por puerto serie, la comunicación mediante USB involucra un proceso de *negociación* entre nuestro medidor (host) y el dispositivo maestro, sea un PC o cualquier sistema microcontrolado que soporte el estándar USB on-the-go, de manera que se precisa establecer una comunicación constante entre el maestro y el servidor para mantener viva la comunicación. Hay que recordar que el estándar USB actúa de manera trasparente para el usuario, permitiendo la conexión o desconexión en caliente de los dispositivos, a costa obviamente de un mayor esfuerzo de programación y diseño.

Más concretamente el interfaz de comunicación empleado, basado en el protocolo CDC, nos exigía entre otras cosas refrescar la comunicación cada pocos milisegundos, lo que entraba en conflicto con uno de los objetivos de nuestro medidor de distancia como es la precisión de las medidas; recordemos que tanto la emisión como la recepción de ultrasonidos deben realizarse minimizando las rutinas bloqueantes presentes en el programa, a fin de no distorsionar la forma de onda del tren emisor de pulsos ni añadir retardos al tiempo de vuelo de la señal ultrasónica. De esta manera, se optó por incluir el refresco de la comunicación USB dentro de una interrupción asociada a un temporizador del microcontrolador, con el doble objetivo de disponer de una frecuencia de refresco de la señal estable así como de minimizar el retardo asociado a la llamada a la subrutina de refresco de señal.

Comprobamos así mismo que podíamos suspender el refresco de la señal de refresco durante unas pocas decenas de milisegundos, de manera que era posible realizar el proceso completo de medición sin necesidad de ejecutar el refresco de la comunicación USB.

Comprobamos además que podíamos mantener sin refrescar la comunicación servidor-maestro durante intervalos superiores incluso a los 36 milisegundos, superando así el límite máximo de tiempo de espera de eco que impone el proceso de medición, y garantizando por tanto que incluso en condiciones extremas de pérdida de eco, donde el lapso de tiempo entre el refresco de comunicación es máximo, la comunicación no se pierde, cumpliendo por tanto el principal objetivo que pretendíamos.

El lapso de tiempo elegido finalmente entre refrescos de señales USB se corresponde con 0.213 milisegundos, más que suficiente para garantizar la estabilidad de la comunicación USB.

Así pues, finalmente decidimos implementar el proceso de medición enmascarando al inicio la interrupción asociada al refresco USB-CDC, para inmediatamente después de concluir el cálculo de tiempo de vuelo, desenmascarar y posteriormente forzar la interrupción por software, consiguiendo así nuestro objetivo de tener una medición sin distorsiones y sin que se pierda la comunicación por USB. A continuación se incluye una tabla donde se detallan las características del refresco de la comunicación por USB:

Figura 4.4.7: principales registros, funciones y variables que determinan el refresco de la señal USB

Registro	Alias	Valor	Descripción
TIMER2	timer_USB	-	Temporizador de 8 bits con preescaler configurable y con interrupción por coincidencia de tmr2 con pr2
t2con	registro_timer_USB	0B0_1111_0_11	Registro de configuración del t***imer_USB***; se usará en modo de 8 bits sin preescaler y asociado al reloj interno.
PIR1_TMR2IF	flag_int_USB	bit	Flag asociado a la interrupción por desbordamiento del ***timer_USB***.
PIE1_TMR2IE	habilitar_int_USB	bit	Máscara de la interrupción t***imer_USB***.
pr2	valor_timer_USB	10	Variable donde se guarda el numero terminal de ciclos para el temporizador ***timer_USB***. Cada vez que tmr2 coincide con pr2 se activa la interrupcion ***int_USB***. Con el valor por defecto se consigue un refresco de señal cada 0.213 ms.
Subrutina – interrupción.	interrupcion_USB()	-	Función asociada a la interrupción ***int_USB,*** que permite el refresco de la señal USB. Como se verá más adelante, también se encarga

4.4.3.B Implementación conjunta de los interfaces USB y RS-232

El conjunto de librerías estándar que implementan el interfaz USB-CDC, incluidos en la librería principal ***usb_serial.jal*** permiten utilizar el periférico USB de manera similar a como se maneja el periférico USART mediante el estándar RS-232, incluido por su parte bajo la librería ***serial_hardware.jal*****.** Las principales funciones que implementan la configuración de ambos protocolos así como su escritura y lectura vienen incluidas en la siguiente tabla;

Figura 4.4.8: funciones, constantes y variables usadas para implementar los interfaces USB-CDC y RS-232

Nombre	Acción
usb_serial_init()	Procedimiento. Inicializa el periférico USB para funcionar bajo CDC.
serial_hw_init()	Procedimiento. Inicializa el periférico USART para funcionar bajo RS-232.
usb_serial_data	Pseudo-variable. Especifica a nivel funcional un flujo bidireccional de datos hacia/desde el periférico USB. Por ejemplo, "usb_serial_data=var" enviaría al valor de la variable "var" hacia el periférico USB, mientras que "var=usb_serial_data" almacenaría en la variable var el valor actual recibido por el periférico.
serial_hw_data	Pseudo-variable. Se comporta exactamente igual que la pseudovariable usb_serial_data, con la diferencia que el periférico de destino/origen será el módulo USART.
usb_serial_flush()	Procedimiento. Encargada del refresco de la comunicación USB-CDC, deberá ejecutarse cada pocos milisegundos. En nuestro programa estará asociada a un temporizador, activándose por interrupción asociada a su desbordamiento. No existe equivalente para el caso del protocolo RS-232.
serial_hw_read()/ usb_serial_read()	Funciones. Encargadas de la lectura de un byte desde el puerto serie o USB, respectivamente.
serial_hw_write(var byte)/ usb_serial_read(var byte)	Procedimientos. Encargadas de la escritura de un byte hacia el puerto serie o USB, respectivamente.
usb_cdc_tx_buffer_size/ usb_cdc_rx_buffer_size/	Constantes. Tamaño del buffer de transmisión y recepción del módulo USB. Por defecto, se reservan 64 bytes para la transmisión y 32 bytes para la recepción. No existe correspondencia para el protocolo RS-232.

A la hora de diseñar el interfaz de comunicación para ambos protocolos tuvimos en cuenta en primer lugar el flexibilizar en lo posible el uso del medidor de distancia para permitir así el cambio entre uno u otro interfaz de comunicación en caliente y éste fuera común para ambos protocolos. Aprovechando esta similitud en el uso y manejo de ambos protocolos, decidimos pues encapsular las funciones de escritura y lectura por ambos protocolos en una función genérica de escritura y otra de lectura, que explicaremos a continuación:

- Encapsulamiento de las funciones de lectura: lectura_serial_USB

La función ***lectura_SERIAL_USB*** encapsula las funciones de lectura para el puerto serie y USB en una única función, cuya estructura de entrada y salida es idéntica al de las funciones usb_serial_read y serial_hw_read; recibe como entrada un byte por referencia y devuelve como salida un bit que indica si efectivamente ha leído o no un byte del puerto serie/USB. A nivel interno, y dado que existen dos protocolos de comunicación accesibles en modo lectura, se elige qué periférico será objeto del trabajo de lectura en función de las variables de tipo bit ***selector_interfaz_serie*** y ***selector_interfaz_usb***, que explicaremos en el apartado siguiente. Así mismo, tras la lectura se habilita siempre la interrupción asociada al refresco USB, a fin de permitir el refresco de la comunicación USB.

- Encapsulamiento de las funciones de escritura: escritura_serial_USB

Al encapsular las funciones de escritura se tuvo en cuenta que los datos que devuelve el medidor de distancia responden a órdenes de medición, lectura o configuración de parámetros internos enviados desde el exterior; en definitiva, los datos que se enviarán por los periféricos USART o USB responden a una serie limitada de comandos y tendrán un formato preestablecido. Por ello, y para hacer más rápido el envío de datos por estos periféricos, se decidió implementar una función de escritura que no solamente encapsulase las funciones de escritura estándar ligadas a los protocolos mencionados, sino que fuera capaz de enviar grandes paquetes de datos con un formato preestablecido.

Para ello se decidió en primer lugar reservar una zona de memoria donde los datos que fueran a ser enviados fueran almacenados, a modo de buffer de memoria. Este buffer, denominado “buffer intermedio”, está formado por un conjunto de variables globales que permiten almacenar temporalmente los datos e identificadores de datos.

Figura 4.4.9: variables que conforman el buffer intermedio de salida

Nombre	Tamaño	Valor que almacena
id1	Byte	Carácter identificador 1
id2	Byte	Carácter identificador 2
dato1	Byte	genérico
dato2	Word	genérico
dato3	Array: Word[numero_medidas_maximo]	Array con el resultado de un set de mediciones

También se hacía necesario especificar para cada petición de envío de datos el formato de los datos de salida; puede ser que una petición concreta como por ejemplo el resultado de una medición simple solamente requiera el envío de un carácter identificador y un dato de tipo word, o bien que involucre el uso de dos caracteres identificadores y el string de datos, etc... Para implementar las diversas opciones de formato se añadió la variable ***formato_buffer_intermedio***, de 8 bits de longitud y donde los diferentes bits que la conforman permiten especificar el número de datos a enviar y el formato de los mismo, de acuerdo con la siguiente tabla:

Figura 4.4.10: estructura de la variable formato_buffer_intermedio

Bit	Alias	Acción de formato
Bit 0	mostrar_id1	Si es 1 envía el byte id1 como carácter identificador. Si es 0 no lo envía
Bit 1	mostrar_id2	Si es 1 envía el byte id1 como carácter identificador. Si es 0 no lo envía
Bit 2	formato_dato	Especifica el formato de los datos de salida; si es 1 las variables dato1, dato2 y/o dato3 se enviarán como cadenas de caracteres. De lo contrario se enviarán en formato numérico.
Bit 3	mostrar_dato1	Si es 1 envía dato1 como dato de salida. Si es 0 no lo envía
Bit 4	mostrar_dato2	Si es 1 envía el dato2 como dato de salida. Si es 0 no lo envía
Bit 5	mostrar_dato3	Si es 1 envía el dato3 como array de datos de salida. Si es 0 no lo envía
Bit 6,7	Sin definir	Ninguna

El envío de datos por medio de los periféricos USART-USB se realizará por tanto escribiendo los datos correspondientes en ***buffer_intermedio***, para posteriormente definir el formato de estos datos mediante ***formato_buffer_intermedio***, y por último llamar la función de escritura ***escritura_serial_usb***.

4.4.3.C Uso conjunto de las funciones de escritura-lectura y de refresco de señal

Una vez hemos definido las funciones básicas que implementan la comunicación con los interfaces externos USB y RS-232, falta por explicar cómo se usan estas funciones dentro del programa diseñado, junto con las funciones encargadas del refresco de la señal.

Recordemos en primer lugar que, como hemos explicado antes, el refresco de la comunicación USB se realiza mediante una interrupción asociada al temporizador ***timer_USB.*** Implementarlo de esta manera, si bien nos garantiza una frecuencia de refresco estable y minimiza el impacto del refresco de la señal sobre el proceso de medición, nos lleva así mismo a otro problema y es que ahora los procesos de escritura/lectura de los periféricos deben ir sincronizados con los procesos de refresco de señal.

Dicho de otro modo, si durante la ejecución de las subrutinas de escritura-lectura se activa la interrupción int_timer_USB, tanto el proceso de lectura-escritura como el de refresco de señal quedan corrompidos, trayendo como resultado la desconexión del protocolo USB y obligándonos a resetear el medidor para restaurar la comunicación.

Por ello, optamos por la solución más simple que consiste en usar las funciones de escritura y lectura junto a la función de refresco dentro de la subrutina de interrupción ligada a timer_USB; así la función ***interrupcion_USB*** no solamente refresca la comunicación USB, sino que además envía escribe y lee datos hacia/desde los periféricos USB y RS232. A continuación detallamos las variables y funciones que permiten implementar este uso conjunto de las funciones de escritura/lectura y de refresco de comunicación USB.

Figura 4.4.11: variables y funciones ligadas al uso conjunto de escritura, lectura y refresco de la señal USB

nombre	Acción
escritura_serial_usb	Procedimiento. Encapsula los procedimientos de escritura de puerto serie y USB-CDC
lectura_serial_usb	Función. Encapsula las funciones de lectura de puerto serie y USB-CDC
flag_int_USB	Variable tipo bit. Flag asociado al desbordamiento de ***timer_USB***. Su puesta a 1 fuerza la activación de la interrupción de nombre interrupcion_USB, permitiendo por tanto tanto el refresco de la señal como la ejecución de las funciones de escritura/lectura, siempre que el flag ***permiso_RX_TX*** esté en alto.
permiso_RX_TX	Variable tipo Bit. Actúa como flag de permiso escritura_lectura para los interfaces externos de comunicación; si esta variable está a 0, no se llama a la función ***usb_serial*** y por tanto no se ejecutan funciones de lectura-escritura. Si está a 1 sí se permite llamar a la función ***serial_usb***
serial_usb	Procedimiento. Esta función se encarga de llamar a las subrutinas ***escritura_serial_usb*** y ***lectura_serial_usb***. Así mismo se encarga de incluir un mensaje de bienvenida en caso de que sea la primera vez que la función ha sido llamada.
interrupcion_USB	Procedimiento. Asociado al la interrupción por desbordamiento de ***timer_USB***. Se encarga de refrescar la comunicación USB mediante la llamada a ***usb_serial_flush*** y del proceso de lectura/escritura mediante llamada a ***serial_USB***. La variable de tipo bit ***permiso_RX_TX*** actuará aquí, como flag de permiso de lectura-escritura.

Mostramos para mayor claridad el procedimiento interrupcion_USB, para poder ver así cómo se efectúan el refresco y la lectura-escritura;

```
procedure interrupcion_USB() is                    --
   pragma inline                                   --directiva de compilador: inline.
 valor_timer_usb=10                                --asignamos pr2 el valor 10, para obtener la
                                                   --frecuencia de refresco deseada.
   flag_int_USB=off                                --Dentro de la interrupción bajamos el flag.
usb_serial_flush()                                 --Refresco de la comunicación USB.
IF (permiso_RX_TX) then                            --Actuación del flag permiso_RX_TX para permitir
          serial_usb(ESTADO)                       --o no la llamada a la función serial_usb, que
   end if                                          --incluye las funciones de lectura y escritura.
        RETURN                                     --Los procedimientos no devuelven ningún valor.
end procedure                                      --Fin del procedimiento.
```

Estas funciones y variables implementan por tanto todo lo relativo a la comunicación con otros dispositivos. Por ejemplo podemos ver dentro del fichero main.jal un ejemplo del uso de la función de escritura, en concreto cuando se pide desde el exterior una medición simple. En el ejemplo se muestra cómo se efectúa el proceso de escritura de datos por los protocolos USB, RS-232, siguiendo los pasos indicados anteriormente; escritura de datos en el buffer intermedio, edición del formato de los datos del buffer y por último forzar la llamada a la interrupción asociada a timer_USB. A continuación se muestra detalladamente este ejemplo;

```
IF (estado==medicion_simple) THEN                --
    BLOCK                                        --
    menu_medicion_simple(numero_pulsos,distancia) -- Llamada a la función que realiza una medición única
    id1="M"                                      -- Escritura de los resultados en el buffer de datos
    dato2=distancia                              -- (identificador y dato de distancia)
    formato_buffer_intermedio=0b00_010_1_01      -- Asignación de formato a los datos de salida
                                                 -- Fuerza la llamada de la interrupción
    flag_int_USB=on                              -- interrupcion_USB, para garantizar el envío
                                                 -- inmediato de los datos almacenados en el buffer
                                                 -- intermedio.
    END BLOCK                                    --
    (...resto de sentencias)                     --
END IF                                           --
```

Como podemos ver, forzamos la activación de la interrupción para así garantizar que los datos efectivamente se envíen y no resulten borrados en el hipotético caso de que las peticiones de escritura en el buffer se produjeran a frecuencias más elevadas que la frecuencia de activación de nuestra interrupción. En lo que se refiere a la lectura de los datos de entrada, ésta, por ser algo más compleja que el procedimiento de escritura, se tratará con mayor detalle en el apartado siguiente.

4.4.3.D Gestión de los mensajes de entrada desde los periféricos USB y USART

La gestión de los datos que se reciben desde los periféricos USART y USB responden a una secuencia de comandos prefijada, de longitud variable entre 1 y 4 bytes. El primero de estos bytes se denomina identificador primario, y permite conocer qué tipo de petición se está realizando, llevando al sistema a diferentes estados, pertenecientes a 3 grupos principales: escritura de parámetros internos, lectura de parámetros (internos o ambientales) y por último peticiones de medición.

Será este primer byte el que nos permitirá también conocer la longitud de la cadena de entrada, de manera que una vez recibido un carácter que pertenezca al grupo de identificadores primarios, será posible saber cuántos bytes adicionales deben leerse inmediatamente después de este, y disponer así de la secuencia completa. La lista de los diferentes identificadores primarios y sus estados asociados se encuentra detallada en la tabla siguiente.

Figura 4.4.12: lista de identificadores primarios y estados asociados

Identificador primario	Nombre del estado	Acción
M	medicion_simple	Medición simple. Salida formato string
m	medicion_simple_word	Medición simple. Salida formato numérico
B	medicion_multiple_batch	Conjunto de mediciones. Salida string
b	medicion_multiple_batch_WORD	Conjunto de mediciones. Salida numérica
N	medicion_multiple_media	Conjunto de mediciones y obtención de mediana. Salida string.
n	medicion_multiple_media_WORD	Conjunto de mediciones y obtención de mediana. Salida numérica
S	configuracion	Acción de escritura de parámetros internos
R	lectura	Acción de lectura de parámetros (ambientales o internos)
Otros	Estados ligados a la interfaz de uso manual; se verán en el apartado siguiente.	

La implementación de la lectura de las consignas de configuración, medición o lectura se dividirá por tanto en 3 partes; lectura del identificador primario, reconocimiento del estado asociado y lectura de la cadena de bytes adicional asociada.

Lectura del identificador primario

La lectura del identificador primario se realiza mediante la función ***usb_serial***, que como hemos visto en el sub-apartado anterior 4.3.3.C es llamada con una frecuencia regular cada 0.213 ms, e incluye las funciones de lectura y escritura de los periféricos USB y USART. Para ello utiliza la variable global ESTADO, que será actualizada con cada ciclo de lectura de la función ***usb_serial*.** Para mayor claridad volvemos a incluir las sub-rutinas y variables implicadas en la lectura de este byte.

Figura 4.4.13: variables y funciones ligadas a la lectura del byte identificador primario

nombre	Acción
lectura_serial_usb	Función. Encapsula las funciones de lectura de puerto serie y USB-CDC
permiso_RX_TX	Variable tipo Bit. Actúa como flag de permiso escritura_lectura para los interfaces externos de comunicación; si esta variable está a 0, no se llama a la función ***usb_serial*** y por tanto no se ejecutan funciones de lectura-escritura. Si está a 1 sí se permite llamar a la función ***serial_usb***
serial_usb	Procedimiento. Esta función se encarga de llamar a las subrutinas ***escritura_serial_usb*** y ***lectura_serial_usb***. Así mismo se encarga de incluir un mensaje de bienvenida en caso de que sea la primera vez que la función ha sido llamada.
interrupcion_USB	Procedimiento. Asociado al la interrupción por desbordamiento de ***timer_USB***. Se encarga de refrescar la comunicación USB mediante la llamada a ***usb_serial_flush*** y del proceso de lectura/escritura mediante llamada a ***serial_USB***. La variable de tipo bit ***permiso_RX_TX*** actuará aquí, como flag de permiso de lectura-escritura.
ESTADO	Variable global. Esta variable almacena el byte leído desde los periféricos USB/USART, cuando se llama a la sub-rutina ***usb_serial.***

Reconocimiento del identificador primario

Como hemos visto la variable global ***ESTADO*** almacena el posible byte identificador primario que nos permitirá saber el estado asociado. Para ello este byte debe compararse con la lista de identificadores primarios válidos indicada anteriormente, comprobando dentro del bucle principal del programa esta lista la variable con la lista. En caso de que efectivamente este byte se

corresponda con un identificador válido, se procederá a la llamada de las sub-rutinas encargadas de la lectura del resto de bytes de la secuencia de entrada y de realizar las acciones correspondientes. Mostramos a modo de ejemplo un trozo de código perteneciente al fichero ***main.jal***.

```
IF (estado==medicion_simple) THEN          --comprobación de la variable estado con el
   BLOCK                                   -- identificador primario "medicion_simple"
   menu_medicion_simple(numero_pulsos,distancia)   -- Llamada a la función que realiza una medición única
   id1="M"                                 -- Escritura de los resultados en el buffer de datos
   dato2=distancia                         -- (identificador y dato de distancia)
   formato_buffer_intermedio=0b00_010_1_01 -- Asignación de formato a los datos de salida
   flag_int_USB=on                         -- Fuerza la llamada de la interrupción
                                           --interrupcion_USB, para garantizar el envío inmediato --de los datos almacenados en el buffer intermedio.
ELSIF (estado==lectura) THEN               --comprobación de la variable estado con el
   BLOCK                                   --identificador primario "lectura"
   formato_buffer_intermedio=0             --Borrado del formato del buffer intermedio
   submenu_lectura()                       --Accede al submenú de lectura
   flag_int_USB=on                         --Fuerza la llamada de interrupcion_USB
ELSIF (estado==configuracion) THEN         --comprobación de la variable estado con el
   BLOCK                                   --identificador primario "configuracion"
   formato_buffer_intermedio=0             --Borrado del formato del buffer intermedio
   submenu_configuracion()                 --Accede al submenú de configuracion
   flag_int_USB=on                         --Fuerza la llamada de interrupcion_USB

   END BLOCK                               --
   (...resto de sentencias)                --
END IF                                     --
```

En este ejemplo podemos ver cómo se gestiona el reconocimiento de los identificadores primarios mediante sentencias de tipo if-else, y cómo se accede a las diferentes subrutinas asociadas a cada estado, dentro del bucle principal asociado a ***main.jal***.

Aunque se verá con mayor profundidad en el apartado relativo a la interfaz de usuario, podemos adelantar que de entre todos los estados señalados arriba los únicos estados que tienen involucrados la lectura de bytes adicionales además del byte primario son los estados "configuracion" y "lectura". Será en el apartado siguiente cuando entremos en detalle sobre cómo se leen estos bytes adicionales en ambos casos.

Lectura de los bytes adicionales: reconocimiento del identificador secundario

La lectura de los bytes adicionales se produce, como acabamos de decir, cuando el identificador primario corresponde a los estados lectura o configuración; el uso del llamado identificador secundario se hace imprescindible para poder saber qué parámetro leer o modificar. Ello hace que una vez que se ha reconocido los estados configuracion o lectura se accedan respectivamente a las sub-rutinas submenu_configuracion() y submenu_lectura(), encargadas del proceso de lectura del identificador secundario y del proceso de lectura/escritura correspondiente.

Tenemos pues dos sub-rutinas, ***submenu_ configuracion() y submenu_lectura()*** , que se encargarán respectivamente de gestionar las peticiones de escritura o lectura de los parámetros del dispositivo. Estas rutinas, así como la lista de identificadores secundarios, se encuentran dentro del fichero ***submenu_configuracion_lectura.jal***, en el que están encapsulados estos procesos. La lista de bytes identificadores secundarios la podemos ver a continuación, junto con las variables globales que modifican o leen;

Figura 4.4.14: lista de identificadores secundarios

Identificador secundario	Nombre del estado	Petición	Variable que modifica/lee
L	configuracion_lapso_minimo lectura_lapso_minimo	Configuración/lectura del tiempo máximo de espera (en decenas de microsegundo) entre la emisión y recepción de ultrasonidos.	lapso_minimo
D	configuracion_retardo lectura_retardo	Configuración/lectura del retardo asociado a las mediciones de distancia (en mm).	retardo
N	configuracion_numero_pulsos lectura_numero_pulsos	Configuración/lectura del número de pulsos ultrasónicos enviados por el emisor.	numero_pulsos
Z	configuracion_deriva_temperatura lectura_deriva_temperatura	Configuración/lectura de la deriva asociada a le medición de temperatura (en centésimas de ºC).	deriva_temperatura
H	configuracion_efecto_humedad lectura_efecto_humedad	Configuración/lectura de la humedad relativa (en escala 0-255).	humedad_relativa
K	configuracion_numero_medidas lectura_numero_medidas	Configuración/lectura de del número de medidas a realizar cuando se selecciona un método de medición múltiple.	numero_medidas

T	lectura_temperatura	Lectura del valor actual de temperatura (centésimas de ºC).	temperatura
V	lectura_velocidad_sonido	Lectura del valor actual de la velocidad del sonido (cm/s).	vel_sonido

Las peticiones de lectura involucran únicamente el reconocimiento del byte identificador primario "***R***", asociado a la lectura, más la lectura del byte secundario asociado al dato que se deberá enviar por USB o RS-232. Será dentro de la propia subrutina ***submenu_lectura()*** donde se realice el proceso de reconocimiento del byte secundario, siguiendo una estructura similar a la empleada por el bucle principal.

El proceso de lectura de este byte secundario se realiza ahora usando directamente la función ***lectura_serial_usb*** y deshabilitando el flag de lectura-escritura asociado a la función ***interrupcion_usb*** y de nombre ***flag_rx_tx***; de esta manera hacemos que la lectura de bytes de los periféricos pase a ser hecha en exclusiva por la función ***escritura_usb*** alojada en la subrutina ***submenu_lectura***, sin perder eso sí la capacidad de refresco de señal ligada a la interrupción. Una vez se ha leído el byte identificador secundario, se rehabilita el flag de de lectura-escritura, para que así se puedan leer desde los periféricos posibles nuevas peticiones encabezadas por el byte identificador principal correspondiente.

Por otro lado, debemos tener en cuenta además que es posible que la secuencia de caracteres que recibamos no esté correctamente codificada, bien sea porque se ha asignado un identificador secundario que no pertenece a ninguno de la lista comentada anteriormente, bien porque existan ruidos o retardos que imposibiliten continuar con la subrutina de lectura de parámetros. Para evitar ello, la lectura del byte identificador secundario se repite un número fijo de veces y durante un lapso de tiempo pequeño; más concretamente, decidimos que el número de veces que debía realizarse la medición sería de unas 50 veces, repitiendo la operación cada 100 microsegundos hasta que se leyera por los periféricos algún byte. Posteriormente este byte leído se compararía con los de la lista de identificadores secundarios y en caso afirmativo realizar el proceso de lectura y enviar los datos correspondientes a los periféricos. En caso contrario, se abortaría inmediatamente la subrutina de lectura y la función devolvería un 0, indicando que el proceso de lectura ha fracasado. Esto implica que el lapso máximo de tiempo entre el envío del primer byte identificador primario y el secundario no deberá exceder nunca los 5 milisegundos.

Mostramos a continuación un fragmento de la ***subrutina submenu_lectura*** para aclarar más cómo funciona el proceso de lectura de parámetros:

```
function submenu_lectura ()  return bit is      --la función devolverá 1 si la petición de lectura ha
                                                --sido válida, un 0 si en caso contrario
permiso_RX_TX=off                               --deshabilita temporalmente el flag de permiso de
                                                --escritura/lectura para la interrupción ligada al USB
var byte parametro_lectura = 0                  --bajo esta variable temporal se almacena el
                                                --identificador secundario.
var byte aux1=0                                 --contador de número de mediciones realizadas.
CONST BYTE TMAX=10                              --tiempo a esperar entre medidas (decenas de us).
CONST BYTE NMAX=50                              --número máximo de veces que se repetirá la lectura.

while(!lectura_serial_usb(parametro_lectura)&   --bucle de lectura. Fin de bucle cuando lea o cuando
   aux1<NMAX) loop                              -- finalice el número máximo de lecturas.
   delay_10us(TMAX)                             --retraso entre lecturas.
   aux1=aux1+1                                  --actualización del contador.
end loop                                        --fin de bucle de lectura.
aux1=0                                          --reseteo del contador.
flag_int_USB=on                                 --Fuerza la llamada de ***interrupcion_USB*** para
                                                --refrescar la señal USB
if aux1==NMAX then                              --comprueba si se ha leído un posible id. secundario.
   permiso_RX_TX=on
   return off                                   --salida de subrutina indicando error.
end if
permiso_RX_TX=on                                --rehabilita el flag de permiso de lectura-escritura
                                                --para la interrupción ligada al USB.
id1="R"                                         --escribe el id. primario "R" en el buffer intermedio
if parametro_lectura==lectura_lapso_minimo then --Comprueba si el identificador secundario es
                                                --"lectura de lapso minimo"
id2=lectura_lapso_minimo                        --escritura del id. secundario en el buffer intermedio.
dato1=lapso_minimo                              -- escritura del lapso mínimo en el buffer intermedio
formato_buffer_intermedio=0b00_001_0_11         --Asignación de formato al buffer intermedio
                                                --Fuerza la llamada de ***interrupcion_USB*** para
flag_int_USB=on                                 --escribir por los periféricos los datos del buffer
                                                --intermedio.
(... sentencias ligadas a la escritura por LCD)
return on                                       --salida de la subrutina, indicando éxito.
elsif parametro_lectura==lectura_velocidad_sonido --comprueba si el id. secundario se corresponde con
then                                            --"lectura de la velocidad del sonido"
(...resto de sentencias)
   return off                                   -- salida de la función, indicando que el byte id.
end FUNCTION                                    -- secundario no ha sido reconocido como válido.
```

Podemos observar cómo una vez se ha conseguido leer un nuevo byte por medio de la función lectura_serial_usb se procede a comprobar si este byte se corresponde efectivamente con algún identificador secundario, editando y asignando formato al buffer intermedio y forzando el envío de estos datos a los periféricos USB/USART. En este caso la subrutina ***submenu_lectura*** devuelve un 1 indicando que se ha reconocido satisfactoriamente la petición de lectura y que en consecuencia se han enviado los datos requeridos a los periféricos correspondientes. En caso de que no se lea en el tiempo establecido un byte, o este byte no corresponda con ninguno de los bytes de la lista mencionada arriba, la función devolverá un 0 indicando que la secuencia de comandos de lectura no era correcta.

La subrutina de gestión de peticiones de configuración está implementado de manera idéntica que la de lectura con la salvedad de que además de tener que leer el byte identificador secundario que indique el parámetro a modificar se deberán leer 1 o dos bytes adicionales donde vendrá recogido el valor del parámetro que será modificado. Incluimos un trozo de código de la función ***submenu_configuracion***

```
function submenu_configuracion()  return bit is      --la función devolverá 1 si la petición de lectura ha
                                                     --sido válida, un 0 si en caso contrario

permiso_RX_TX=off                                    --deshabilita temporalmente el flag de permiso de
                                                     --escritura/lectura para la interrupción ligada al USB

var byte parametro_configuracion = 0                 --identificador secundario.
var byte parametro_configuracion2 = 0                --dato a modificar
var byte parametro_configuracion3=0                  --dato a modificar (en caso de que ocupe 2 bytes)
var byte aux1=0                                      --contador de número de mediciones realizadas.
CONST BYTE TMAX=10                                   --tiempo a esperar entre medidas (decenas de us).
CONST BYTE NMAX=50                                   --número máximo de veces que se repetirá la lectura.

while(!lectura_serial_usb(parametro_lectura)&        --bucle de lectura. Fin de bucle cuando lea o cuando
   aux1<NMAX) loop                                   -- finalice el número máximo de lecturas.
   delay_10us(TMAX)                                  --retraso entre lecturas.
   aux1=aux1+1                                       --actualización del contador.
end loop                                             --fin de bucle de lectura.
aux1=0                                               --reseteo del contador.
if aux1==NMAX then                                   --comprueba si se ha leído un posible id. secundario.
   permiso_RX_TX=on                                  --antes de salir rehabilita el flag de permiso E/L
   return off                                        --salida de subrutina indicando error.
end if

If parametro_configuracion==configuracion_lapso_minimo --Comprueba si el identificador secundario es
then                                                 --"configuracion de lapso minimo"
   aux1=0                                            --repetición del bucle de lectura para obtener el valor
   while (!lectura_serial_usb(parametro_configuracion2) del lapso mínimo.
```

```
   &aux1<NMAX) loop
      delay_10us(TMAX)
      aux1=aux1+1
   end loop
   if aux1==NMAX then
      permiso_RX_TX=on                          -- comprobación de obtención correcta del byte
      return off
   end if
   permiso_RX_TX=on                             --rehabilitación del flag de lectura-escritura

                                                --comprueba si el valor obtenido pertenece al rango
  If parametro_configuracion2>=1 then           --de valores admisible.
        lapso_minimo=parametro_configuracion2   --Un valor !=0 actualiza el lapso_minimo.
     else
        lapso_minimo=lapso_minimo_defecto       -- Si es un 0, vuelve a los parámetros de fábrica.
     end if

data_eeprom_write(offset_eeprom_lapso_minimo,lapso_minimo)  -- escritura del nuevo valor en la eeprom
(... sentencias ligadas a la escritura por LCD)
return on                                       --salida de la subrutina, indicando éxito.
elsif parametro_lectura==lectura_velocidad_sonido then  --comprueba si el id. secundario se corresponde con
                                                --"lectura de la velocidad del sonido"
(...resto de sentencias)
   return off                                   -- salida de la función, indicando que el byte id.
end FUNCTION                                    -- secundario no ha sido reconocido como válido.
```

Se puede observar que como ya hemos dicho la diferencia fundamental es que se precisan leer hasta dos bytes adicionales que contienen el valor que deberá ser escrito en la memoria de programa, tanto en la memoria volátil como en la EEPROM. Este valor será antes comprobado para confirmar que está entre el rango de valores válido en cada caso; rango de deriva de temperatura entre ±25.00 ºC número de pulsos mayor que 0, etc... Guardar en la EEPROM el valor modificado tiene la utilidad de conservar el valor de los parámetros de manera permanente aunque se apague el dispositivo.

4.4.4 Modo de funcionamiento manual

En el apartado 3.3.5 explicamos cómo estaba implementada a nivel físico el interfaz de uso manual, indicando los interruptores y botones que realizaban las únicas dos tareas que se podían realizar desde el interfaz manual; lectura de temperatura y medición de distancias. Aquí explicaremos como está implementado la parte del firmware encargado de gestionar estas peticiones de lectura y medición.

Recordemos que la interfaz manual de usuario disponía de dos botones encargados de estas dos acciones. Aquí se detallan las peticiones comentadas así como las variables de estado asociadas:

Figura 4.4.15: Comandos de la interfaz manual y acciones, estados y variables asociadas

Elemento de la interfaz manual	Acción asociada	Variable de estado relacionada (tipo bit)	Estado del sistema asociado
Botón de medición	Mientras está pulsado, el dispositivo realiza continuamente mediciones con filtrado de mediana. Mostrará por el LCD la mediana de los valores muestreados, así como el número de medidas fallidas.	pin_hacer_medicion	medicion_distancia_ modo_manual
Botón de temperatura	Mientras esté pulsado, el dispositivo realiza continuamente mediciones de la temperatura ambiente y las muestra por el LCD	pin_leer_temperatura	medicion_temperatura_ modo_manual
Interruptor de modo manual/ automático	Cambia entre el modo manual y el modo automático.	pin_cambio_modo_manual	

La implementación de la gestión de estas peticiones es muy sencilla, puesto que únicamente requieren del uso de una función que esté continuamente leyendo las variables de estado arriba señaladas, sin que sea tampoco preciso que exista una elevada velocidad de muestreo. Es por ello por lo que la función encargada de la lectura de estas variables de estado y de su gestión subsiguiente se incluya explícitamente dentro del bucle principal del programa.

Esta subrutina, de nombre ***menu_manual,*** está incluida dentro del fichero ***interfaz_comunicacion.jal*** que como ya comentamos agrupa todo lo relacionado con las diferentes interfaces de comunicación incluidas en el medidor. Se encargará de leer las variables de estado ***pin_leer_temperatura*** y ***pin_hacer_medicion***, de modo que si el modo de funcionamiento es el modo manual, es decir, el interruptor de modo manual está activado, modificará la variable global ***ESTADO*** llevando al sistema a los estados respectivos “medicion_distancia_modo_manual” y “leer_temperatura_modo_manual”.

Como pasaba con la comunicación USB, la variable global ***ESTADO*** registra la petición actual de medición o lectura, comprobándose dentro del bucle principal si dicho estado se corresponde con alguno de los estados de la lista. En el caso de que esté activa la interfaz manual, lógicamente, la variable ***ESTADO*** podrá tomar únicamente los dos estados posibles indicados arriba, realizándose las rutinas de medición con filtro de mediana o de lectura de temperatura. Hay que indicar que, como veremos en el apartado siguiente, la activación del interruptor asociado a la interfaz manual desactiva completamente los interfaces de comunicación USB y RS-232, de modo que en estos casos la variable global ***estado*** solo será modificada desde la botonera; se evita así el riego de interferencias entre interfaces.

4.4.5 Configuración de las interfaces de usuario

Se ha señalado en anteriores ocasiones que nuestro medidor de distancias dispone de 3 interfaces de comunicación distintos; por USB, RS-232 y manual, por medio de una botonera. Será tarea del firmware del dispositivo el evitar la interferencias entre las diferentes interfaces de usuario y de gestionar correctamente la comunicación mediante estas interfaces. Esta parte en concreto, como ya indicamos anteriormente, se encuentra dentro del fichero ***interfaz_comunicación.jal***, encargado de implementar los diferentes interfaces y la relación entre ellos.

Como en anteriores ocasiones, optamos por la solución más sencilla para el usuario, consistente en gestionar en la medida de lo posible la activación o desactivación de los interfaces de forma automática. Así mismo, pensamos que lo ideal sería que el cambio entre las diferentes interfaces de usuario se pudiera realizar en caliente (con el dispositivo funcionando) permitiendo por ejemplo que nuestro medidor funcionara al inicio de manera manual y se pudiera conectar sin antes apagarlo al puerto serie o USB para reconfigurarlo, o viceversa

De esta manera, consideramos una serie de condiciones que determinaban el uso de las diferentes interfaces ;

- La interfaz manual, al activarse, anula las interfaces RS-232 y USB. La interfaz manual se activa mediante el interruptor que conmuta entre el modo manual y modo automático. Así, sólo cuando este interruptor esté desactivado se permitirá la comunicación por medio de USB y/o RS-232. Ello permite el uso del medidor de distancia de forma manual, sin interferencias desde el USB o el puerto serie, con la ventaja de no tener que quitar las conexiones y pudiendo en consecuencia alimentarlo desde el cable USB.

- La interfaz USB solo puede ser activada cuando el protocolo CDC esté correctamente configurado. Ello se comprueba llamando a la función de la librería estándar ***usb_is_configured()***, que devuelve un 1 si el proceso de negociación entre el dispositivo y el servidor ha concluido satisfactoriamente y el medidor ha sido reconocido. Evitamos así bloquear el dispositivo, ya que obligamos a la correcta configuración del periférico antes de obligarlo a que lea o escriba.

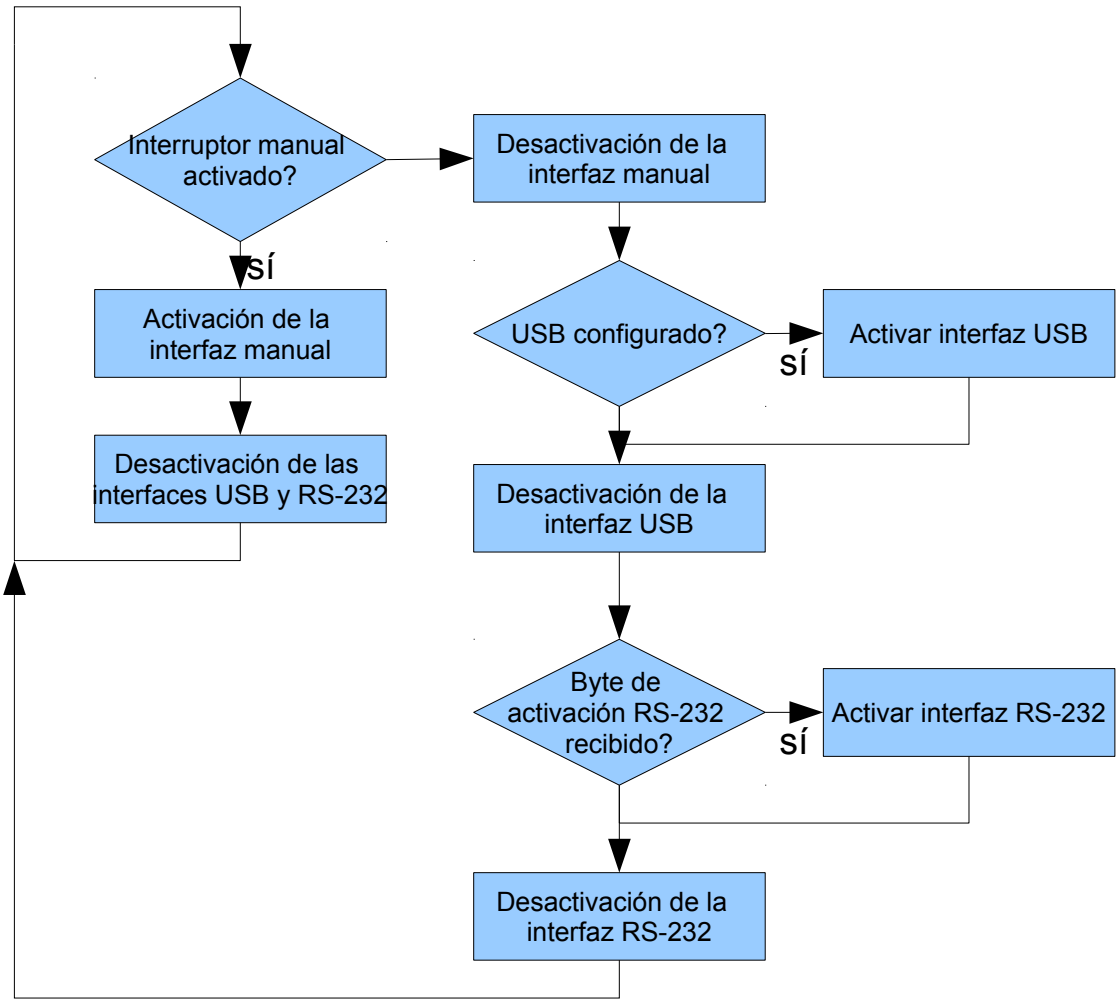
- -La interfaz RS-232 sólo puede ser activada cuando desde el puerto serie se reciba una señal de activación. Esto es debido a que a diferencia del protocolo USB, no existe ninguna confirmación de que al otro lado de la conexión por RS-232 exista un aparato que esté a la escucha. La señal de activación es muy sencilla, consistente en el envío de un byte concreto. A partir de ese momento, se considerará el puerto RS-232 como una fuente válida de recepción y /o envío de datos.

- -Al inicio se debe elegir al menos una interfaz válida de comunicación. En la práctica ello supone el forzar un bucle infinito hasta que el puerto USB esté configurado, se reciba un byte de activación desde el puerto serie o se active el interruptor asociado a la interfaz manual.

- -Se debe permitir el cambio entre las diferentes interfaces existentes.

Con todas estas condiciones desarrollamos dentro del fichero ***interfaz_comunicacion.jal*** las subrutinas ***reconfigurar_conexiones_hardware*** y ***reconfigurar_conexiones_inicio***, que se encargan de implementar la activación y reconexión dinámica de los interfaces, siguiendo el diagrama de flujo siguiente:

Figura 4.24.16: Diagrama de flujo representando la configuración de los interfaces.



Las funciones que implementan el diagrama de flujo mostrado anteriormente son las siguientes;

Figura 4.4.17: Funciones que implementan la selección y reconfiguración de los interfaces.

Nombre	Descripción
reconfigurar_conexiones_inicio	Configura por defecto la conexión usb mientras espera a que se reciba una señal de activación desde el puerto serie, se termine de configurar el USB o el interruptor de la interfaz manual se active. Esta función por tanto provoca un bucle infinito hasta que al menos uno de los interfaces existentes esté en condiciones de funcionar. Se ejecuta una sola vez, al inicio.
reconfigurar_conexiones_hardware	Comprueba el estado de de las interfaces de comunicación, procediendo a modificar la interfaz de comunicación activa las en función de su estado actual y pasado. La interfaz manual tiene preferencia sobre el resto, desactivando las interfaces USB y Rs-232 en cuanto se activa el interruptor asociado a la interfaz manual. Cuando el interruptor asociado a la interfaz manual está desactivado, se encargará de añadir los interfaces RS-232 o USB si anteriormente estaban desactivados. Esta función se ejecuta dentro del bucle principal.

El estado actual de las interfaces estará directamente ligado por su parte a las variables de estado que a continuación mostramos:

Figura 4.4.18: Variables de estado asociadas a las interfaces.

Interfaz asociada	Variable de estado	Descripción
RS-232	selector_interfaz_serie	Variable global tipo bit. Si es 1 indica que la interfaz RS-232 está actualmente activa
USB	selector_interfaz_usb	Variable global tipo bit. Si es 1 indica que la interfaz USB está actualmente activa
Manual	pin_cambio_modo_manual	Alias del pin digital de entrada asociado al interruptor de cambio de modo manual/automático. Si es 1 indica que la interfaz activa actualmente es la interfaz manual.

Dado que la velocidad y la precisión en la frecuencia de conmutación entre las diferentes interfaces no son factores críticos, la función ***reconfigurar_conexiones_hardware*** es ejecutada dentro del bucle principal, comprobándose así cada pocos milisegundos el estado actual de las conexiones y permitiendo un cambio bastante rápido entre los diferentes interfaces.

4.4.6 Uso de las librerías configuracion_LCD.jal y configuracion_EEPROM.jal

Resta por explicar el uso de las librerías encargadas del uso del display de cristal líquido (LCD) y a la librería que permite el registro de las variables en la eeprom del sistema para finalizar completamente con la explicación del firmware desarrollado. La librería encargada de la escritura/lectura de la eeprom tiene por nombre configuracion_eeprom, y no incluye más que la librería estándar ***pic_data_EEPROM*** de la que hemos hablado con anterioridad así como constantes que nos permiten conocer la posición de memoria en la eeprom de cada variable que precise ser incluida en la eeprom. La lista de variables cuyo valor se almacena en la eeprom es la siguiente:

Figura 4.4.19: Lista de variables que se incluyen en la EEPROM.

Posición de memoria	Variable que almacena
offset_eeprom_lapso_minimo	lapso_minimo
offset_eeprom_numero_pulsos	numero_pulsos
offset_eeprom_retardo	retardo
offset_eeprom_vel_sonido	vel_sonido
offset_eeprom_coeficiente_temperatura	coeficiente_temperatura
offset_eeprom_temperatura	temperatura
offset_eeprom_deriva_temperatura	deriva_temperatura
offset_eeprom_humedad_relativa	humedad_relativa
offset_eeprom_numero_medidas	numero_medidas

El uso de la memoria permanente EEPROM nos permite almacenar los parámetros internos del sistema de manera permanente, sin tener que reconfigurarlos cada vez que se apague el medidor de distancias como pasa con la memoria flash de datos del microcontrolador. Ello nos da una mayor versatilidad en su uso y una mejora de la comodidad al emplear el dispositivo medidor.

Por otra parte, la librería encargada del LCD, de nombre ***configuracion_LCD.jal***, incluye

simplemente una lista de constantes de tipo cadena de caracteres que serán las que muestren por pantalla la diferente información sobre el estado del sistema así como los resultados de peticiones de lectura, medición o modificación de parámetros internos. Así, cada acción que se realice sobre el medidor de distancia tendrá como consecuencia la actualización de la información que muestra el display.

Figura 4.4.20: lista de texto del LCD en función del estado del medidor.

Acción del medidor	Pantalla LCD	
Inicio:buscando interfaces válidos	Conectando...	Cuando encuentra una conexión válida, el mensaje pasa a ser: Hecho.
Conexión efectuada con el interfaz USB	Conexion USB	
Conexión efectuada on el interfaz RS-232	Conexion RS-232	
Conexión efectuada con los interfaces USB + RS-232	Conexion USB+232	
Conexión efectuada con el modo manual	Modo manual	
Medicion simple/ medición_simple_word	Medida:=XXXX mm	XXXX=distancia obtenida
Medicion_multiple_batch/ medicion_multiple_batch_word	Midiendo...	Cuando termina el conjunto de mediciones, aparece el mensaje: Hecho.
Medicion_multiple_media/ medicion_multiple_media_word/ medicion_distancia_modo_manual	D(XXX) =YYYYmm	YYYY=valor de mediana del array de distancias. XXX=número de medidas erróneas (valores nulos) obtenidos.
Lectura temperatura/ medicion_temperatura_modo_manual	Temp(oC)=XXX.XX	XXX.XX = temperatura/100
Lectura velocidad del sonido	Vel(m/s)=XXX.XX	XXX.XX=vel_sonido/100
Lectura humedad relativa/ Configuración humedad relativa	H.Rel(%)=XXX.XX	XXX.XX=humedad_relativa*100/255
Lectura error deriva temperatura/ configuracion error deriva temperatura	Error(oC)=XX.XX	XX.XX=error_deriva_temperatura/100
Lectura retardo/ configuración retardo	Retardo=XXXXX mm	XXXXX=retardo.
Lectura_lapso_minimo/ configuracion_lapso_minimo	Lapso_min=XXX0us	XXX=lapso_minimo. Hay que recordar que las unidades nativas de la variable lapso_minimo son decenas de microsegundo, de ahí añadir el "0" al final
Lectura numero de pulsos/ configuracion del número de pulsos	Num_pulsos=XXX	XXX=num_pulsos.
Lectura numero de medidas/ configuracion del número de medidas	Num. medidas=XXX	XXX=numero_medidas.

Además de estas constantes, se incluye una función que permite escribir un array de caracteres de longitud variable para un display específico de 16x1, que es el que usamos en este proyecto. Esta función tiene por nombre ***escribir_lcd***, y su uso permite simplificar un poco los algoritmos de escritura en el LCD.

Con este último sub-aparatado terminamos la explicación del software de nuestro y por tanto del medidor en sí. En el capítulo 5 se muestran resultados de las mediciones, mientras que en los anexos se adjuntará todo el código fuente así como los scripts de matlab y los esquemáticos y PCB´s del circuito electrónico.

5 - RESULTADOS

En este capítulo mostraremos una serie de resultados de nuestro dispositivo medidor de distancias con el fin de comprobar así su fiabilidad. Recordemos la ecuación que relaciona linealmente la distancia a un objeto con el tiempo de vuelo, teniendo en cuenta los factores atmosféricos;

$d_{real}=\frac{v_s*t}{2}=\frac{(331.58+0.62T_{real}+0.015\,HR)*t}{2}+d_{retardo}$, donde d_{real} es la distancia real al objeto, *d* es el valor de distancia teórico (sin corrección de condiciones ambientales), T_{real} es la temperatura en ^{o}C , HR es la humedad relativa en tanto por 100 y $d_{retardo}$ es el retardo asociado al circuito electrónico.

La temperatura se obtiene automáticamente a partir del sensor de temperatura del propio medidor, teniendo de fábrica un error de $\mp$ 0.5 ºC. La humedad relativa, al ser un parámetro para el que no conocíamos el valor exacto, no ha sido tenida en cuenta, siendo esta una pequeña fuente incertidumbre que como vimos en el capítulo 2 afectará como mucho en un 0.45 % suponiendo el caso más extremo de una diferencia de humedad relativa del 100%, o lo que es lo mismo, en 4.5 mm por metro, error que asumiremos.

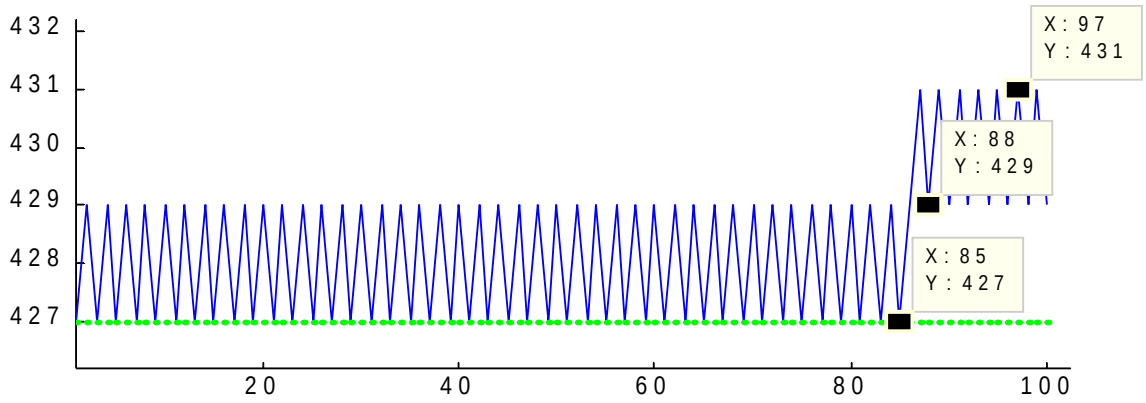
Tenemos entonces que el único parámetro que queda libre en esta ocasión es el retardo asociado al propio circuito electrónico, que en principio deberá ser constante e independiente de la distancia y las condiciones ambientales. Obtendremos empíricamente el retardo usando la ecuación anterior, $d_{retardo}=d_{real}-\frac{(331.58+0.62\,T_{real}+0.015\,HR)*t}{2}$, y este valor nos servirá para conocer la bondad de la estimación de la distancia y por consiguiente del propio medidor. Obviamente, se espera que el retardo tome valores muy pequeños, del orden de 20-30mm (equivalente a menos de 100 us). Mostramos a continuación un conjunto de mediciones a diferentes distancias y bajo diferentes condiciones ambientales, y veremos los errores asociados y su distribución.

5.1 Comportamiento del medidor ante cambios en las condiciones ambientales y estimación de errores.

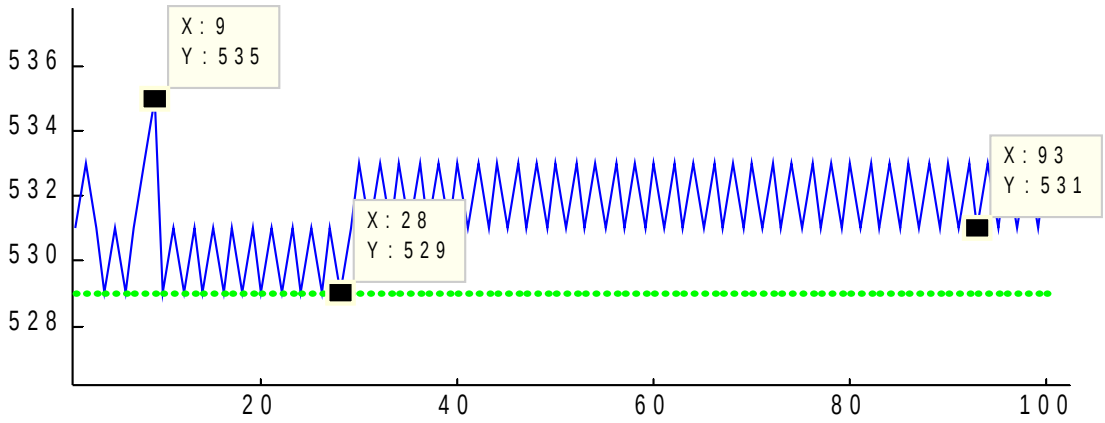
Esta serie completa de mediciones se ha realizado enfrentando nuestro dispositivo a una plancha de metal de 15x 15 cm^2 situado a distancias preestablecidas de entre 300 y 3000 mm, y dispuestos perpendicularmente a nuestro medidor, bajo diferentes condiciones de temperatura y humedad. El número de pulsos por medición es de 8, mientras que cada prueba consistía en la realización de 100 mediciones de distancia, con unas 5 centésimas de segundo entre cada medición individual, excepto la última de las pruebas, que se detallará posteriormente. Para conocer el error del medidor hemos realizado un total de 60 muestras, bajo un rango de temperaturas entre 10 y 35ºC, realizados en un ambiente cerrado e intentando evitar corrientes de aire o gradientes de temperatura que pudieran afectar a las mediciones.

Mostramos en primer lugar los resultados gráficos de una muestra aleatoria de 16 de estas muestras, pudiendo observar la distribución de mediciones, el valor mínimo, máximo y de mediana de cada set de mediciones. El objetivo será conocer cómo se distribuyen las mediciones y qué estadístico es el más adecuado para estimar el valor más verosímil de los sets de medidas.

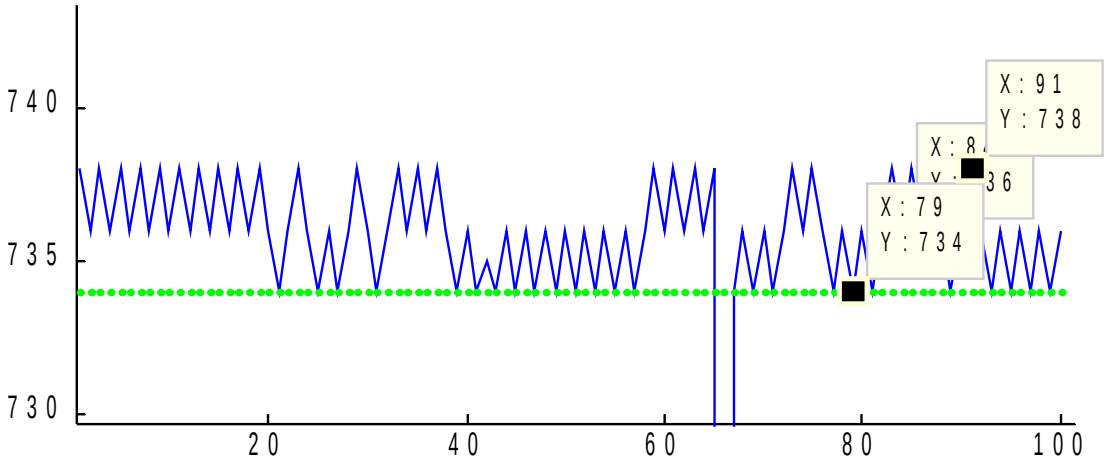
Distancia real: 400 mm; Tª= 22ºC, error de deriva sin corregir: +29 mm



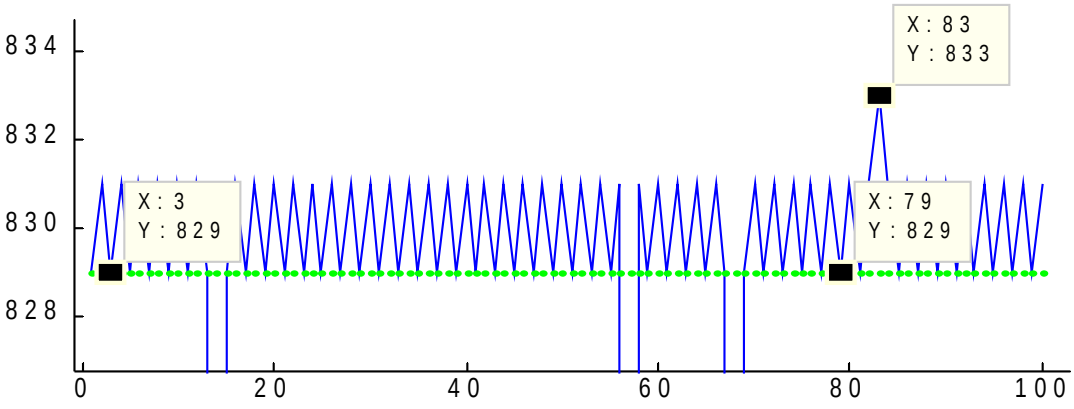
Distancia real: 500 mm; Tª= 19ºC, error de deriva sin corregir: +31 mm



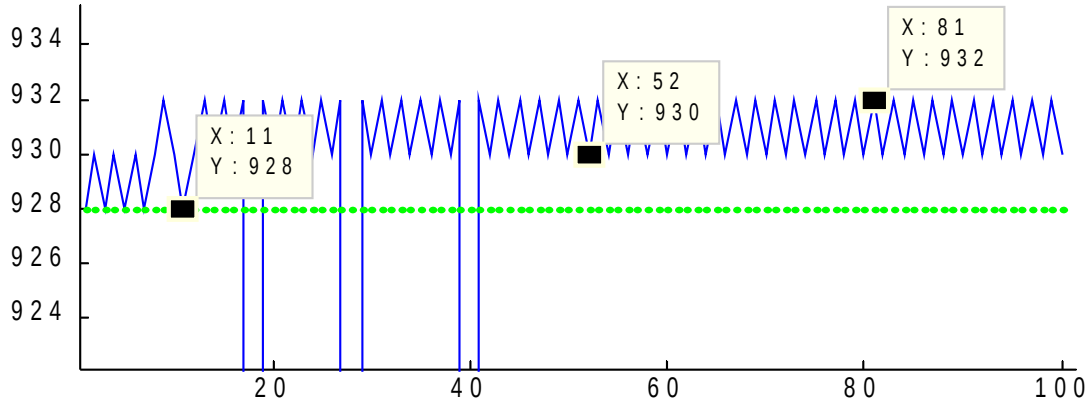
Distancia real: 700 mm, Tª= 31ºC, error de deriva sin corregir: +36mm



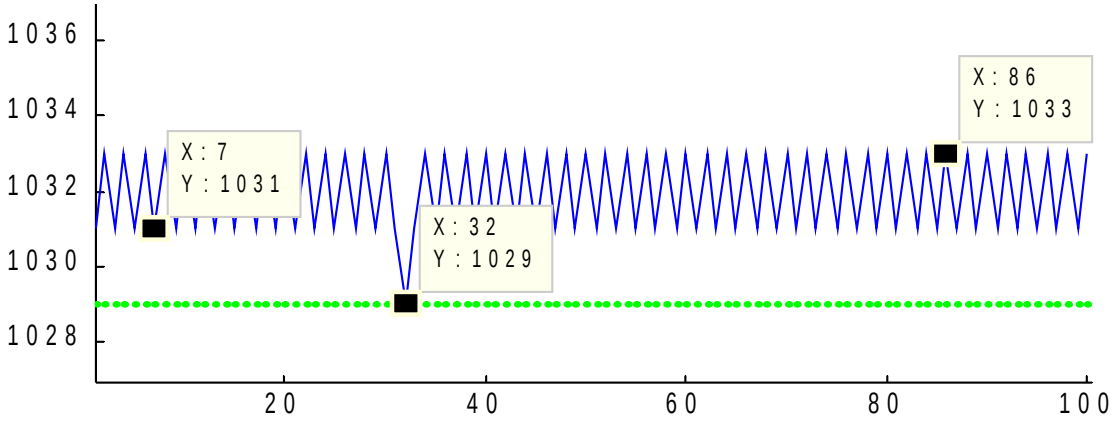
Distancia real: 800 mm, Tª= 25ºC, error de deriva sin corregir: +29 mm



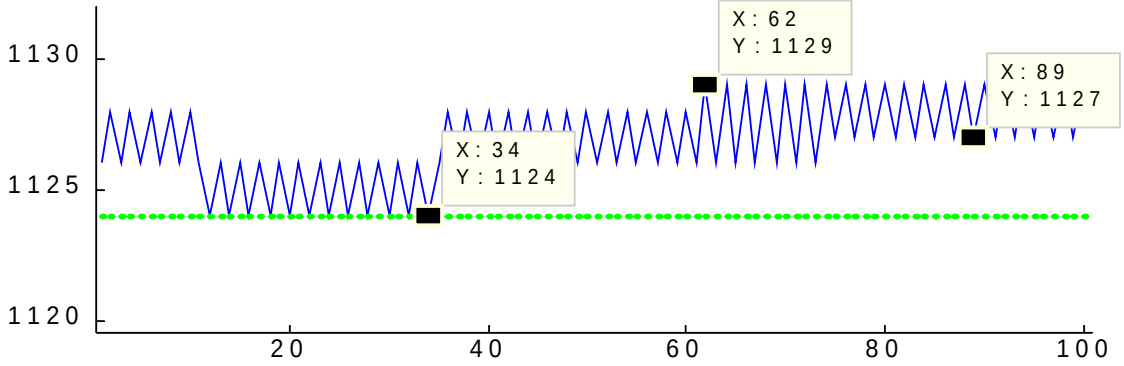
Distancia real: 900 mm; Tª= 24ºC, error de deriva sin corregir: +30 mm



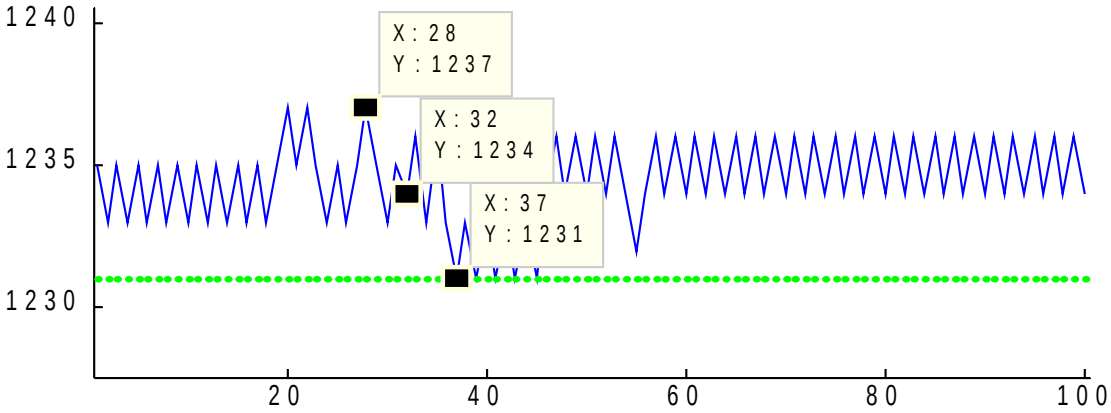
Distancia real: 1000 mm; Tª= 33ºC, error de deriva sin corregir: +31mm



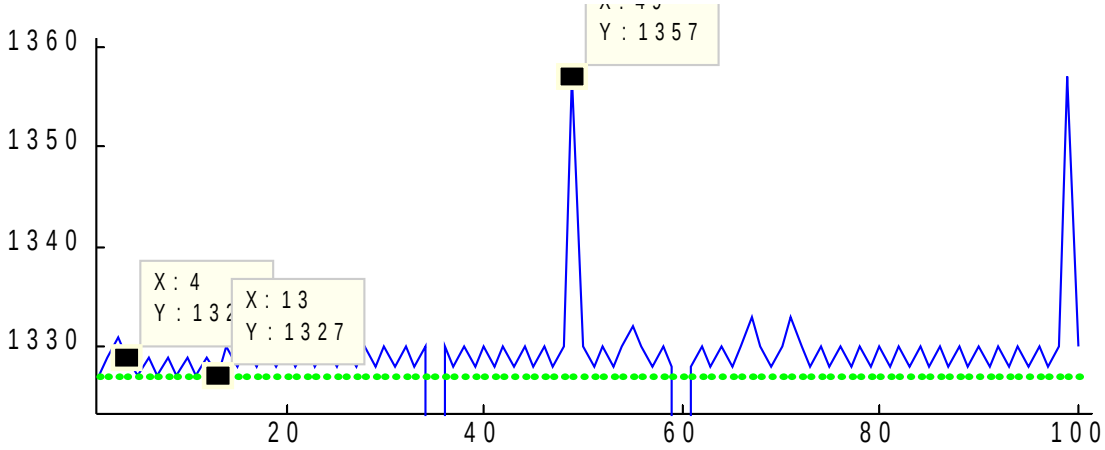
Distancia real: 1100 mm;   Tª= 17ºC, error de deriva sin corregir: +27 mm



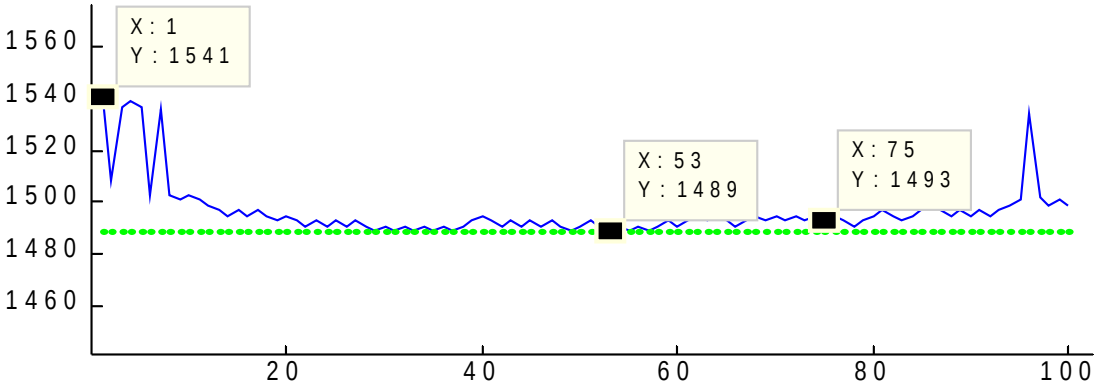
Distancia real: 1200 mm; Tª= 29ºC, error de deriva sin corregir: +34 mm



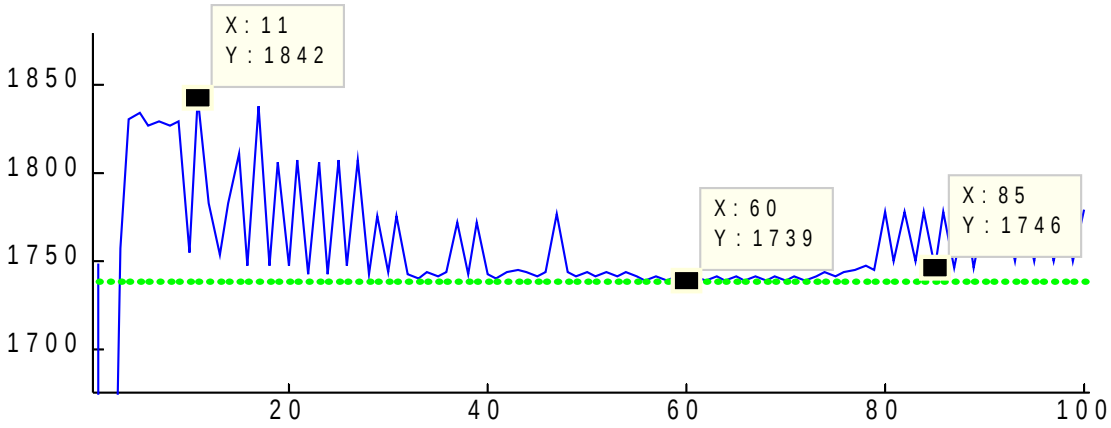
Distancia real: 1300 mm; Tª= 29ºC, error de deriva sin corregir: +29 mm



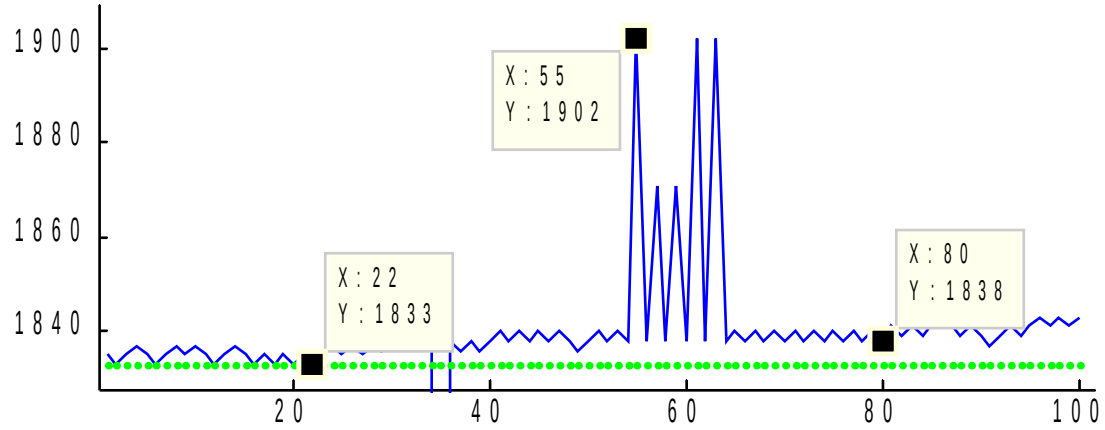
Distancia real: 1450 mm; Tª= 19ºC, error de deriva sin corregir: +43 mm



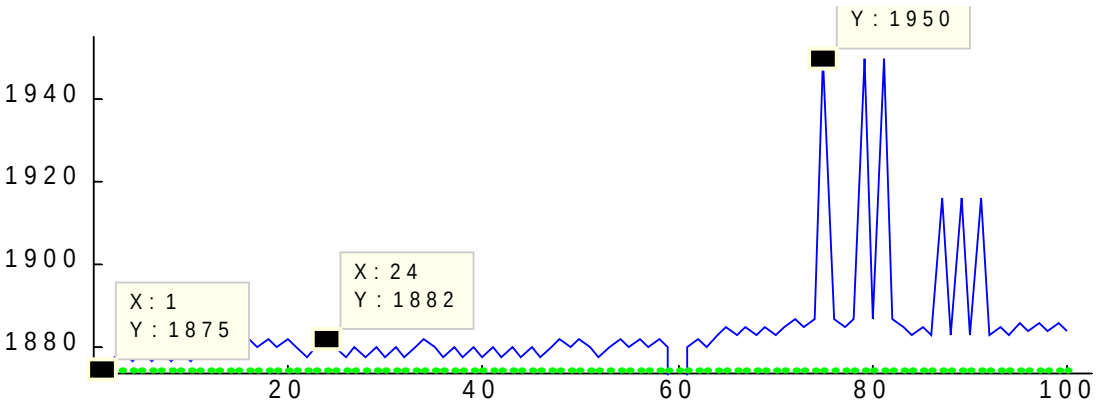
Distancia real: 1700 mm; Tª= 24ºC, error de deriva sin corregir: +46 mm



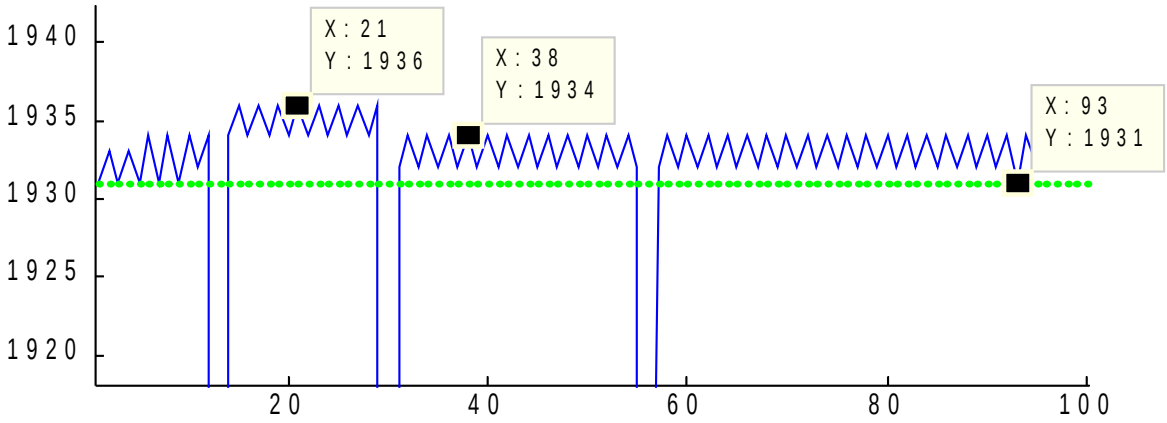
Distancia real: 1800 mm; Tª= 11ºC, error de deriva sin corregir: +38 mm



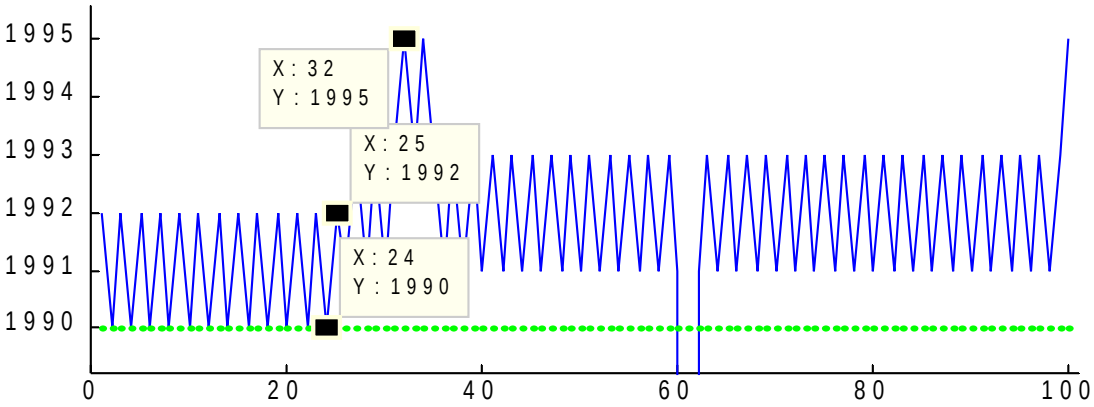
Distancia real: 1850 mm; Tª= 12ºC, error de deriva sin corregir: +32 mm



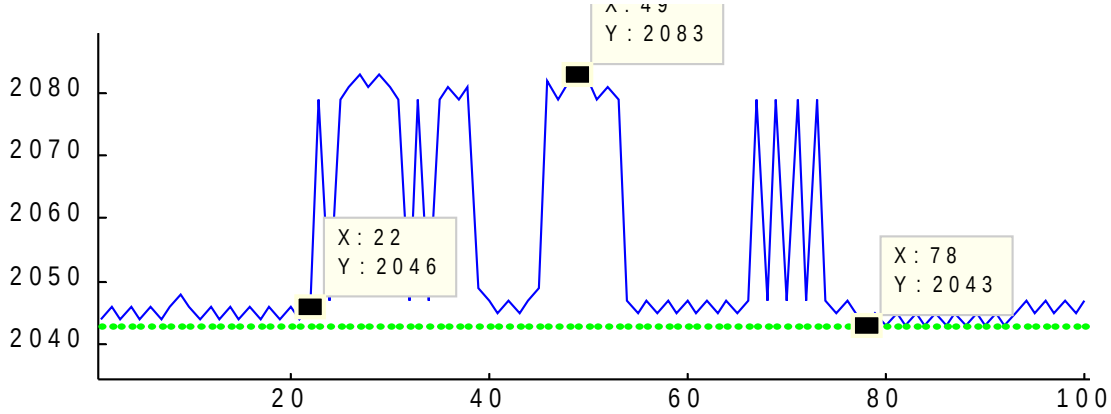
Distancia real: 1900 mm; Tª= 23ºC, error de deriva sin corregir: +34 mm



Distancia real: 1950 mm; Tª= 30ºC, error de deriva sin corregir: + 42 mm



Distancia real: 2000 mm; Tª= 34ºC, error de deriva sin corregir: + 46 mm



Para un mejor análisis, pasaremos a incluir estos resultados en una tabla, incluyendo la temperatura, valores máximo y mínimo, media y el retardo resultante.

Figura 5.1: tabla de resultados

Distancia real	Temperatura	Valor mínimo(mm)	Valor máximo(mm)	Valor de mediana(mm)	Retardo (mm)
400	22	427	431	429	29
500	19	529	535	531	31
700	31	734	738	736	36
800	25	829	833	829	29
900	24	928	932	930	30
1000	33	1029	1033	1031	31
1100	17	1124	1129	1127	27
1200	29	1231	1237	1234	34
1300	29	1329	1357	1329	29
1450	19	1489	1541	1493	43
1700	24	1739	1842	1746	46
1800	11	1833	1902	1838	38
1850	12	1875	1952	1882	48
1900	23	1931	1936	1934	34
1950	30	1990	1995	1992	42
2000	34	2043	2083	2046	46

Podemos observar varios aspectos interesantes de los resultados obtenidos. En primer lugar, la uniformidad de los resultados; viendo cualquiera de las pruebas efectuadas, se observa que para cada medición los valores que más se repiten están en un rango comprendido entre ± 4 mm respecto al valor de mediana, encontrándose en ocasiones picos que o bien son puntuales o bien aun siendo presentando elevaciones o descensos sostenidos no son representativos del conjunto de resultados obtenidos.

Observamos por último que el número de medidas fallidas es prácticamente nulo, no superando nunca el 3% de las medidas totales realizadas, y siendo del orden del 0.75% para todo el conjunto de pruebas realizado.

Resulta apropiado, por tanto, suponer que el valor de mediana del conjunto de resultados sea el estadístico que nos da la distancia más verosímil, y es esta la razón por la que se usará cuando queremos realizar conjuntos de mediciones. Pasamos ahora a considerar el set completo de 60 mediciones, realizadas en el rango de los 400 a 3000 mm.

Figura 5.2: tabla de resultados totales

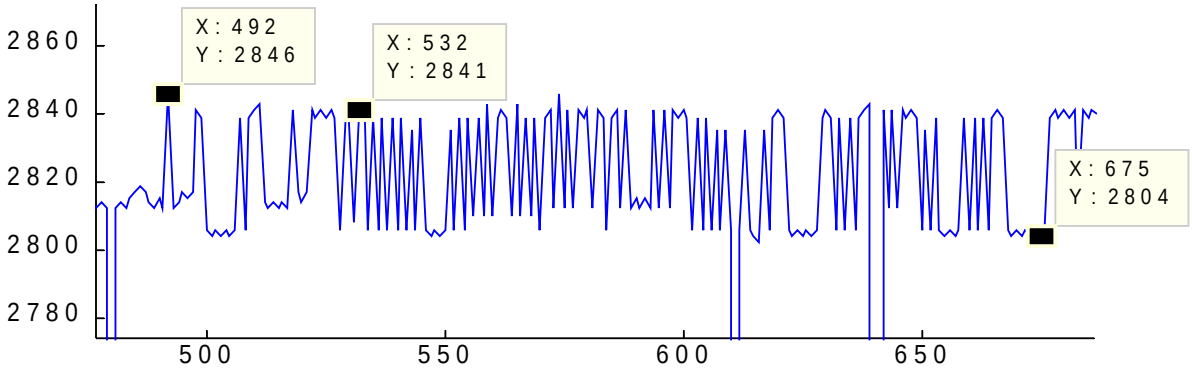
Distancia real	Temperatura	Valor de mediana(mm)	Retardo (mm)	Distancia real	Temperatura	Valor de mediana(mm)	Retardo (mm)
400	22	429	29	1550	19	1596	46
400	25	427	27	1700	24	1746	46
450	18	479	29	1700	31	1729	29
450	12	484	34	1700	35	1730	30
500	19	531	31	1800	11	1838	38
500	31	530	30	1800	15	1835	35
500	11	537	37	1800	28	1836	36
600	26	634	34	1800	35	1835	35
700	31	736	36	1850	12	1882	48
700	30	734	34	1850	35	1883	33
800	25	829	29	1900	23	1934	34
800	10	836	36	1900	12	1938	38
900	24	930	30	1950	30	1992	42
900	16	927	27	1950	11	1981	31
900	35	940	40	2000	34	2046	46
1000	33	1031	31	2000	22	2031	31
1000	27	1029	29	2100	27	2136	36
1050	17	1079	29	2200	13	2246	46
1100	17	1127	27	2200	10	2234	34
1100	24	1134	34	2250	14	2941	41
1100	16	1138	38	2400	31	2445	45
1200	29	1234	34	2500	14	2548	48
1200	11	1237	37	2500	18	2536	36
1300	29	1329	29	2600	20	2636	36
1300	14	1342	42	2700	27	2741	41
1300	20	1341	41	2800	21	2837	37
1400	33	1430	43	2850	25	2898	48
1450	15	1485	35	2900	15	2949	49
1450	19	1493	43	3000	19	3032	32
1450	11	1485	35	3000	28	3030	30

Observando el conjunto completo de mediciones podemos comprobar que es destacable la poca variación existente entre el valor real y el valor de mediana obtenido para cada test a diferentes distancias y, sobre todo, cómo esta diferencia se mantiene sin pocas variaciones ante las diferentes pruebas realizadas. Valiéndonos de la suposición de que las medidas obtenidas siguen una distribución normal, podemos considerar que el valor adecuado del retardo se corresponde con $retardo = \mu \mp 2\sigma = 36.12 \mp 12.335\, mm$, con un intervalo de confianza del 95 % y siempre que asumamos como correcta la suposición de distribución normal.

Será este por tanto el valor que incluiremos en el error de deriva de la distancia, teniendo pues un medidor con un error de precisión de ±12 mm y con un error de retardo de 36mm. Observamos por último que el número de medidas fallidas es prácticamente nulo, no superando nunca el 3% de las medidas totales realizadas, y siendo del orden del 0.75% para todo el conjunto de pruebas realizado.

La última prueba que se realizó tenía por objetivo comprobar la repetibilidad de los resultados; la prueba consistió en realizar un conjunto de 1000 medidas ante una pared situada a 2800 mm, con un tiempo entre cada medición de 5 segundo, lo que implicó una duración de la prueba de 83 minutos. El objetivo no era tanto obtener el retardo ni la distribución de los errores sino comprobar que los resultados de la medición eran coherentes a lo largo del tiempo, o por el contrario fluctuaban ante un uso prolongado del dispositivo de medición o ante cambios en las condiciones ambientales. Durante el experimento se hizo variar la temperatura ambiente desde los 15 a los 30ºC, y los resultados fueron los siguientes

Distancia real: 2800 mm; error de deriva sin corregir: + 41 mm



Como puede observarse en los resultados, no se percibe ninguna distorsión apreciable en función del tiempo en la distribución ni desviación de las medidas frente al valor de mediana, lo que nos lleva a concluir que nuestro medidor se comporta de forma robusta ante diferentes condiciones ambientales y conserva su precisión y resolución tras un tiempo continuado de funcionamiento.

5.2 Rango de trabajo y precisión del dispositivo medidor

El rango de trabajo viene determinado en su parte inferior por la limitación comentada en el capítulo 4 de lapso mínimo entre emisión y recepción, que es por defecto de unos 6.8 cm. En lo que se refiere al alcance máximo, hemos podido realizar pruebas frente a objetos de 50x50 cm^2 situados a distancias límite de 4.5 metros con resultados positivos para trenes de ondas de 18 pulsos y con un porcentaje de medidas fallidas (sin retorno de eco) superiores a un 75%. En estos casos conseguimos una precisión de ± 20 mm y un error de deriva de 50 mm.

Para distancias superiores a estos 4.5 metros, el aumento del número de pulsos no consigue un aumento apreciable de la máxima distancia que puede medir, mientras que la precisión de las medida se deteriora rápidamente. Hay que recordar que como vimos en el capítulo 4 la precisión de las mediciones depende enormemente del número de pulsos que se envían así como de las condiciones del medio en el que se desenvuelve el aparato, por lo que establecer una precisión y una desviación fijas para un rango amplio de mediciones resulta muy complicado además de poco práctico. Es por ello por lo que decidimos permitir la opción de modificar el retardo así como el número de pulsos, a fin de adaptarlo a diferentes condiciones de uso.

Mostramos a continuación una tabla en la que se incluyen el rango de funcionamiento del medidor así como los valores recomendados de número de pulsos y retardo correspondiente.

Figura 5.3: tabla de resultados

Rango de trabajo	Error	Número de pulsos recomendado	Retardo recomendado(mm)
6-1000mm	±6mm	6	25mm
1000-3000mm	±12mm	8	36mm
3000-4500mm	±20mm	18	50mm

Se ha decidido tomar como parámetros por defecto los valores de número de pulsos y retardo recomendados para distancias entre 1000 y 3000 mm, buscando así un compromiso entre exactitud en las mediciones y alcance máximo de distancia.

5.3 Resumen de prestaciones y características del dispositivo medidor

Procedemos por último a resumir las características más importantes del medidor de distancias, incluyendo consumo, protocolos de comunicación, alcance, resolución ,etc...:

Figura 5.3: características del medidor de ultrasonidos

Características	Valor
Consumo	5 Voltios, 500mW (100 mA máximo)
Alimentación	Dual: 4 baterías AAA y/o USB
Interfaces de comunicación	USB-CDC (virtual COM) RS-232 Manual
Alcance mínimo	68 mm
Alcance máximo	4500 mm
Precisión	±6 mm para distancias < 3000mm (valor para distancias cortas) ±12 mm para distancias < 4000mm (valor por defecto) ±20 mm para distancias ~4500mm (valor para distancias largas)
Resolución	1mm
Prestaciones	Autocalibración a partir de la temperatura ambiente. Obtención de múltiples mediciones con una sola orden. Mediciones con filtro de mediana. Incluye opciones de calibración fina del medidor. Lectura de temperatura con precisión de 0.25 ºC
Área	7.86x9.65 cm^2
Altura	3 cm

6- ANEXOS

En este capítulo se incluirá el código fuente del programa, dividido en sus diferentes ficheros, los esquemas y PCB's de los diferentes circuitos electrónicos así como los scripts de matlab que permiten el manejo del medidor mediante conexión USB/RS-232.

6.1 Código fuente del programa

6.1.1. Fichero main.jal

```
-- PROYECTO Javier Bermejo López: medidor de distancias por ultrasonido con pic18f4550

include 18f4550
pragma target clock      48_000_000

-- añadido de un offset
pragma bootloader loader18 4096

-- fuses
pragma target PLLDIV      P3        -- divide by 3 - 12MHZ_INPUT
pragma target CPUDIV      P2        -- OSC1_OSC2_SRC_1_96MHZ_PLL_SRC_2
pragma target USBPLL      F48MHZ    -- CLOCK_SRC_FROM_96MHZ_PLL_2
pragma target OSC         HS_PLL
pragma target FCMEN       DISABLED
pragma target IESO        DISABLED
pragma target PWRTE       DISABLED   -- power up timer
pragma target VREGEN      ENABLED    -- USB voltage regulator
pragma target VOLTAGE     V20        -- brown out voltage
pragma target BROWNOUT    DISABLED   -- no brownout detection
pragma target WDTPS       P32K       -- watch dog saler setting
pragma target WDT         DISABLED   -- no watchdog
pragma target CCP2MUX     pin_C1     -- CCP2 pin
pragma target PBADEN      DIGITAL    -- digital input port<0..4>
```

```
pragma target LPT1OSC      LOW_POWER   -- low power timer 1
pragma target MCLR         EXTERNAL    -- master reset on RE3
pragma target STVR         DISABLED    -- reset on stack over/under flow
pragma target LVP          DISABLED    -- no low-voltage programming
pragma target XINST        ENABLED     -- extended instruction set
pragma target DEBUG        DISABLED    -- background debugging
pragma target CP0          DISABLED    -- code block 0 not protected
pragma target CP1          DISABLED    -- code block 1 not protected
pragma target CP2          DISABLED    -- code block 2 not protected
pragma target CP3          DISABLED    -- code block 3 not protected
pragma target CPB          DISABLED    -- bootblock code not write protected
pragma target CPD          DISABLED    -- eeprom code not write protected
pragma target WRT0         DISABLED    -- table writeblock 0 not protected
pragma target WRT1         DISABLED    -- table write block 1 not protected
pragma target WRT2         DISABLED    -- table write block 2 not protected
pragma target WRT3         DISABLED    -- table write block 3 not protected
pragma target WRTB         DISABLED    -- bootblock not write protected
pragma target WRTD         DISABLED    -- eeprom not write protected
pragma target WRTC         DISABLED    -- config not write protected
pragma target EBTR0        DISABLED    -- table read block 0 not protected
pragma target EBTR1        DISABLED    -- table read block 1 not protected
pragma target EBTR2        DISABLED    -- table read block 2 not protected
pragma target EBTR3        DISABLED    -- table read block 3 not protected
pragma target EBTRB        DISABLED    -- boot block not protected



var volatile bit MODO_DESARROLLO       =true
const byte NUMERO_MEDIDAS_MAXIMO       =128
var volatile byte numero_medidas--     =64
var volatile word set_medida[NUMERO_MEDIDAS_MAXIMO]



--sinonimos(alias) de los pines asociados a los leds de estado.
alias pinprueba1           is pin_E0         --configurar pin asociado a cambios de estado
alias pinprueba1_direction   is pin_E0_direction
alias pinprueba2           is pin_E1         --configurar pin asociado a cambios de estado
alias pinprueba2_direction   is pin_E1_direction
```

```
alias pinprueba3           is pin_e2
alias pinprueba3_direction   is pin_E2_direction



--CONFIGURACION E/S PINES
PINPRUEBA1_DIRECTION=OUTPUT
PINPRUEBA2_DIRECTION=OUTPUT
PINPRUEBA3_DIRECTION=OUTPUT



--PUESTA A 0 DE LOS PINES
PINPRUEBA1=0
PINPRUEBA2=0
pinprueba3=0

-- librerias añadidas
include delay                    --libreria estándar: retardos
include print                                --librería estándar: funciones de lectura-escritura
include configuracion_eeprom          --eeprom
include configuracion_ADC           --ADC

include interfaz_comunicacion          --comunicacion USB-serial-manual
include menu_medicion                --submenu de medicion
include menu_configuracion_lectura      --submenu de configuracion-lectura



---comienzo de la rutina principal del microcontrolador: configracion de timers y pines
INTCON_GIE = true   --habilitamos de manera general las interrupciones
INTCON_PEIE = true  --también las asociadas a periféricos
ENABLE_DIGITAL_IO() --todas las entradas son por defecto digitales



--PUESTA A 0 DE LAS VARIABLES GLOBALES
ESTADO=0
```

```
--limpia los registros del array de mediciones
var word long=count(set_medida)
var word i
for long using i loop
   set_medida[i]=0
end loop

lcd_clear_screen()
escribir_lcd(str_lcd_conf_usb_1)
lcd_cursor_blink_display(1, 1, 1)


--configuramos las conexiones USB y RS232 al iniciar el dispositivo.
configurar_conexion_inicio()



forever loop --bucle principal

   --asignamos pines en funcion del tipo de conexion
   PINPRUEBA1=ON
   pinprueba2=selector_interfaz_usb
   pinprueba3=selector_interfaz_serie

   VAR VOLATILE WORD DISTANCIA=0

   -- reconfigura cíclicamente el dispositivo en función del estado de los interfaces de comunicación
   reconfigurar_conexiones_hardware()

   -- activa las funciones del modo manual.
   menu_manual (estado)

   -- ÁRBOL DE ACCIONES: EN FUNCION DEL VALOR QUE ADOPTE LA VARIABLE GLOBAL ESTADO,
   -- SE REALIZARÁ UNA U OTRA ACCIÓN.

   --      MEDICION_SIMPLE_STRING:
```

```
   if (estado==medicion_simple) then
      BLOCK
      lcd_clear_screen()
      escribir_lcd(str_medicion_simple_1)
      lcd_cursor_blink_display(1, 1, 1)
      menu_medicion_simple(numero_pulsos,distancia)
      id1="M"
      dato2=distancia
      lcd_cursor_blink_display(0, 0, 1)
      lcd_cursor_position(1,0)
      print_word_dec(lcd,dato2)
      lcd=" "
      lcd="m"
      lcd="m"
      formato_buffer_intermedio=0b00_010_1_01
      flag_int_USB=on
      asm nop
      END BLOCK


--      MEDICION_SIMPLE_WORD:
   elsif (estado==medicion_simple_word) then
      block
      lcd_clear_screen()
      escribir_lcd(str_medicion_simple_1)
      lcd_cursor_blink_display(1, 1, 1)
      menu_medicion_simple(numero_pulsos,distancia)
      id1="m"
      dato2=distancia
      lcd_cursor_blink_display(0, 0, 1)
      lcd_cursor_position(1,0)
      print_word_dec(lcd,dato2)
      lcd="m"
      lcd="m"
      formato_buffer_intermedio=0b00_010_0_01
      flag_int_USB=on
      asm nop
      END BLOCK
```

```
--      MEDICION_MULTIPLE_BATCH: salida del conjunto completo de mediciones, formato string
   elsif(estado==medicion_multiple_batch) then
      BLOCK
      lcd_clear_screen()
      escribir_lcd(str_medicion_batch_1)
      lcd_cursor_blink_display(1, 1, 1)
      if(menu_medicion_multiple_batch(numero_pulsos)) then
         id1="B"
         lcd_cursor_blink_display(0, 0, 1)
         LCD="h" lcd="e" lcd="c" lcd="h" lcd="o"
         dato1=numero_medidas
         formato_buffer_intermedio=0b00_101_1_01
         flag_int_USB=on
         asm nop
         formato_buffer_intermedio=0
      end if
      END BLOCK

--      MEDICION_MULTIPLE_BATCH_WORD: salida del conjunto completo de mediciones, formato word
   elsif(estado==medicion_multiple_batch_WORD) then
      BLOCK
      lcd_clear_screen()
      escribir_lcd(str_medicion_batch_1)
      lcd_cursor_blink_display(1, 1, 1)
      if(menu_medicion_multiple_batch(numero_pulsos)) then
         id1="b"
         lcd_cursor_blink_display(0, 0, 1)
         LCD="h" lcd="e" lcd="c" lcd="h" lcd="o"
         dato1=numero_medidas
         formato_buffer_intermedio=0b00_101_0_01
         flag_int_USB=on
         asm nop
         formato_buffer_intermedio=0
      end if
      END BLOCK
      --end if
```

```
--    MEDICION MULTIPLE_MEDIA: devuelve la mediana del conjunto de mediciones, formato string
    elsif(estado==medicion_multiple_media) then
      BLOCK
      var word mediana=1
      var byte numero_ceros=0
      if (menu_medicion_multiple_media(numero_pulsos,numero_ceros,mediana)) then
        id1="N"
        dato1=numero_ceros
        dato2=mediana
        formato_buffer_intermedio=0b00_011_1_01
        lcd_clear_screen()
        lcd="D" lcd="(" print_byte_dec(lcd,dato1) lcd=")" lcd=":"  lcd=" "
        lcd_cursor_position(1,0)
        print_word_dec(lcd,dato2) lcd=" " lcd="m" lcd="m"
        flag_int_USB=on
        asm nop
        formato_buffer_intermedio=0
      end if
    END BLOCK


--    MEDICION MULTIPLE_MEDIA_WORD: devuelve la mediana del conjunto de mediciones, formato word
    elsif(estado==medicion_multiple_media_WORD) then
      BLOCK
      var word mediana=0
      var byte numero_ceros=0
      if(menu_medicion_multiple_media(numero_pulsos,numero_ceros,mediana)) then
        id1="n"
        dato1=numero_ceros
        dato2=mediana
        formato_buffer_intermedio=0b00_011_0_01
        lcd_clear_screen()
        lcd="D" lcd="(" print_byte_dec(lcd,dato1) lcd=")" lcd=":"  lcd=" "
        lcd_cursor_position(1,0)
        print_word_dec(lcd,dato2) lcd=" " lcd="m" lcd="m"
        flag_int_USB=on
        asm nop
        formato_buffer_intermedio=0
```

```
      end if
    END BLOCK

elsif(estado==medicion_distancia_modo_manual) then
      BLOCK
      var word mediana=0
      var byte numero_ceros=0
      if(menu_medicion_multiple_media(numero_pulsos,numero_ceros,mediana)) then
        id1="n"
        dato1=numero_ceros
        dato2=mediana
        formato_buffer_intermedio=0b00_011_0_01
        lcd_clear_screen()
        lcd="D" lcd="[" print_byte_dec(lcd,dato1) lcd="]" lcd=":"  lcd=" "
        lcd_cursor_position(1,0)
        print_word_dec(lcd,dato2) lcd=" " lcd="m" lcd="m"
        flag_int_USB=on
        asm nop
        formato_buffer_intermedio=0
      end if
    END BLOCK



    elsif(estado==medicion_temperatura_modo_manual) then
      BLOCK
      temperatura=obtener_temperatura_cent(canal_ADC_sensor_temperatura,deriva_temperatura)
      data_eeprom_write(offset_eeprom_temperatura,temperatura)

      --aseguramos el refresco de la comunicacion usb
      formato_buffer_intermedio=0
      flag_int_USB=on
      asm nop



      --mostramos por lcd los datos solicitados
      lcd_clear_screen()
      var sword aux_temp=temperatura
```

```
      if (aux_temp<0) then
         lcd="-"
         aux_temp=-aux_temp
         end if

      escribir_lcd(str_lectura_temperatura)
      print_word_dec(lcd,aux_temp/100)
      lcd="."
      if (aux_temp%100<10) then
         lcd="0"
      end if
      print_word_dec(lcd,aux_temp%100)
      lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "

      END BLOCK


   --CONFIGURACION: configura el dispositivo
   elsif (ESTADO==CONFIGURACION) then
      BLOCK
      formato_buffer_intermedio=0
      submenu_configuracion()
      flag_int_USB=on
      asm nop
   END BLOCK

   --LECTURA: muestra a peticion del usuario los parámetros del dispositivo
   elsif (ESTADO==LECTURA) then
      BLOCK
      formato_buffer_intermedio=0
      submenu_lectura()
      flag_int_USB=on
      asm nop
   END BLOCK

--testeo
 elsif (estado=="x") then
   block
```

```
      flag_int_usb=off

        --configuracion timer_pwm e int. asociadas
      valor_timer_PWM=255-FREC_PWM_REAL   --determina la frecuencia de la señal pwm (40KHz)
                registro_timer_PWM=0B0_1001_000     --Se configura el timer_pwm con preescaler=2. Aun no se activa
                habilitar_int_PWM=ON              --Se habilita la interrupcion asociada al desbordamiento
      activar_timer_PWM=on

      forever loop
      end loop



      flag_int_USB=on
      asm nop
      END BLOCK




--      ABORTAR:
--      elsif estado==abortar     then
--         BLOCK
--         PINPRUEBA2=0
--         activar_timer_PWM=oFF
--         habilitar_int_RECEPTOR=OFF
--      consigna_escritura=4
--         flag_int_USB=on
--         asm nop
--      END BLOCK
   end if
end loop
```

6.1.2. Fichero configuracion_adc.jal

```
--CONFIGURACION DEL CONVERSOR ANALOGICO DIGITAL
-- parametros que definen el sensor de temperatura analogico
-- la relacion voltaje-temperatura viene dada por la ecuacion V=Vo+T*Co, donde
-- Vo es el voltaje a 0ºC en Voltios, T es la temperatura en ºC, y C es el coeficiente de temperatura en V/ºC.
-- para obtener la temperatura tenemos T=V/Co-Vo/Co. Como la tension se recibe en escala 0-1024 bits,
-- y la temperatura se medirá en centésimas de ºC cambiamos de escala, teniendo
--
-- T[ºC/100] = V[0-1024 bits]*5/1024*100/Co - Vo*100/Co [ecuacion 1] => agrupando constantes
--
-- T = V*m - To, donde
-- V es el valor de tension en escala de 10 bits,
-- T la temperatura en centésimas de ºC
-- To(offset temperatura)       = Vo*100/Co,
-- m(relacion voltaje/[ºC /100]) = 5/1024*100/Co,
-- Para el sensor de microchip MCP9701A, Vo=0.4V, Co=19.53e-3V, => To~2048, m~25
--
--
--
-- obtencion de la temperatura a partir de la lectura del sensor analogico de temperatura
-- Para obtener la velocidad del sonido se recurre a la ecuacion S [m/s] = 331.58 + 0.62T(ºC).
-- Pasando a cm/s, y sustituyendo T por la ecuacion 1, tenemos
--
-- S [m/s] = 33158 - 62*Vo/Co + V*62*5/1024/Co [ecuacion 2] => agrupando constantes
--
-- S = n*V + So, [ecuacion 2], donde
-- S es la velocidad del sonido en cm/s.
-- V es el valor de tension del sensor de Tª en escala 10bits
-- So (offset velocidad)       = 33158 - 62*V0/Co        ==31888     ~ 31/2
-- n (relacion voltaje/[cm/s])  = 62*5/1024/Co           ==15.501    ~ 3/5
-- Así, el resultado será
const word To_defecto                    =2048      --To
const byte coeficiente_temperatura_defecto      =25        --m
const word So_defecto                    =31888     --So
const byte num_coef_velocidad_defecto          =31        --n. el coeficiente es 15.5=31/2; después dividimos entre 2
```

```
const byte den_coef_velocidad_defecto          =2       --n. el coeficiente es 15.5=31/2; después dividimos entre 2

--si añadimos el efecto de la humedad, la ecuacion se amplía a
-- S [m/s] = 331.58*(T+ATs) +So + 0.015*HR[humedad relativa](escala 0-100)
--pasando a cm/s, y escala de humedad relativa de 0-255, nos queda
--
-- S [m/s] = 33158 - 62*Vo/Co + V*62*5/1024/Co  + 0.62*100*ATs + 0.015 * 100 * HR(escala 0-255) *100/255
--[ecuacion 3] =>
--
-- S = n*V + So + h*HR + e*ATs, [ecuacion 2], donde
-- S es la velocidad del sonido en cm/s.
-- V es el valor de tension del sensor de Tª en escala 10bits
-- So (offset velocidad)      = 33158 - 62*V0/Co        ==31888
-- n (relacion voltaje/[cm/s]) = 62*5/1024/Co           ==15.501     ~ 31/2
-- h (relacion HR/cm/s)       =100*100/255*0.015        ==0.5882     ~ 3/5
-- e (relacion centésimas de grado/cm/s) =0.62*100      ==62         ~ 5/8
-- Así, las constantes quedan:

const byte num_coef_efecto_humedad        =3       --3/5
const byte den_coef_efecto_humedad        =5       --3/5

const byte num_coef_efecto_temperatura    =5       --5/8
const byte den_coef_efecto_temperatura    =8       --5/8

;var word To
;var word So
;var byte coeficiente_temperatura
;var byte coeficiente_velocidad
;



;--LEEMOS LA EEPROM PARA OBTENER LOS VALORES ANTERIORMENTE SELECCIONADOS
;--
;--MODO DE DESARROLLO; NO USAMOS LA EEPROM PARA CARGAR LOS VALORES INICIALES,
-- SINO QUE USAMOS VALORES POR DEFECTO
;IF (defined(MODO_DESARROLLO)) THEN
;   To=To_defecto
;   coeficiente_temperatura=coeficiente_temperatura_defecto
```

```
;  So=So_defecto
;  coeficiente_velocidad=coeficiente_velocidad_defecto
;ELSE
;   data_eeprom_read_word(offset_eeprom_To,To)
;   data_eeprom_read_word(offset_eeprom_coeficiente_temperatura,coeficiente_temperatura)
;END IF


---configuracion previa ADC
const bit ADC_HIGH_RESOLUTION = true  --Alta resolucion
const byte ADC_NVREF = 0            --sin tension de referencia externa, se usará la que alimenta al micro
const byte ADC_NCHANNEL=1           --Un solo canal analogico
const byte canal_ADC_sensor_temperatura=0-- leeremos la temperatura de este canal

include adc

adc_init()--se inicializa la libreria



--CONVERTIR_ADC_TEMPERATURA recibe un valor de tension de 10 bits y calcula la temperatura correspondiente.
--las unidades serán centésimas de grado. Efectua la conversion segun la relacion temperatura tensión, que es
-- T[Temperatura](ºC) = V[Tension](10 bits)*m - To. No se tiene en cuenta por tanto el efecto del error de deriva
-- del sensor. Esto se tendrá en cuenta en la función obtener_temperatura_cent.
function convertir_ADC_temperatura(word in dato_adc) return sword is
   pragma inline


   --por defecto, los resultados en alta resolucion los justifica a la izquierda;
   --como la resolucion es de 10 bits, desplazamos 6 bits (2^6=64) a la derecha
   if (ADC_HIGH_RESOLUTION ==true)then

      dato_adc=dato_adc/64

   end if

   var sword temperatura=sword(dato_adc)*coeficiente_temperatura_defecto - To_defecto
   return temperatura
```

```
end function

-----------------------------------------------------------------------------------------------
--
--CONVERTIR_ADC_VEL_SONIDO recibe un valor de tensión en 1024 bits y calcula la velociada del sonido correspondiente.
--las unidades serán cm/s
--la relacion velocidad del sonido / temperatura es S[vel_sonido(m/s)]=331.58 + 0.62 T[temperatura](ºC)
--no tenemos en cuenta en esta conversion los efecto de la h.relativa ni de la deriva de la temperatura; esto se tendrá
--en cuenta en la funcion obtener_vel_sonido_cent
--sustituyendo T por mV-To y agrupando términos, tenemos  S = n*T + So
function convertir_ADC_vel_sonido(word in dato_adc) return word is
   pragma inline

   --por defecto, los resultados en alta resolucion los justifica a la derecha;
   --corregimos y enviamos a la izquierda
   if (ADC_HIGH_RESOLUTION ==true) then

      dato_adc=dato_adc/64

   end if

   var word vel_sonido=So_defecto + (dato_adc*num_coef_velocidad_defecto)/den_coef_velocidad_defecto
   return vel_sonido
end function

-----------------------------------------------------------------------------------------------
--
--FUNCION OBTENER_TEMPERATURA_CENT determina la temperatura a partir del valor que haya en el canal analogico
--elegido.
--Corrige el error de deriva y lo incluye en el resultado, de acuerdo con la ecuación
--Tr[temperatura real] = ATs[error de deriva] +  V*m - To
--Devuelve la temperatura en centesimas de grado, en forma de entero de 16 bits con signo.
```

```
--El rango de temperatura es de -20.51ºC hasta 235ºC
function obtener_temperatura_cent(byte in canal_ADC, sword in deriva_temperatura) return sword is

   var word  dato_adc      = adc_read(canal_ADC)                    --leemos del canal analogico correspondiente
   var sword temperatura   = convertir_ADC_temperatura(dato_adc)        --realizamos la conversion
   temperatura           = temperatura + deriva_temperatura           --corregimos el error de deriva
   return temperatura
end function



-------------------------------------------------------------------------------------------------
--
--FUNCION OBTENER_VELOCIDAD_CENT determina la velocidad del sonido a partir del valor que haya en el canal
--analogico elegido. Devuelve la velocidad en cm/s, en forma de entero de 16 bits sin signo.
--Corrige los errores debido a la humedad relativa y a la deriva de medición del sensor de temperatura, de acuerdo con
--la ecuacion:   S = n*(T+ATs) + So + HR[Humedad relativa]*k
--El rango de velocidad estará entre 16016 y 47760 cm/s
function obtener_vel_sonido_cent(byte in canal_ADC,sword in deriva_temperatura,byte in humedad_relativa) return
word is

   var word  dato_adc  =   adc_read(canal_ADC)                 --leemos del canal analogico correspondiente
   var word vel_sonido =   convertir_ADC_vel_sonido(dato_adc)     --realizamos la conversion

   --corregimos error de temperatura
                           vel_sonido                                                =                vel_sonido       +
(num_coef_efecto_temperatura*deriva_temperatura)/den_coef_efecto_temperatura

   --corregimos error  de humedad relativa
   vel_sonido        =   vel_sonido + num_coef_efecto_humedad*sword(humedad_relativa)/den_coef_efecto_humedad
   return vel_sonido
end function
```

6.1.3. fichero configuracion_eeprom.jal

```
const word offset_eeprom_lapso_minimo            =0      --1 byte
```

```
const word offset_eeprom_numero_pulsos            =1      --1 byte
const word offset_eeprom_retardo                  =2      --2 byte
const word offset_eeprom_vel_sonido               =4      --2 byte
const word offset_eeprom_coeficiente_temperatura  =6      --1 byte
--const word offset_eeprom_To                     =7      --2 byte
const word offset_eeprom_temperatura              =9      --2 byte
const word offset_eeprom_deriva_temperatura       =11     --2 byte
const word offset_eeprom_humedad_relativa         =13     --2 byte
const word offset_eeprom_numero_medidas           =15     --1 byte
include pic_data_eeprom
```

6.1.4. fichero interfaz_comunicacion.jal

```
--LIBRERIA INTERFAZ_COMUNICACION: esta librería incluye todas las funciones, variables
-- constantes y procedimientos encargados de la interfaz de comunicacion puerto
-- serie, USB y manual



-------------------------------------------------------------------------------------------------
-- SELECTOR DE INTERFAZ: permite seleccionar automáticamente entre las diferentes interfaces
-- disponibles, en funcion de si existe conexion USB, serie o se ha activado el interruptor manual.



--variables de selección de interfaz de hardware;
var volatile byte selector_interfaz
var volatile bit selector_interfaz_serie    at selector_interfaz:0
var volatile bit selector_interfaz_usb      at selector_interfaz:1



--incializo las variables; al inicio no hay ninguna conexion establecida
selector_interfaz_serie                =false
selector_interfaz_usb                  =false



-------------------------------------------------------------------------------------------------
--INTERFAZ MANUAL: pines y variables asociados a la interfaz manual del medidor.
--Desde la interfaz manual solamente se podrá acceder a dos funciones de medicion;
-- 1. medicion multiple de distancia y filtro de mediana. ESTADO=MEDICION_MULTIPLE_media_WORD
-- 2. obtención de la temperatura.                       ESTADO=LEER_TEMPERATURA_MANUAL



-- constantes
const byte MEDICION_DISTANCIA_MODO_MANUAL   ="h"
const byte MEDICION_TEMPERATURA_MODO_MANUAL ="t"



--pines asociados a los botones del interfaz manual
alias pin_leer_temperatura              is  pin_d3
```

```
alias pin_cambio_modo_manual              is  pin_d1
alias pin_hacer_medicion                  is  pin_d4

alias pin_leer_temperatura_direction      is  pin_d3_direction
alias pin_cambio_modo_manual_direction    is  pin_d1_direction
alias pin_hacer_medicion_direction        is  pin_d4_direction

pin_leer_temperatura_direction            = input
pin_hacer_medicion_direction              = input
pin_cambio_modo_manual_direction          = input



--llamada a la librería encargada de la comunicacion por medio del LCD
include configuracion_LCD



------------------------------------------------------------------------------------------------------------------
--BUFFER INTERMEDIO DE DATOS. Este buffer de datos, de acceso comun para las funciones de escritura por puerto
--serie
-- y USB, permite el encapsulado de las funciones de escritura para ambos protocolos bajo una unica funcion de
-- escritura, de nombre escritura_usb_serial. El buffer de datos en sí esta constituido por un conjunto de variables
-- de tipo byte, word, y un array de variables de tipo word, que se detallan a continuacion.
--
-- id1, id2: variable tipo byte, donde se escribirán los caracteres de control que permitan identificar el tipo de dato
-- que está registrado en dato1,dato2,dato3.
-- dato1: variable tipo byte, donde se escribirán datos de 1 byte de longitud
-- dato2: variable tipo word, donde se escribirán datos de hasta dos bytes de longitud.
-- set_medida: array de variables de longitud NUMERO_MEDIDAS_MAXIMO, donde se almacenarán los resultados de
múltiples
--mediciones. podrán ser enviados en formato numérico o string, en función de la variable formato_dato definida
--abajo.
--
-- Así mismo, se precisa especificar, para cada petición de escritura, cuáles de los datos del buffer intermedio
-- deberán ser enviados a cualesquiera de los periféricos señalados. De ello se encarga la variable
-- formato_buffer_intermedio. También permite especificar el formato de los datos propiamente dicho, mediante la
-- variable formato_dato, que si es 1 enviará los datos en formato de cadena de caracteres y si es 0 en formato
```

```
--numérico big-endian.


--formato del buffer intermedio
var volatile byte   formato_buffer_intermedio =0
var volatile bit    mostrar_id1          at formato_buffer_intermedio:0 -- si es 1 permite el envío de id1
var volatile bit    mostrar_id2          at formato_buffer_intermedio:1 -- si es 1 permite el envío de id2
var volatile bit    formato_dato         at formato_buffer_intermedio:2 -- 1 => datos en string. 0=>numérico
var volatile bit    mostrar_dato1        at formato_buffer_intermedio:3 -- si es 1 permite el envío de dato1
var volatile bit    mostrar_dato2        at formato_buffer_intermedio:4 -- si es 1 permite el envío de dato2
var volatile bit    mostrar_dato3        at formato_buffer_intermedio:5 -- si es 1 permite el envío de dato3

--datos del buffer intermedio
var volatile byte   dato1
var volatile word   dato2
alias               dato_array           is set_medida --el array set_medida está definido en la funcion main
var volatile byte   id1
var volatile byte   id2



--------------------------------------------------------------------------------
-- INTERFAZ USB: constantes, funciones y variables asociadas la interfaz de comunicacion
-- por USB, mediante el protocolo CDC. Precisa de la libreria USB_SERIAL
--


--alias de los registros asociados con el refresco de la comunicacion USB
alias habilitar_int_USB                 is PIE1_TMR2IE
alias flag_int_USB                      is PIR1_TMR2IF
alias activar_timer_USB                 is t2con_tmr2on
alias valor_timer_USB                   is pr2
alias registro_timer_USB                is t2con


-- constantes asociadas a la configuracion USB
const  byte str_welcome[]               = "\nSONAR PIC 18F4550 USB-CDC"
```

```
const byte USB_STRING1[]                ="\nPFC Javier Bermejo López. Sonar PIC"
const byte USB_CDC_TX_BUFFER_SIZE       = 0x40
const byte USB_CDC_RX_BUFFER_SIZE       = 0x20


-- variables asociadas a la comunicacion serie-USB
var volatile byte CONTADORUSB           =0
var volatile byte ESTADO                =0
var volatile bit has_shown_welcome_msg  =false
var volatile bit permiso_RX_TX          =on
var volatile byte ch



--------------------------------------------------------------------------------------------
-- INTERFAZ rs232: constantes, funciones y variables asociadas la interfaz de comunicacion
-- por puerto serie, mediante el protocolo CDC. Precisa de la libreria USB_SERIAL
--

--constantes asociadas a la comunicacion rs-232.
const serial_hw_baudrate                = 19_200
const sennal_activacion_puerto_serie    ="y"


--configuracion interrupcion base de tiempo del usb
valor_timer_usb                 =10
registro_timer_USB              =0B0_1111_0_11
habilitar_int_USB               =ON
activar_timer_USB               =on



--declaracion de las librerías estandar asociadas
include usb_serial              --comunicacion serie-USB (emulacion puerto com virtual)
include serial_hardware     --comunicacion serie-232


 --inicializo los dos interfaces de comunicacion
```

```
serial_hw_init()
usb_serial_init()



-------------------------------------------------------------------------------------------------------------------


 -- lectura_SERIAL_USB encapsula las funciones de lectura para el puerto serie y USB en una única función, cuya
 -- estructura de entrada y salida es idéntica al de las funciones usb_serial_read y serial_hw_read; recibe como
 -- entrada un byte por referencia y devuelve como salida un bit que indica si efectivamente ha leido un bit del puerto
 -- serie/USB.
 -- A nivel interno, y dado que existen dos protocolos de comunicación accesibles en modo lectura, se elige qué
 -- periferico será objeto del trabajo de lectura en función de las variables de tipo bit selector_interfaz_serie y
 -- selector_interfaz_usb.
 -- Así mismo, tras la lectura se habilita siempre la interrupción asociada al refresco USB, a fin de permitir el
 -- refresco de la comunicacion usb.
-------------------------------------------------------------------------------------------------------------------
 function lectura_SERIAL_USB (BYTE OUT LEER_BUFFER) return bit is
   habilitar_int_usb=Off      --deshabilito la comunicacion USB-Serie para que no interfiera

   IF (selector_interfaz_serie & serial_hw_read( LEER_BUFFER )) THEN
      habilitar_int_USB=On      --rehabilito la comunicacion USB-serie
     return on

   ELSIF (selector_interfaz_USB & USB_SERIAL_READ( LEER_BUFFER )) THEN
      habilitar_int_USB=On      --rehabilito la comunicacion USB-serie
      return on
   else
      LEER_BUFFER=0
      habilitar_int_USB=On      --rehabilito la comunicacion USB-serie
      return off
   end if

end function



-------------------------------------------------------------------------------------------------------------------
 -- escritura_SERIAL_USB encapsula las funciones de escritura para el puerto serie y USB en una única función.
```

```
 -- Esta funcion ni devuelve ni recibe ningún parámetro, ya que los datos que enviará por puerto serie o USB y el
 -- formato de los mismos están recogidos en el buffer intermedio de datos, de acceso público.
-- Esta función envía los datos tanto por USB como por RS-232 independientemente del estado del selector de interfaz.
-------------------------------------------------------------------------------------------------------------
procedure escritura_serial_usb () is
   habilitar_int_USB=Off     --deshabilito la comunicacion USB-Serie para que no interfiera
   USB_SERIAL_FLUSH()        --refresco de la comunicación USB

   --escritura dato
   if formato_buffer_intermedio!=0 then

     --escritura del identificador primario
      if (mostrar_id1==0b1) then
         usb_serial_data=id1
         serial_hw_data=id1
      end if

     --escritura del identificador secundario
      if (mostrar_id2==0b1) then
         usb_serial_data=id2
         serial_hw_data=id2
      end if

      --envío de los datos en formato string
      if (formato_dato==1) then

         BLOCK --MOSTRAR LOS DATOS EN FORMATO STRING

         --envio del dato 1
         if mostrar_dato1==1 then

            print_byte_dec(usb_serial_data,dato1)
            PRINT_CRLF(usb_serial_data)

            print_byte_dec(serial_hw_data,dato1)
            PRINT_CRLF(serial_hw_data)

         end if
```

```
   --envio del dato 2
   if mostrar_dato2==1 then

      print_word_dec(usb_serial_data,dato2)
      PRINT_CRLF(usb_serial_data)

      print_word_dec(serial_hw_data,dato2)
      PRINT_CRLF(serial_hw_data)

   end if

   --envio del dato 3
   if mostrar_dato3==1 then
      var word ind=0

      for numero_medidas using ind loop
         var word AuxByteWord=dato_array[ind]

         print_word_dec(usb_serial_data,AuxByteWord)
         PRINT_CRLF(usb_serial_data)

         print_word_dec(serial_hw_data,AuxByteWord)
         PRINT_CRLF(serial_hw_data)

      end loop
   END IF
   END BLOCK   --FIN BLOQUE MOSTRAR DATOS EN FORMATO STRING

--envio de datos en formato byte/word
else
   BLOCK      --MOSTRAR DATOS EN FORMATO BIG ENDIAN

   --envio del dato 1
   if mostrar_dato1==1 then
      usb_serial_data=dato1
      serial_hw_data=dato1
   end if
```

```
        --envio del dato 2
        if mostrar_dato2==1 then
          var byte AuxByte[2] at dato2
                                        usb_serial_data=AuxByte[1]
                                        serial_hw_data=AuxByte[1]
--                                              delay_10us(1)
                                        usb_serial_data=AuxByte[0]
                                        serial_hw_data=AuxByte[0]
        end if

        --envio del dato 3
        if mostrar_dato3==1 then
          var word ind=0
          for numero_medidas using ind loop
            var word AuxByteWord=dato_array[ind]
            var byte AuxByte[2] at AuxByteWord
                                        usb_serial_data=AuxByte[1]
            serial_hw_data=AuxByte[1]
                                        usb_serial_data=AuxByte[0]
                                        serial_hw_data=AuxByte[0]
          end loop
        END IF
        END BLOCK   --FIN BLOQUE MOSTRAR DATOS EN FORMATO BIG ENDIAN
      end if

    end if
    formato_buffer_intermedio=0
    habilitar_int_USB=ON      --la permito ahora qye ya no molesta
    return
    end procedure



-------------------------------------------------------------------------------------------------------------
--serial_USB se encarga de refrescar la comunicacion USB y de gestionar los mensajes a nivel global. Está ligada a la
--interrupcion asociada al desbordamiento del timer_USB, de nombre interrupcion_USB().
-------------------------------------------------------------------------------------------------------------
```

```
procedure serial_usb(BYTE out buffer_salida_datos) is
      if !has_shown_welcome_msg then
        --envío de cadena de bienvenida por usb y RS-232
        has_shown_welcome_msg = true
        print_string( usb_serial_data, str_welcome )
                              print_crlf(usb_serial_data)
        print_string( serial_hw_data, str_welcome )
                              print_crlf(serial_hw_data)
      ELSE
        --lectura y escritura
        lectura_SERIAL_usb(buffer_salida_datos)-- lectura desde el puerto USB-CDC y RS-232
        escritura_serial_usb ()              -- escritura por el puerto USB-CDC y RS-232
      end if
   RETURN
end procedure


-------------------------------------------------------------------------------------------------------------
--interrupcion_usb es un procedimiento asociado a la comunicacion USB, realizando las funciones de
--refresco de la señal USB y, si el flag de permiso de escritura-lectura está activo, recepción y envío de bytes.
--desde/por el interfaz USB
PROCEDURE INTERRUPCION_USB() is
   pragma inline
   valor_timer_usb=10      --asignación de la frecuencia de llamada de esta interrupción
              flag_int_USB=off       --??
              USB_SERIAL_FLUSH()        --refresco de la comunicacion USB
   IF (permiso_RX_TX) then  --si el flag de lectura- escritura está activo,
                serial_usb(ESTADO)    --permitimos la lectura-escritura cada vez que esta interrupción es activada.
   end if
              RETURN
end procedure


 --gestor de interrupciones asociado al USB
PROCEDURE sonar_isr_interrupcion_USB() is
              pragma interrupt
   IF (flag_int_USB)     THEN
     INTERRUPCION_USB()
```

```
     return
   END IF
end procedure


;PROCEDURE main_USB() is
;--   valor_timer_usb=10
;--            flag_int_USB=off
;             USB_SERIAL_FLUSH()                    --refresco de la comunicacion USB
;   IF (permiso_RX_TX) then
;               serial_usb(ESTADO)
;   end if
;             RETURN
;end procedure

;
;function configurar_conexion_inicio() is
;
;while(!usb_is_configured()) loop
;   selector_interfaz_usb=true
;   pinprueba2=1
;   delay_1ms(2)
;   pinprueba2=0
;   delay_1ms(12)
;   var byte sennal_recibida_puerto_serie=0
;   if (sennal_recibida_puerto_serie==sennal_activacion_puerto_serie) then
;      selector_interfaz_serie=true
;      exit loop
;   end if
;END loop
;
;
;--caso 1: se ha establecido comunicacion con rs-232 pero no con USB: solo se activa la interfaz 232
;if (selector_interfaz_serie & !usb_is_configured()) then
;   lcd_cursor_blink_display(0, 0, 1)
;   escribir_lcd(str_lcd_conf_serie_2)
;   selector_interfaz_serie =true
;   selector_interfaz_usb   =false
```

```
;
;--caso 2: no se ha establecido comunicacion con rs-232 pero sí con usb; solo activamos la interfaz usb
;elsif(!selector_interfaz_serie & usb_is_configured)then
;    lcd_cursor_blink_display(0, 0, 1)
;    escribir_lcd(str_lcd_conf_usb_2)
;    selector_interfaz_serie =false
;    selector_interfaz_usb   =true
;
;--caso 3: se han establecido ambas comunicaciones; activamos ambas interfaces
;elsif(selector_interfaz_serie & usb_is_configured) then
;    lcd_cursor_blink_display(0, 0, 1)
;    escribir_lcd(str_lcd_conf_mixto_2)
;    selector_interfaz_serie =true
;    selector_interfaz_usb   =true
;end if
;
;



-------------------------------------------------------------------------------------------------------------
--configurar_conexion_inicio se encarga de configurar por defecto la conexión usb mientras espera a que se reciba
--una señal de activación desde el puerto serie, se termine de configurar el USB o el interruptor de la interfaz
-- manual se active. Esta funcion por tanto provoca un bucle infinito hasta que al menos uno de los interfaces
-- existentes esté en condiciones de funcionar
-------------------------------------------------------------------------------------------------------------
 procedure configurar_conexion_inicio() is
   while(!usb_is_configured()) loop
      selector_interfaz_usb=false
      pinprueba1=1
      delay_1ms(2)
      pinprueba1=0
      delay_1ms(12)

      --esperamos a que se configure via RS-232
      var byte sennal_recibida_puerto_serie=0
      --serial_hw_write(sennal_recibida_puerto_serie)
      if (serial_hw_read(sennal_recibida_puerto_serie)==true &
        sennal_recibida_puerto_serie==sennal_activacion_puerto_serie) then
```

```
        selector_interfaz_serie=true
        exit loop
      end if

      --o via manual
      if pin_cambio_modo_manual == true then
      exit loop
      end if
   END loop


--caso 0: no se ha establecido comunicacion con ningún interfaz; pasa a modo manual
   if (pin_cambio_modo_manual == true) then
      lcd_cursor_blink_display(0, 0, 1)
      escribir_lcd(str_lcd_conf_manual)
      selector_interfaz_serie =false
      selector_interfaz_usb   =false
      return

--caso 1: se ha establecido comunicacion con rs-232 pero no con USB: solo se activa la interfaz 232
   elsif (selector_interfaz_serie & !usb_is_configured()) then
      lcd_cursor_blink_display(0, 0, 1)
      escribir_lcd(str_lcd_conf_serie_2)
      selector_interfaz_serie =true
      selector_interfaz_usb   =false
      return

--caso 2: no se ha establecido comunicacion con rs-232 pero sí con usb; solo activamos la interfaz usb
   elsif(!selector_interfaz_serie & usb_is_configured)then
      lcd_cursor_blink_display(0, 0, 1)
      escribir_lcd(str_lcd_conf_usb_2)
      selector_interfaz_serie =false
      selector_interfaz_usb   =true
      return

--caso 3: se han establecido ambas comunicaciones; activamos ambas interfaces
   elsif(selector_interfaz_serie & usb_is_configured) then
      lcd_cursor_blink_display(0, 0, 1)
```

```
      escribir_lcd(str_lcd_conf_mixto_2)
      selector_interfaz_serie =true
      selector_interfaz_usb   =true
      return
   end if

   return
end procedure


-------------------------------------------------------------------------------------------------------------
--reconfigurar_conexiones_hardware se encarga de comprobar el estado de de las interfaces de comunicación,
--procediendo a modificar la interfaz de comunicación activa las en función de su estado actual y pasado.
--La interfaz manual tiene preferencia sobre el resto, desactivando las interfaces USB y Rs-232 en cuanto se activa el
--interruptor asociado a la interfaz manual. Cuando el interruptor asociado a la interfaz manual está desactivado, se
--encargará de añadir los interfaces RS-232 o USB si anteriormente estaban desactivados.
-------------------------------------------------------------------------------------------------------------
procedure reconfigurar_conexiones_hardware() is
pragma inline

   var byte sennal_recibida_puerto_serie=0

--comprobamos la conectividad actual de la interfaz manual. Si el pin asociado al interruptor manual se activa,
--cerramos el resto de interfaces y activamos solo la interfaz manual.
   if (pin_cambio_modo_manual==true) then
     if (selector_interfaz_serie==true|selector_interfaz_usb==true) then
        lcd_cursor_blink_display(0, 0, 1)
        escribir_lcd(str_lcd_conf_manual)
        selector_interfaz_serie=false
        selector_interfaz_usb=false
     end if
     return
   end if

   --comprobamos la conectividad actual del puerto serie. Si se recibe señal de activación desde el puerto serie,
   --añadimos la interfaz puerto serie a la lista de interfaces activas
   if (!selector_interfaz_serie & (serial_hw_read(sennal_recibida_puerto_serie)&
     sennal_recibida_puerto_serie==sennal_activacion_puerto_serie))    then
```

```
      selector_interfaz_serie=true
      if(selector_interfaz_usb) then
         lcd_cursor_blink_display(0, 0, 1)
         escribir_lcd(str_lcd_conf_mixto_2)
      else
         escribir_lcd(str_lcd_conf_serie_2)
      end if
      return
   end if

   --comprobamos la conectividad actual del puerto USB. Si el periférico USB está correctamente configurado,
   --añadimos la interfaz USB a la lista de interfaces activas.
   if (!selector_interfaz_usb & usb_is_configured) then
      selector_interfaz_usb=true
      if(selector_interfaz_serie) then
         lcd_cursor_blink_display(0, 0, 1)
         escribir_lcd(str_lcd_conf_mixto_2)
      else
         escribir_lcd(str_lcd_conf_usb_2)
      end if
      return
   end if

end procedure



-------------------------------------------------------------------------------------------------------------
--menu_manual (byte in out estado) se encarga de gestionar las peticiones de medicion, lectura o cambio de modo de
--funcionamiento asociadas a los botoenes de la interfaz de uso manual; estos botones permitían leer la temperatura,
--realizar una medicion con filtro de mediana o conmutar entre el modo manual o el modo de funcionamiento
--autónomo por medio de los interfaces USB,RS-232
-------------------------------------------------------------------------------------------------------------
procedure menu_manual (byte IN out estado)   is
   pragma inline
   if pin_cambio_modo_manual == true then
      if pin_hacer_medicion == true then
         estado = MEDICION_DISTANCIA_MODO_MANUAL
         delay_1ms(50)
```

```
      elsif pin_leer_temperatura == true then
         estado = MEDICION_TEMPERATURA_MODO_MANUAL
         delay_1ms(50)
      end if
   end IF

   return
end procedure
```

6.1.5. fichero configuracion_LCD.jal

```
-- LCD IO definition
;alias lcd_rs                    is pin_D2          -- LCD command/data select.
;alias lcd_rs_direction          is pin_D2_direction
;alias lcd_en                    is pin_D3          -- LCD data trigger
;alias lcd_en_direction          is pin_D3_direction
;alias lcd_dataport              is portb_high      -- LCD data  port
;alias lcd_dataport_direction    is portb_high_direction


var   bit  lcd_d4                is  pin_A5     -- databit d4 pin
var   bit  lcd_d5                is  pin_A4     -- databit d5 pin
var   bit  lcd_d6                is  pin_A3     -- databit d6 pin
var   bit  lcd_d7                is  pin_A2     -- databit d7 pin

var   bit  lcd_d4_direction      is  pin_A5_direction     -- databit d4 pin
var   bit  lcd_d5_direction      is  pin_A4_direction     -- databit d5 pin
var   bit  lcd_d6_direction      is  pin_A3_direction     -- databit d6 pin
var   bit  lcd_d7_direction      is  pin_A2_direction     -- databit d7 pin

alias lcd_rs                     is pin_B5          -- LCD command/data select.
alias lcd_rs_direction           is pin_B5_direction
alias lcd_en                     is pin_B4          -- LCD data trigger
alias lcd_en_direction           is pin_B4_direction

const byte LCD_ROWS              = 2            -- 2 lines
const byte LCD_CHARS             = 8            -- 8 chars per line
```

```
const byte str_lcd_inicio[]              = "PFCSONAR 18f4550"
const byte str_lcd_conf_usb_1[]          ="Conectando..."
const byte str_lcd_conf_usb_2[]          ="Conexion USB OK "
const byte str_lcd_conf_serie_2[]         ="Conexion 232 OK "
const byte str_lcd_conf_mixto_2[]        ="Conexion USB+232"
const byte str_lcd_conf_manual[]         ="Modo manual"
const byte str_medicion_simple_1[]       ="Medida:"
const byte str_medicion_batch_1[]        ="Midiendo..."
const byte str_medicion_batch_2[]        ="Hecho"

const byte str_lectura_vel_sonido[]      ="Vel(m/s)="
const byte str_lectura_num_pulsos[]      ="Num_pulsos="
const byte str_lectura_lapso_minimo[]    ="Lapso_min="
const byte str_lectura_retardo[]         ="Retardo="
const byte str_lectura_temperatura[]     ="Temp(oC)="
const byte str_lectura_deriva_temperatura[] ="Error(oC)="
const byte str_lectura_humedad_relativa[]  ="H.Rel(%)="
const byte str_lectura_numero_medidas[]  ="Num. medidas="

--lcd_rs_direction                  = output
--lcd_en_direction                   = output
--lcd_dataport_direction               = output


lcd_rs_direction                   = output
lcd_en_direction                    = output

lcd_d7_direction                    =output
lcd_d6_direction                    =output
lcd_d5_direction                    =output
lcd_d4_direction                    =output


 include lcd_hd44780_4             -- libreria LCD para puerto de 4 pines

procedure escribir_lcd (byte in str[]) is
--pragma inline
lcd_cursor_position(0,0)
```

```
var volatile byte aux=0
for count(str) using aux loop
   lcd=str[aux]
   if (aux%LCD_CHARS==7) then
      lcd_cursor_position(1,0)
   end if

end loop


end procedure




--inicializa el dispositivo
LCD_INIT()

escribir_lcd(str_lcd_inicio)

;var byte aux=0
;for count(str_lcd_inicio) using aux loop
;   if aux==8 then
;      lcd_cursor_position(1,0)
;   end if
;   lcd=str_lcd_inicio[aux]
;end loop


--test
;var byte aux=0
;;lcd_cursor_position(1,0)
;for 256 loop
;delay_1s(1)
;lcd_cursor_position(0,0)
;print_byte_dec(lcd,aux)
;lcd_cursor_position(1,0)
;print_byte_hex(lcd,aux)
```

```
;aux=aux+1
;end loop
```

6.1.6 fichero menu_configuracion_lectura.jal

```
------------------------------------------------------------------------------------------------------------
--
--Libreria encargada de la lectura de las peticiones de de lectura-edicion de los parametros del dispositivo,
-- entre lo que se incluyen la temperatura (ºC), el numero de pulsos a enviar, el retardo (en mm), el lapso minimo
-- entre emision y recepcion (en unidades de  10us), el modo de calibracion (por temperatura o manual) y la velocidad
-- del sonido (en cm/s). Cada vez que se modifique cualquiera de estos parámetros, serán guardados en la eeprom del
-- microcontrolador para garantizar la permanencia de los mismos.
--Así mismo, se encarga de mostrar por el LCD los parámetros que serán modificados y leídos.
--
--
--
------------------------------------------------------------------------------------------------------------
--CONSTANTES ASOCIADAS AL SUBMENU DE CONFIGURACION
const byte CONFIGURACION                ="S"
const byte LECTURA                      ="R"
const byte CONFIGURACION_LAPSO_MINIMO   ="L"
const byte CONFIGURACION_RETARDO        ="D"
const byte CONFIGURACION_NUMERO_PULSOS  ="N"
const byte configuracion_deriva_temperatura ="Z"
const byte configuracion_efecto_humedad  ="H"
const byte configuracion_numero_medidas  ="K"

--CONSTANTES ASOCIADAS AL SUBMENU DE LECTURA
const byte LECTURA_LAPSO_MINIMO         ="L"
const byte LECTURA_VELOCIDAD_SONIDO     ="V"
const byte LECTURA_RETARDO              ="D"
const byte LECTURA_NUMERO_PULSOS        ="N"
const byte LECTURA_TEMPERATURA          ="T"
const byte LECTURA_DERIVA_TEMPERATURA   ="Z"
const byte LECTURA_EFECTO_HUMEDAD       ="H"
const byte LECTURA_NUMERO_MEDIDAS       ="K"

------------------------------------------------------------------------------------------------------------
--
```

```
--funcion que activa o desactiva los pines asociados a los leds, para testeo
--
-------------------------------------------------------------------------------------------------------------------
procedure activa_led (bit in tipo) is
   pinprueba1=0
   pinprueba2=0

   if tipo then
    for 3 loop
        pinprueba1=1
        pinprueba2=0
        delay_1ms(100)
        pinprueba1=1
        pinprueba2=0
        delay_1ms(100)
    end loop
       pinprueba1=0
   else
       for 3 loop
          pinprueba1=1
          pinprueba2=1
          delay_1ms(100)
          pinprueba1=0
          pinprueba2=0
          delay_1ms(100)
      end loop
end if

return

end procedure

-------------------------------------------------------------------------------------------------------------------
--
--submenu configuracion implmenta la modificacion de los parametros del dispositivo
--(temperatura, velocidad del sonido, numero de pulsos, retardo, lapso minimo)
--Se encargará además de mostrar por el LCD los parámetros modificados, y de guardar en la eeprom dichos parámetros.
```

```
--
-----------------------------------------------------------------------------------------------------------------
function submenu_configuracion ()  return bit is
   permiso_RX_TX=off    --impido que se lea o escriba por el puerto serie-USB

   --variables donde se guardará temporalmente los datos que se reciban via USB
   var byte parametro_configuracion2 = 0
   var byte parametro_configuracion3 = 0
   var byte parametro_configuracion=0

   --contadores asociados al tiempo máximo de espera para recibir un dato nuevo; el tiempo viene determinado por
   --la ecuacion NMAX*(TMAX*10us); por defecto, 10*50*10us->  5ms
   var byte aux1=0
   CONST BYTE TMAX=10
   CONST BYTE NMAX=50

   --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
   while(!lectura_serial_usb(parametro_configuracion)&aux1<NMAX) loop
      delay_10us(TMAX)
      aux1=aux1+1
   end loop

   --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
   if aux1==NMAX then
     permiso_RX_TX=on
     return off
   end if

   aux1=0


   --CONFIGURACION LAPSO MINIMO
   if parametro_configuracion==configuracion_lapso_minimo then
      aux1=0

    --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
      while (!lectura_serial_usb(parametro_configuracion2)&aux1<NMAX) loop
         delay_10us(TMAX)
```

```
      aux1=aux1+1
    end loop

  --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
    if aux1==NMAX then
      permiso_RX_TX=on
      return off
    end if

    permiso_RX_TX=on     --permitiendo nuevamente la lectura-escritura en modo "síncrono"

    --determina si se cambia al valor por defecto o por el contrario el que se incluye por USB
    if parametro_configuracion2>=1 then
      lapso_minimo=parametro_configuracion2
    else
      lapso_minimo=lapso_minimo_defecto
    end if

    --guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
    data_eeprom_write(offset_eeprom_lapso_minimo,lapso_minimo)



     --mostramos los nuevos valores en el LCD
    escribir_lcd(str_lectura_lapso_minimo)
    print_byte_dec(lcd,lapso_minimo)
    lcd="0"
    lcd="u"
    lcd="s"
    lcd=" " lcd=" "
    RETURN  on

;   --CONFIGURACION NUMERO DE MEDIDAS
   elsif parametro_configuracion==configuracion_numero_medidas then
    aux1=0

;     lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
    while (!lectura_serial_usb(parametro_configuracion2)&aux1<NMAX) loop
      delay_10us(TMAX)
```

```
      aux1=aux1+1
    end loop

;   si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
    if aux1==NMAX then
      permiso_RX_TX=on
      return off
    end if

;     permiso_RX_TX=on     permitiendo nuevamente la lectura-escritura en modo "síncrono"

;     determina si se cambia al valor por defecto o por el contrario el que se incluye por USB
    if parametro_configuracion2>0 & parametro_configuracion2<=NUMERO_MEDIDAS_MAXIMO then
      numero_medidas=parametro_configuracion2
    else
      numero_medidas=numero_medidas_maximo
    end if

;     guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
    data_eeprom_write(offset_eeprom_numero_medidas,numero_medidas)


;     mostramos los nuevos valores en el LCD
    escribir_lcd(str_lectura_numero_medidas)
    print_byte_dec(lcd,numero_medidas)
    lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
    RETURN  on


  --BLOQUE CONFIGURACION RETARD
  elsif parametro_configuracion==configuracion_retardo then

    -lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
    aux1=0
    while (!lectura_serial_usb(parametro_configuracion2)&aux1<NMAX) loop
      delay_10us(TMAX)
      aux1=aux1+1
    end loop
```

```
 --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
  if aux1==NMAX then
    permiso_RX_TX=on
    return off
  end if

 --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
  aux1=0
  while (!lectura_serial_usb(parametro_configuracion3)&aux1<NMAX) loop
     delay_10us(TMAX)
     aux1=aux1+1
  end loop

--si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
  if aux1==NMAX then
    permiso_RX_TX=on
    return off
  end if

  permiso_RX_TX=on     --permitiendo nuevamente la lectura-escritura en modo "síncrono"


   --determina si se cambia al valor por defecto o por el contrario el que se incluye por USB
  if parametro_configuracion2==0 & parametro_configuracion3==0 then
     retardo=retardo_defecto
  else
     retardo=word(parametro_configuracion2)<<8
     retardo=retardo+word(parametro_configuracion3)
  end if

  --guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
  data_eeprom_write_word(offset_eeprom_retardo,retardo)

  --mostramos los nuevos valores en el LCD
  escribir_lcd(str_lectura_retardo)
  print_word_dec(lcd,retardo)
  lcd="m"
```

```
      lcd="m"
      lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
      RETURN  on


--CONFIGURACION NUMERO DE PULSOS
   elsif parametro_configuracion==configuracion_numero_pulsos then

        --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
      aux1=0
      while (!lectura_serial_usb(parametro_configuracion2)&aux1<NMAX) loop
         delay_10us(TMAX)
         aux1=aux1+1
      end loop

          --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
      if aux1==NMAX then
        permiso_RX_TX=on
        return off
      end if

      permiso_RX_TX=on     --permitimos nuevamente la lectura-escritura en modo "síncrono"


      --determina si se cambia al valor por defecto o por el contrario el que se incluye por USB
      if parametro_configuracion2>=1 then
         numero_pulsos=parametro_configuracion2*2
      else
         numero_pulsos=numero_pulsos_defecto
      end if

      --guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
      data_eeprom_write(offset_eeprom_numero_pulsos,numero_pulsos)

      --mostramos los nuevos valores en el LCD
      escribir_lcd(str_lectura_num_pulsos)
```

```
      print_byte_dec(lcd,numero_pulsos/2)
      lcd=" " lcd=" " lcd=" " lcd=" "
      RETURN  on


--CONFIGURACION DERIVA TEMPERATURA
   elsif parametro_configuracion==configuracion_deriva_temperatura then

     --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
      aux1=0
      while (!lectura_serial_usb(parametro_configuracion2)&aux1<NMAX) loop
         delay_10us(TMAX)
         aux1=aux1+1
      end loop

         --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de
configuracion
      if aux1==NMAX then
        permiso_RX_TX=on
        return off
      end if

        --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el
buffer
      aux1=0
      while (!lectura_serial_usb(parametro_configuracion3)&aux1<NMAX) loop
         delay_10us(TMAX)
         aux1=aux1+1
      end loop

         --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de
configuracion
      if aux1==NMAX then
        permiso_RX_TX=on
        return off
      end if

      permiso_RX_TX=on     --permitiendo nuevamente la lectura-escritura en modo "síncrono"
```

```
var word aux_temp=word(parametro_configuracion2)*256+word(parametro_configuracion3)
deriva_temperatura=aux_temp

--evitamos que la corrección sea superior a los 25º por encima o por debajo de la temperatura del sensor
if deriva_temperatura<-2500 | deriva_temperatura >2500 then
   deriva_temperatura=0
end if


--guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
data_eeprom_write(offset_eeprom_deriva_temperatura,deriva_temperatura)

--mostramos los nuevos valores en el LCD
escribir_lcd(str_lectura_deriva_temperatura)
aux_temp=deriva_temperatura
if (deriva_temperatura<0) then
  lcd="-"
  aux_temp=-deriva_temperatura
end if

print_word_dec(lcd,aux_temp/100)
lcd="."
if (aux_temp%100<10) then
   lcd="0"
end if
print_word_dec(lcd,aux_temp%100)


lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
RETURN  on



--CONFIGURACION EFECTO DE LA HUMEDAD RELATIVA
  elsif parametro_configuracion==configuracion_efecto_humedad then
```

```
    --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
    aux1=0
    while (!lectura_serial_usb(parametro_configuracion2)&aux1<NMAX) loop
      delay_10us(TMAX)
      aux1=aux1+1
    end loop

      --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
    if aux1==NMAX then
      permiso_RX_TX=on
      return off
    end if

    permiso_RX_TX=on    --permitimos nuevamente la lectura-escritura en modo "síncrono"

    --actualizamos la variable
    humedad_relativa=parametro_configuracion2

    --guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
    data_eeprom_write(offset_eeprom_humedad_relativa,humedad_relativa)


    --mostramos los nuevos valores en el LCD
    escribir_lcd(str_lectura_humedad_relativa)
    print_word_dec(lcd,(word(humedad_relativa)*100)/255)
    lcd="."
    if  (((word(humedad_relativa)*100)%255)<10) then
      lcd="0"
    end if
    print_word_dec(lcd,(word(humedad_relativa)*100)%255)
    lcd=" " lcd=" " lcd=" " lcd=" "
    RETURN  on
```

```
   end if

   permiso_RX_TX=on     --permitiendo nuevamente la lectura-escritura en modo "síncrono"
   return off
end FUNCTION



----------------------------------------------------------------------------------------------------------------
--
--submenu lectura implmenta el interfaz USB encargado de leer los parametros del dispositivo
--(temperatura, velocidad del sonido, numero de pulsos, retardo, lapso minimo)
--Se encargará además de mostrar por el LCD los parámetros solicitados.
--
----------------------------------------------------------------------------------------------------------------
function submenu_lectura ()  return bit is
   permiso_RX_TX=off    --impido que se lea o escriba por el puerto serie-USB

   --variables donde se guardará temporalmente los datos que se reciban via USB
   var byte parametro_lectura = 0

   --constantes y variables asociados al tiempo máximo de espera para recibir un dato nuevo; el tiempo viene
   --determinado por la ecuacion NMAX*(TMAX*10us); por defecto, 10*50*10us->  5ms
   var byte aux1=0
   CONST BYTE TMAX=10
   CONST BYTE NMAX=50

   --lee del puerto serie-usb con un timeout máximo de TMAX*NMAX; saldrá antes si ha leido un caracter en el buffer
   while(!lectura_serial_usb(parametro_lectura)&aux1<NMAX) loop
      delay_10us(TMAX)
      aux1=aux1+1
   end loop
   aux1=0

   --si ha transcurrido el tiempo de espera máximo y no ha recibido ningún carácter, se sale del menu de configuracion
   if aux1==NMAX then
     permiso_RX_TX=on
     return off
```

```
end if

permiso_RX_TX=on     --permitiendo nuevamente la lectura-escritura en modo "síncrono"

--escribimos en el buffer intermedio el caracter R como identificador de parámetro de lectura
id1="R"


--BLOQUE LECTURA LAPSO MINIMO   (byte)
if parametro_lectura==lectura_lapso_minimo then

  --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
   id2=lectura_lapso_minimo
   dato1=lapso_minimo
   formato_buffer_intermedio=0b00_001_0_11
   flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
   asm nop
   formato_buffer_intermedio=0

   --mostramos por el lcd los datos solicitados
   escribir_lcd(str_lectura_lapso_minimo)
   print_byte_dec(lcd,dato1)
   lcd="0"
   lcd="u"
   lcd="s"
   lcd=" " lcd=" "
   RETURN  on


--BLOQUE LECTURA VELOCIDAD SONIDO (word)   (m/s)
elsif parametro_lectura==lectura_velocidad_sonido then

  --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
   id2=lectura_velocidad_sonido
   vel_sonido=obtener_vel_sonido_cent(canal_ADC_sensor_temperatura,deriva_temperatura,humedad_relativa)
   dato2=vel_sonido
   formato_buffer_intermedio=0b00_010_0_11
   flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
```

```
   asm nop
   formato_buffer_intermedio=0


   --guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
   data_eeprom_write(offset_eeprom_vel_sonido,vel_sonido)


   --mostramos por el lcd los datos solicitados
   escribir_lcd(str_lectura_vel_sonido)

   print_word_dec(lcd,dato2/100)
   lcd="."
   if (dato2%100<10) then
      lcd="0"
   end if
   print_byte_dec(lcd,dato2%100)
   --print_word_dec(lcd,dato2)
   lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
   RETURN  on


--BLOQUE LECTURA RETARDO         (word)  (mm)
elsif parametro_lectura==lectura_retardo then

  --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
   id2=lectura_retardo
   dato2=retardo
   formato_buffer_intermedio=0b00_010_0_11
   flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
   asm nop
   formato_buffer_intermedio=0

   --mostramos por el lcd los datos solicitados
   escribir_lcd(str_lectura_retardo)
   print_word_dec(lcd,dato2)
   lcd="m"
   lcd="m"
```

```
    lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
    RETURN  on


--BLOQUE LECTURA NUMERO DE PULSOS (byte)
elsif parametro_lectura==lectura_numero_pulsos then

   --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
   id2=lectura_numero_pulsos
    dato1=numero_pulsos/2
    formato_buffer_intermedio=0b00_001_0_11
    flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
    asm nop
    formato_buffer_intermedio=0

    --mostramos por el lcd los datos solicitados
    escribir_lcd(str_lectura_num_pulsos)
    print_byte_dec(lcd,numero_pulsos/2)
    lcd=" " lcd=" " lcd=" " lcd=" "
    RETURN  on


--BLOQUE LECTURA Temperatura (word)
--mostrará por el interfaz escogido la temperatura ambiente, teniendo en cuenta el factor de corrección
--debido a errores de deriva; Tª real = Tª sensor + deriva_temperatura.
elsif parametro_lectura==lectura_temperatura then

   --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
    id2=lectura_temperatura
    asm nop
    var sword temperatura=obtener_temperatura_cent(canal_ADC_sensor_temperatura,deriva_temperatura)
    dato2=temperatura--  + deriva_temperatura
    formato_buffer_intermedio=0b00_010_0_11
    flag_int_USB=on

    --guardamos la variable en la eeprom, para que permanezcan los parámetros después de apagar el dispositivo
    data_eeprom_write(offset_eeprom_temperatura,temperatura)
```

```
   --mostramos por lcd los datos solicitados
   lcd_clear_screen()
   var sword aux_temp=temperatura--+deriva_temperatura
   if (aux_temp<0) then
     lcd="-"
     aux_temp=-aux_temp
   end if

   escribir_lcd(str_lectura_temperatura)
   print_word_dec(lcd,aux_temp/100)
   lcd="."
   if (aux_temp%100<10) then
      lcd="0"
   end if
   print_word_dec(lcd,aux_temp%100)
   lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
   RETURN  on

     --BLOQUE LECTURA DERIVA TEMPERATURA (WORD)
elsif parametro_lectura==lectura_deriva_temperatura then

  --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
   id2=lectura_deriva_temperatura
   dato2=deriva_temperatura
   formato_buffer_intermedio=0b00_010_0_11
   flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
   asm nop
   formato_buffer_intermedio=0

   --mostramos por el lcd los datos solicitados
   escribir_lcd(str_lectura_deriva_temperatura)
   var sword aux_temp=deriva_temperatura
   if (deriva_temperatura<0) then
     lcd="-"
     aux_temp=-deriva_temperatura
   end if
```

```
    print_word_dec(lcd,aux_temp/100)
    lcd="."
    if aux_temp%100<10 then
       lcd="0"
    end if
    print_word_dec(lcd,aux_temp%100)


    lcd=" " lcd=" " lcd=" " lcd=" " lcd=" "
    RETURN  on

       --BLOQUE LECTURA EFECTO HUMEDAD RELATIVA (BYTE)
elsif parametro_lectura==lectura_efecto_humedad then

  --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
    id2=lectura_efecto_humedad
    dato1=humedad_relativa
    formato_buffer_intermedio=0b00_001_0_11
    flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
    asm nop
    formato_buffer_intermedio=0

    --mostramos por el lcd los datos solicitados
    escribir_lcd(str_lectura_humedad_relativa)
    print_word_dec(lcd,(word(humedad_relativa)*100)/255)
    lcd="."
    if  (((word(humedad_relativa)*100)%255)<10) then
       lcd="0"
    end if
    print_word_dec(lcd,(word(humedad_relativa)*100)%255)
    lcd=" " lcd=" " lcd=" " lcd=" "

    RETURN  on


--BLOQUE LECTURA NUMERO MEDICIONES (BYTE)
elsif parametro_lectura==lectura_numero_medidas then
```

```
      --escribimos el resto de identificadores y variables en el buffer intermedio, y configuramos el buffer
       id2=lectura_numero_medidas
       dato1=numero_medidas
       formato_buffer_intermedio=0b00_001_0_11
       flag_int_USB=on    --forzamos el envio por USB de los datos del buffer intermedio
       asm nop
       formato_buffer_intermedio=0

       --mostramos por el lcd los datos solicitados
       escribir_lcd(str_lectura_numero_medidas)
       print_byte_dec(lcd,numero_medidas)
       lcd=" " lcd=" " lcd=" " lcd=" "

       RETURN  on


    end if




    permiso_RX_TX=on     --permitiendo nuevamente la lectura-escritura en modo "síncrono"
    return off
end function
```

6.1.7 fichero menu_configuracion_lectura.jal

```
------------------------------------------------------------------------------------------------------------------
--
--Libreria encargada del proceso de obtencion de distancia a partir del tiempo de vuelo de la señal ultrasonica.
--Incluye las diferentes estrategias de medicion de distancia (unica, lote de medidas, media de un lote de medidas),
--así como el interfaz USB encargado de la selección del tipo de medicion y del envio de datos por USB.
--Ademas se encarga de configurar los timeres y int. asociadas a la recepcion y envio de la señal ultrasonica
--
------------------------------------------------------------------------------------------------------------------

--CONSTANTES ASOCIADAS AL INTERFAZ USB-SERIAL ENCARGADO DE SELECCION DEL TIPO DE MEDICION
const byte MEDICION_SIMPLE              ="M"
const byte MEDICION_SIMPLE_WORD         ="m"
const byte MEDICION_MULTIPLE_BATCH      ="B"
const byte MEDICION_MULTIPLE_BATCH_WORD ="b"
const byte MEDICION_MULTIPLE_media      ="N"
const byte MEDICION_MULTIPLE_media_WORD ="n"


--VALORES "DE FABRICA" DE LOS PARAMETROS QUE DEFINEN EL PROCESO DE MEDICION
const byte lapso_minimo_defecto         =40   --tiempo de espera entre emisión a recepcion en tramos de 10 us
const dword  ciclos_decima_segundo      =300000 --12MOPS /preescaler[8] *2 /10
const word retardo_defecto              =25   --retardo en la medicion en mm
const word vel_sonido_defecto           =34000  --medida en centimetros/segundo
const byte numero_medidas_defecto       =NUMERO_MEDIDAS_MAXIMO --numero de mediciones por defecto
const sword temperatura_defecto         =3000   --medida en centésimas de ºC
const byte deriva_temperatura_defecto   =0    --correccion de la deriva de temperatura
const byte humedad_relativa_defecto     =0    --correccion de la velocidad debido a la humedad relativa
--VARIABLES ASOCIADAS A LOS PARAMETROS QUE DEFINEN EL PROCESO DE MEDICION
var volatile byte lapso_minimo
var volatile word vel_sonido
var volatile word retardo
var volatile sword temperatura
var volatile sword deriva_temperatura
var volatile byte   humedad_relativa
```

```
--
--
--
-----------------------------------------------------------------------


--LIBRERIAS ENCARGADAS DE LOS PROCESO DE EMISION Y RECEPCION DE LA SEÑAL ULTRASONICA
include configuracion_emision_ultrasonidos      --pines y constantes asociados a la emision de u.sonidos
include configuracion_recepcion_ultrasonidos    --pines y constantes asociados a la recepcion de u.sonidos

--LEEMOS LA EEPROM PARA OBTENER LOS VALORES ANTERIORMENTE SELECCIONADOS
--
--MODO DE DESARROLLO; NO USAMOS LA EEPROM PARA CARGAR LOS VALORES INICIALES,
--SINO QUE USAMOS VALORES POR DEFECTO
IF (defined(MODO_DESARROLLO)) THEN
   vel_sonido=vel_sonido_defecto
   retardo=retardo_defecto
   lapso_minimo=lapso_minimo_defecto
   numero_pulsos=numero_pulsos_defecto
   temperatura=temperatura_defecto
   deriva_temperatura=deriva_temperatura_defecto
   humedad_relativa=humedad_relativa_defecto
   numero_medidas=numero_medidas_defecto
ELSE

   data_eeprom_read_word(offset_eeprom_vel_sonido,vel_sonido)
   data_eeprom_read_word(offset_eeprom_retardo,retardo)
   data_eeprom_read(offset_eeprom_lapso_minimo,lapso_minimo)
   data_eeprom_read(offset_eeprom_numero_pulsos,numero_pulsos)
   data_eeprom_read(offset_eeprom_temperatura,temperatura)
   data_eeprom_read(offset_eeprom_deriva_temperatura,deriva_temperatura)
   data_eeprom_read(offset_eeprom_humedad_relativa,humedad_relativa)
   data_eeprom_read(offset_eeprom_numero_medidas,numero_medidas)
END IF


-------------------------------------------------------------------------------------------------------------
--
```

```
--ordena_burbuja recibe como parametros un vector "set" de tipo word de longitud definida por "n", y los ordena
--de modo ascendente. Realiza la ordenacion mediante el algoritmo burbuja,
--simple pero computacionalmente costoso.
--Se debe procurar evitar que n sea mayor que la longitud maxima del vector
--
-----------------------------------------------------------------------------------------------------------
procedure ordena_burbuja(byte in n, word in set[]) is
   var word inter=0
   var word ind2=0
   var word ind3=0

   --así evitamos posibles desbordamiento en el caso de que el indice fuera mayor que el tamaño del array
   if n<count(set) then
      n=count(set)
   end if

   --ordenamiento tipo burbuja
   var word indice_2=n - 1
   for indice_2 using ind2 loop
      for indice_2 using ind3 loop
         if set[ind3]>set[ind3+1] then
            inter=set[ind3]
            set[ind3]=set[ind3+1]
            set[ind3+1]=inter
         end if
      end loop
   end loop
end procedure




-----------------------------------------------------------------------------------------------------------
--
--obtencion_distancia: funcion que recibe como parametro el tiempo de vuelo en ciclos de timer
--y devuelve la distancia asociada en milimetros
--
-----------------------------------------------------------------------------------------------------------
function obtencion_distancia (WORD in lapso) return word is
```

```
   var dword aux
   aux=dword(lapso)
   --cálculo del tiempo de vuelo; distancia=vel_sonido*(tiempo_transcurrido/2)
            aux=(aux*dword(vel_sonido))/ciclos_decima_segundo
   aux=word(aux)
   return aux
end function



-------------------------------------------------------------------------------------------------------------------
--
-- calcular_TOF: funcion que realiza el cálculo del tiempo de vuelo; se encarga de configurar
-- la emision de ultrasonidos, esperar a que se reciba el eco y calcular el tiempo de vuelo resultante.
-- Se encarga así mismo de configurar y gestionar los timeres e interrupciones asociadas al envio y recepcion
-- de ultrasonidos.
-- Recibe como parámetro de entrada 'num_pulsos' una variable byte con el número de pulsos de ultrasonidos a emitir
--
-- Devuelve como parámetro de salida 't'; una variable word con el tiempo de vuelo medido en número de ciclos de reloj
-- del timer correspondiente.
-- el tiempo de vuelo resultante ()así mismo como
-- Posteriormente, a partir de este tiempo obtenido (en ciclos de procesador) se podrá determinar la distancia gracias
--a la funcion obtencion_distancia, explicada anteriormente.
--
-------------------------------------------------------------------------------------------------------------------
procedure calcular_TOF(WORD OUT t,byte in num_pulsos) is

   pinpwm=0                  --ponemos a 0 el pin pwm
   lapso_transcurrido=0          --ponemos a 0 la variable global que nos mide el tiempo de vuelo
   flag_int_RECEPTOR=OFF          --configuracion inicial rb1 (pulso de vuelta): flag a 0
   habilitar_int_RECEPTOR=off      --configuracion inicial rb1 (pulso de vuelta): deshabilitar interrupcion

   --configuracion timer_pwm e int. asociadas
            valor_timer_PWM=255-FREC_PWM_REAL   --determina la frecuencia de la señal pwm (40KHz)
            registro_timer_PWM=0B0_1001_000     --Se configura el timer_pwm con preescaler=2. Aun no se activa
            habilitar_int_PWM=ON            --Se habilita la interrupcion asociada al desbordamiento
```

```
   --configuracion timer_lapso maximo e int. asociadas
            tmr1=NUMERO_CICLOS_ESPERA          --determina el tiempo máximo de espera de eco.
            registro_timer_lapso_maximo=0b00_11_000_0  --Se configura con preescaler=8. Aun no se activa
            habilitar_int_lapso_maximo=off     --Se deshabilita la interrupcion asociada a su dedsordamiento


   contadorpwm=0              --puesta a 0 del contador asociado a la señal pwm
   activar_timer_USB=OFF      --deshabilitar momentaneamente la comunicacion USB
   activar_timer_lapso_maximo=on   --activacion del timer lapso maximo(que mide el tiempo de vuelo)
   activar_timer_PWM=on       --activacion del timer pwm (que genera la señal modulada)

 --proceso de envio de la señal; mientras el contador de ciclos sea menor que el numero de pulsos, continua emitiendo
            while (contadorpwm<num_pulsos) loop end loop
   activar_timer_PWM=off      --finalizar envio pulso pwm
     habilitar_int_lapso_maximo=ON    --configuracion timer1: activa interrupciones asociadas al tiempo de espera
maximo.
   delay_10us(lapso_minimo)       --lapso entre envio y recepcion, para evitar resonancia.
   flag_int_RECEPTOR=OFF          --borramos el flag de recepcion de eco para borrar cualquier ruido
   habilitar_int_RECEPTOR=on      --activa interrupcion asociada a la recepcion de eco.

   --proceso de medida: mientras la variable global medicionpwm se mantenga a 0, seguirá corriendo el temporizador.
   --la variable se pondrá a 1 solo cuando las interrupciones asociadas a la recepcion y al tiempo maximo de espera
   medicionpwm=0
            while(medicionpwm==0) loop
            end loop

   --se ha terminado el proceso de medida, sea porque se ha recibido eco o porque ha transcurrido un tiempo mayor
   --que el especificado. Se desactivan los contadores y de la interrupcion asociada al pin b1
    t=lapso_transcurrido          --determinacion del tiempo de vuelo
   habilitar_int_RECEPTOR=off     --desactivamos las interrupciones
   habilitar_int_lapso_maximo=OFF  --configuracion timer1: desactiva interrupciones
            activar_timer_lapso_maximo=off
   activar_timer_USB=ON           --rehabilitar la comunicacion USB
   return
end procedure
```

```
----------------------------------------------------------------------------------------------------------------
--
--menu_medicion simple se encarga de la obtencion de la distancia al objeto mediante una unica medicion.
--Determina la
--distancia al objeto calculando el tiempo de vuelo, calculando despues la distancia asociada, corrgiendo ademas el
--retardo.
--
----------------------------------------------------------------------------------------------------------------
procedure menu_medicion_simple(byte in num_pulsos, word out dist)  is
   var volatile word t=0

   calcular_TOF(t,num_pulsos)       --obtencion del tiempo de vuelo

   --actualización de la velocidad del sonido de acuerdo con el valor de temperatura
   vel_sonido=obtener_vel_sonido_cent(canal_ADC_sensor_temperatura,deriva_temperatura,humedad_relativa)

   dist=obtencion_distancia(t)        --obtencion de la distancia al objeto


   if dist>retardo then              --correcion del retardo

      dist=dist-retardo            --si se ha recibido eco añadimos retardo.

   else

      dist=0                    --si no hay eco, lo dejamos a 0

   end if

   return
end procedure



----------------------------------------------------------------------------------------------------------------
--
--menu_medicion multiple IMPLEMENTA EL PROCESO COMPLETO DE MEDICION DE UN SET DE MEDICIONES;
--DETERMINACION DE NUMERO
--DE MEDICIONES, OBTENCION DEL TIEMPO DE VUELO Y CALCULO DE LA DISTANCIA EN FUNCION DE LA VELOCIDAD
```

```
---DEL SONIDO
--LA FUNCION GUARDARA EN UN ARRAY TODAS LAS MEDICIONES
--
-------------------------------------------------------------------------------------------------------------
function menu_medicion_multiple_batch(byte in num_pulsos) return bit is
   var word dist=0
   var volatile word t
   var byte indice=0

--actualización de la velocidad del sonido de acuerdo con el valor de temperatura.
--lo hacemos una sola vez para no ralentizar demasiado el proceso de medicion
vel_sonido=obtener_vel_sonido_cent(canal_ADC_sensor_temperatura,deriva_temperatura,humedad_relativa)

--realizacion del bucle de mediciones
   for numero_medidas using indice loop
      t=0
--       if estado==abortar then
--          indice=num_medidas-1
--       end if
      calcular_TOF(t,num_pulsos)
      dist=obtencion_distancia(t)
      if dist>retardo then
         dist=dist-retardo --si se ha recibido eco añadimos retardo.
      else dist=0                  --si no hay eco, lo dejamos a 0
      end if
      set_medida[indice]=dist
      end loop
   return  on
end function




-------------------------------------------------------------------------------------------------------------
--
--menu medicion multiple REALIZA UN SET DE MEDICIONES, DETERMINANDO EL NUMERO DE MEDICIONES FALLIDAS Y
EL VALOR DE
--MEDIANA DE LAS MEDIDAS QUE HAN RESULTADO POSITIVAS.
--
```

```
-----------------------------------------------------------------------------------------------------------------
function menu_medicion_multiple_media(byte in num_pulsos,byte out numero_ceros, word out mediana) return bit
is
   --variables auxiliares: por defecto, se considera que todas las mediciones han fallado
   var word dist=0                    --almacena temporalmente la distancia
   var word t                         --almacena temporalmente el tiempo de vuelo
   var BYTE indice=0                  --indice para recorrer la matriz
   var BYTE primer_dato_valido=numero_medidas-1   --indice al primer elemento !=0; por defecto, el ultimo
   var BYTE numero_medio=0

--actualización de la velocidad del sonido de acuerdo con el valor de temperatura.
--lo hacemos una sola vez para no ralentizar demasiado el proceso de medicion
vel_sonido=obtener_vel_sonido_cent(canal_ADC_sensor_temperatura,deriva_temperatura,humedad_relativa)

   --bucle de realizacion de mediciones
   for numero_medidas using indice loop
      t=0
      calcular_TOF(t,num_pulsos)
      set_medida[indice]=t
   end loop

   ordena_burbuja(numero_medidas,set_medida)--ORDENA EL ARRAY EN MODO ASCENDENTE

   --obtencion de la mediana, descartando los valores nulos
   if (numero_medidas>1) then --si hay mas de una medida,y al menos una de ellas es valida...
      indice=0
      while (indice<numero_medidas & primer_dato_valido==0) loop
         if set_medida[indice]!=0 then
            primer_dato_valido=indice
         end if
         indice=indice+1
      end loop
   else
      primer_dato_valido=0
   end if
   numero_medio=primer_dato_valido + (numero_medidas - 1 - primer_dato_valido)/2

   --obtencion del numero de ceros
```

```
   indice=0
   numero_ceros=0
   for numero_medidas using indice loop
      if set_medida[indice]==0 then
         numero_ceros=indice+1
      end if
   end loop


    dist=obtencion_distancia(set_medida[numero_medio])--obtencion de la distancia del valor correspondiente a la
mediana
   if (dist>retardo) then
      mediana=dist-retardo --si se ha recibido eco añadimos retardo.
   else mediana=0
   end if
   return on
end function
```

6.1.8 fichero configuracion_emision_ultrasonidos.jal

```
-------------------------------------------------------------------------------------------------------------
--
--libreria encargada de configurar la emision de ultrasonidos. Define y configura las interrupciones  y timers
--asociados a la emision de la señal modulada a 40KHz, asi como las variables relacionadas con la emision de la señal.
--
-------------------------------------------------------------------------------------------------------------


--constantes y variables asociadas al anvio-recepcion de ultrasonidos a 40KHz
const byte FREC_PWM_REAL= 63--39.825KHz--69
const byte NUMERO_PULSOS_DEFECTO = 16
var volatile byte numero_pulsos
var volatile BYTE contadorPWM=0



--alias de los pines y registros asociados a la generacion de la señal ultrasonica
alias pinPWM              is pin_d0         --configurar el pin asociado a la emision de ultrasonidos
alias pinPWM_direction    is pin_d0_direction
alias habilitar_int_PWM   is intcon_tmr0ie
alias flag_int_PWM        is intcon_tmr0if
alias activar_timer_PWM   is t0con_tmr0on
alias valor_timer_PWM     is tmr0
alias registro_timer_PWM  is t0con

PINPWM_DIRECTION=output


--configuracion interrupcion envio ultrasonido
pinPWM_direction=output
valor_timer_PWM=255-FREC_PWM_REAL
registro_timer_PWM=0B0_1001_000
habilitar_int_PWM=ON
```

```
--interrupcion asociada al desobrdamiento del timer pwm: genera la señal modulada a 40KHz
PROCEDURE INTERRUPCION_EMISION() is
        PINPwm=!PINPwm
        flag_int_PWM=off
        valor_timer_PWM=255-FREC_PWM_REAL
        contadorPWM=contadorPWM +1
        RETURN
end PROCEDURE


--gestor de interrupciones
PROCEDURE sonar_isr_interrupcion_EMISION() is
        pragma interrupt
   IF (flag_int_PWM)    THEN
     INTERRUPCION_EMISION()
     return
   END IF
END PROCEDURE
```

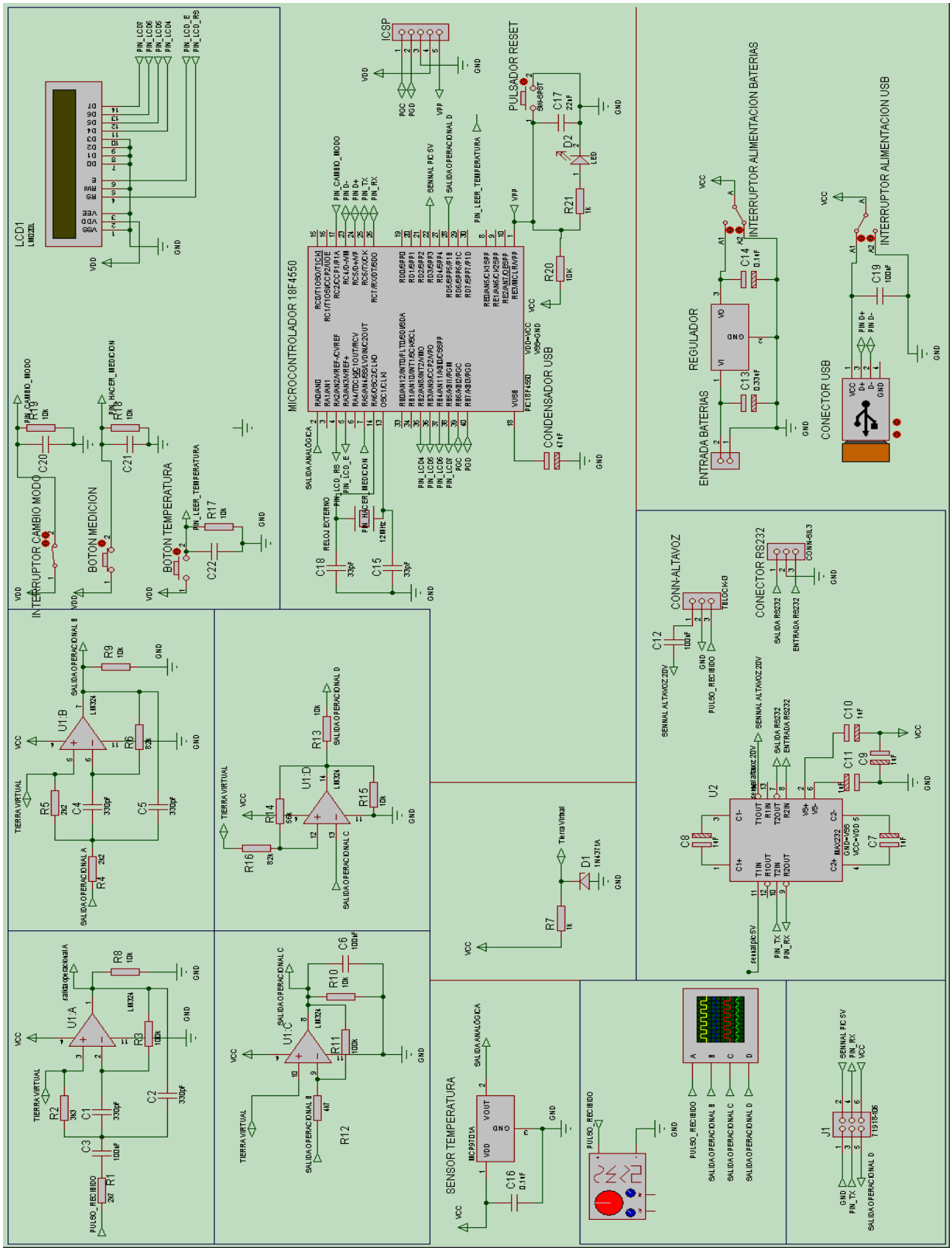
6.1.9 fichero configuracion_recepcion_ultrasonidos.jal

```
-------------------------------------------------------------------------------------------------------------------
--
--libreria encargada de configurar la recepcion de ultrasonidos. Define y configura las interrupciones  y timers
--asociados a la recepcion de la señal modulada a 40KHz, asi como las variables relacionadas con la recepcion.
--
-------------------------------------------------------------------------------------------------------------------


--alias de los pines y registros asociados a la recepcion de ultrasonidos
--se deberá escoger un pin que lleve asociado una interrupcion externa: int0, int1...
var volatile bit pinreceptor              is pin_b0
var volatile bit pinreceptor_direction    is pin_b0_direction
var volatile bit habilitar_int_RECEPTOR   is intcon_int0iE
var volatile bit flag_int_RECEPTOR                is intcon_int0if

--declaramos el pin receptor como pin de entrada
pinreceptor_direction                =input


--alias de los registros asociados al tiempo maximo de espera de eco
var volatile bit habilitar_int_lapso_maximo   is PIE1_TMR1IE
var volatile bit flag_int_lapso_maximo                is PIR1_TMR1IF
var volatile bit activar_timer_lapso_maximo   is t1con_tmr1on
var volatile word valor_timer_lapso_maximo    is tmr1
var volatile byte registro_timer_lapso_maximo is t1con


const word NUMERO_CICLOS_ESPERA          =12595



--configuracion interrupcion contador distancia maxima
valor_timer_lapso_maximo             =NUMERO_CICLOS_ESPERA
registro_timer_lapso_maximo           =0b00_11_000_0
habilitar_int_lapso_maximo           =ON
```

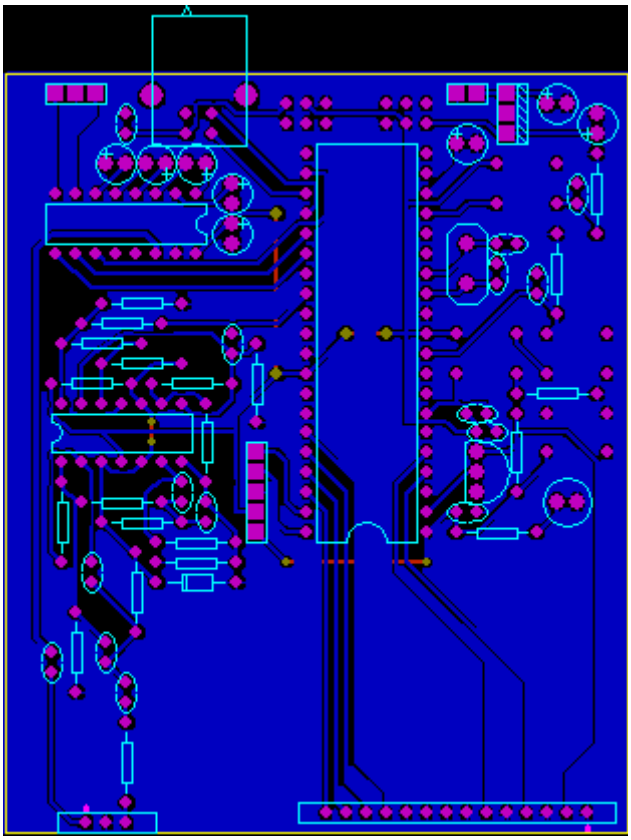
```
--variables globales asociadas al proceso de envio, recepcion y medida
var volatile word lapso_transcurrido         =0
var volatile bit medicionpwm                 =0



--interrupcion asociada a la rececpcion de eco de la señal ultrasonica
PROCEDURE INTERRUPCION_RECEPCION() is
   lapso_transcurrido=valor_timer_lapso_maximo - NUMERO_CICLOS_ESPERA
              medicionpwm=1
              flag_int_RECEPTOR=OFF
              return
end PROCEDURE

--interrupcion asociada al desbordamiento del timer 1: establece el tiempo de espera de eco máximo
PROCEDURE INTERRUPCION_LAPSO_MAXIMO() IS
              medicionpwm=1
   lapso_transcurrido=0
              flag_int_lapso_maximo=OFF
END PROCEDURE



PROCEDURE sonar_isr_interrupcion_RECEPCION() is
              pragma interrupt
   IF (flag_int_receptor)     THEN
     INTERRUPCION_RECEPCION()
     return
   END IF
              IF (flag_int_lapso_maximo)         THEN
                        INTERRUPCION_LAPSO_MAXIMO()
                        RETURN
              END IF
END PROCEDURE
```

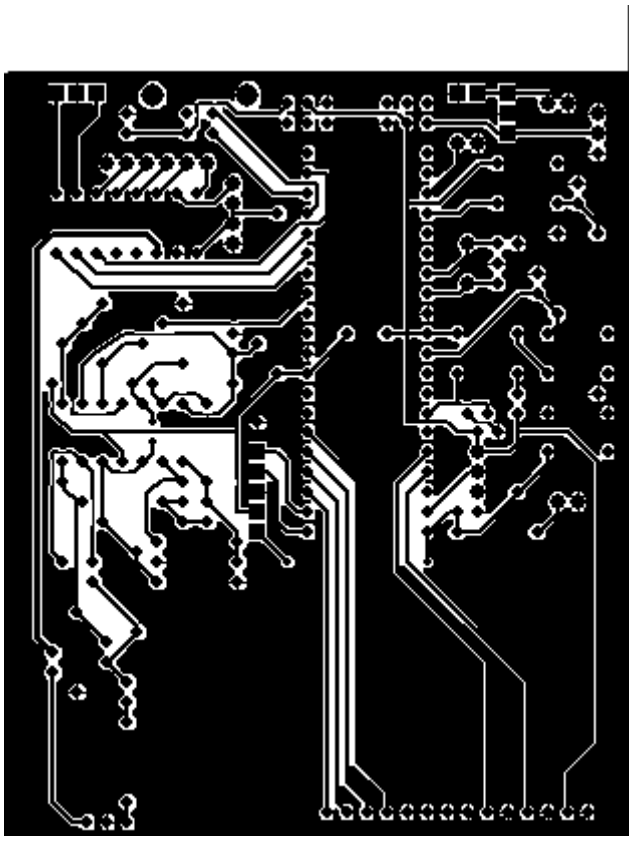
6.2 Esquemáticos y PCB´s del medidor de distancias

Diagrama esquemático del circuito eléctrico del medidor de distancia



PCB del circuito del medidor de distancia, junto con la plantilla para impresión.





Como puede observarse, hemos conseguido incluir todos lo componentes en una sola cara de la placa de circuito, lo que implica un diseño mucho más económico y fácil de desarrollar. Únicamente son precisas 3 vías en el lado de los componentes para poder producir el circuito.

Por otro lado, vemos que el circuito se ha realizado para componentes through-hole, que tiene como contrapartidas el ocupar un tamaño mucho mayor que en su versión SMD y con unos precios de los componentes algo más caros. No obstante decidimos emplear esta tecnología porque su implementación resulta más sencilla, así como poque teníamos un acceso mucho más sencillos a los componentes que en su versión SMD. En cualquier caso, se puede ver que el tamaño del circuito es bastante pequeño, cabiendo en

la palma de la mano.

Lista de componentes (BOM)

Bill Of Materials

```
=================
Design:   C:\PROYECTO FIN DE CARRERA\Receptor + emisor 1.2.DSN
Doc. no.: <NONE>
Revision: <NONE>
Author:   <NONE>
Created:  24/03/10
Modified: 13/11/10

QTY  PART-REFS           VALUE               CODE
---  ---------           -----               ----
Resistors
---------
1    R1                  2k7
1    R2                  3k3
2    R3,R11              100k
2    R4,R5               2k2
2    R6,R16              82k
2    R7,R21              1k
9    R8-R10,R13,R15,     10k
     R17-R20
1    R12                 4k7
1    R14                 56k

Capacitors
----------
7    C1,C2,C4,C5,C20-C22 330pF
4    C3,C6,C12,C19       100nF
5    C7-C11              1uF
1    C13                 0.33uF
2    C14,C16             0.1uF
2    C15,C18             33pF
1    C17                 22nF

Integrated Circuits
-------------------
1    U1                  LM324
1    U2                  MAX232

Diodes
------
1    D1                  1N4371A
1    D2                  LED
```

Miscellaneous

4	BOTON MEDICION, BOTON TEMPERATURA, INTERRUPTOR CAMBIO MODO, PULSADOR RESET	SW-SPST	
1	CONDENSADOR USB	47uF	
2	CONECTOR PLACA INFERIOR, CONECTOR PLACA SUPERIOR	CONN-SIL6	
1	CONECTOR RS232	CONN-SIL3	
1	CONECTOR USB	AU-Y1007-R	Digikey AE9925-ND
1	CONN-ALTAVOZ	TBLOCK-I3	
1	ENTRADA BATERIAS	CONN-SIL2	
1	ICSP	CONN-SIL5	
1	INTERRUPTOR ALIMENTACION	SW-SPDT	
1	J1	71918-106	FCI 71918-106
1	LCD1	LM020L	
1	MICROCONTROLADOR 18F4550	PIC18F4550	
1	REGULADOR	7805	
1	SENSOR TEMPERATURA	MCP9701A	
1	X1	RELOJ EXTERNO 20 MHz	

Además de todo lo anterior, para poder construir por completo el medidor de distancia faltaría por añadir el compartimento de las baterías.

6.3 Código fuente de los scripts matlab

A continuación de adjuntan los scripts de matlab que permiten el uso del medidor de distancia desde esta plataforma, permitiendo realizar todo tipo de mediciones así como lectura y modificación de datos del dispositivo. Para su correcto funcionamiento, solamente se requiere que estos ficheros estén todos en una misma carpeta.

6.3.1 Fichero Configuracion_inicial_sonar.m

```
%%este fichero se encarga de establecer la comunicación con el medidor de
distancias. Para ello se debe elegir correctamente el número del puerto COM
virtual al que está conectado, y cambiarlo en consecuencia. Ello puede verse
accediendo al administrador de dispositivos del sistema.
    delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
    delete(instrfind)
    clear all
end
s = serial('COM11');
set(s,'BaudRate',115200)
set(s,'DataBits',8)
set(s,'Parity','none')
set(s,'StopBits',1)
set(s,'FlowControl','none')
set(s,'Timeout',0.05)
fopen(s)

warning off

%limpiar buffer de salida
while (1)
        if isempty(fscanf(s)) break
        end
        pause(0.005);
end
```

6.3.2 Fichero Configuracion_parametros.m

```
%%este fichero incluye una serie de scripts encargados de modificar todos los
%parámetros del medidor de distancia.
%como puede verse, las ordenes de configuracion son todas iguales, comenzando
por el byte primario ("S"), solamente cambia para cada tipo de parámetro el byte
secundario


configuracion_inicial_sonar
aux=[];
medida=[];
%% configuracion del retardo
%importante; dependiendo de la plataforma, los datos pueden ser tratados
%en matlab como litte endian o big endian. dado que el protocolo pic
%implementado trabaja con big-endian, es necesario que cuando matlab le
%envie datos, estos estén codificados en dicho formato. generalmente no es
%un problema porque la mayoria de los datos, individualmente, tienen un
%solo byte de longitud, pero en el caso del retardo se deben
%introducir valores en el rango de los 0-65535, por lo que se hace
%imprescindible codificar al formato big endian

retardo=23;
[computerType, maxSize, endian] = computer;
aux=typecast(uint16(retardo),'uint8');
%si el formato nativo es little endian, cambiamos a big endian.
if endian=='L'
    aux=aux(end:-1:1);
end
%separamos los bytes 1, 2 3 4. solos nos interesan los 3 ultimos
byte1=aux(1);
byte2=aux(2);

while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('S'),uint8('D'),byte1,byte2];


    fwrite(s,mataux)
    pause(0.005);
     while (1)
        aux=fread(s);
        if isempty(aux) break
        else string(aux)
        end
    end
    pause(0.005)

pause
```

```
%% configuracion del lapso minimo

lapso_minimo=30;
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('S'),uint8('L'),uint8(lapso_minimo)];


    fwrite(s,mataux)
    pause(0.005);
     while (1)
        aux=fread(s);
        if isempty(aux) break
        else string(aux)
        end
    end

    pause(0.005)
    pause

% %% configuracion de la velocidad del sonido
% %importante; dependiendo de la plataforma, los datos pueden ser tratados
% %en matlab como litte endian o big endian. dado que el protocolo pic que he
% %implementado trabaja con big-endian, es necesario que cuando matlab le
% %envie datos, estos estén codificados en dicho formato. generalmente no es
% %un problema porque la mayoria de los datos, individualmente, tienen un
% %solo byte de longitud, pero en el caso de la vel. del sonido se deben
% %introducir valores en el rango de los 0-65535, por lo que se hace
% %imprescindible codificar al formato big endian
%
% vel_sonido=33999;
% [computerType, maxSize, endian] = computer;
% aux=typecast(uint16(vel_sonido),'uint8');
% %si el formato nativo es little endian, cambiamos a big endian.
% if endian=='L'
%     aux=aux(end:-1:1);
% end
% %separamos los bytes 1, 2 3 4. solos nos interesan los 3 ultimos
% byte1=aux(1);
% byte2=aux(2);
%
%
% while (1)
%         if isempty(fread(s)) break
%         end
%         pause(0.005);
%     end
% mataux=[uint8('S'),uint8('V'),byte1,byte2];
%
% fwrite(s,mataux)
% pause(0.005);
% while (1)
%     aux=fread(s);
%     if isempty(aux) break
%     end
```

```
% end
%
%     pause(0.005)
%
%     pause

%% configuracion numero de pulsos

numero_pulsos=18;
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('S'),uint8('N'),uint8(numero_pulsos)];


  fwrite(s,mataux)
pause(0.005);
while (1)
    aux=fread(s);
    if isempty(aux) break
    end
end

    pause(0.005)
    pause
%% configuracion de la deriva de temperatura
%importante; dependiendo de la plataforma, los datos pueden ser tratados
%en matlab como litte endian o big endian. dado que el protocolo pic
%implementado trabaja con big-endian, es necesario que cuando matlab le
%envie datos, estos estén codificados en dicho formato. generalmente no es
%un problema porque la mayoria de los datos, individualmente, tienen un
%solo byte de longitud, pero en el caso de la temperatura se deben incluir
valores
%retardo se deben en el rango de los -32767,32768, por lo que se hace
%imprescindible codificar al formato big endian

deriva_temperatura=-250;
[computerType, maxSize, endian] = computer;
if deriva_temperatura<0
    deriva_temperatura=2^16 + deriva_temperatura;
end
    aux=typecast(uint16(deriva_temperatura),'uint8');
%si el formato nativo es little endian, cambiamos a big endian.
if endian=='L'
    aux=aux(end:-1:1);
end
%separamos los bytes 1, 2 3 4. solos nos interesan los 3 ultimos
byte1=aux(1);
byte2=aux(2);

while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('S'),uint8('Z'),byte1,byte2];


    fwrite(s,mataux)
```

```
    pause(0.005);
     while (1)
        aux=fread(s);
        if isempty(aux) break
        else string(aux)
        end
    end
    pause(0.005)
    pause


%% configuracionde efecto de la humedad relativa
%%valor en escala 0-100, que se convierte a escala 0-255.
configuracion_inicial_sonar
humedad_relativa=50;
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('S'),uint8('H'),uint8((humedad_relativa*255)/100)];


  fwrite(s,mataux)
pause(0.005);
while (1)
    aux=fread(s);
    if isempty(aux) break
    end
end

    pause(0.005)
    pause


%% configuracion numero de medidas

numero_medidas=10;
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('S'),uint8('K'),uint8(numero_medidas)];


fwrite(s,mataux)
pause(0.005);
while (1)
    aux=fread(s);
    if isempty(aux) break
    end
end

    pause(0.005)
    pause
%% limpieza de buffer

        delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
```

```
    delete(instrfind)
    clear all
end
```

6.3.3Fichero lectura_parametros.m

```
%%este scripts se encarga de leer todos los parámetros internos del medidor de distancia. Para hacer más cómoda la presentación de resultados, se ha incluido una pausa de teclado entre cada medición.

configuracion_inicial_sonar

%% lectura retardo

while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('D')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='D')
        retardo=aux(3)*256+aux(4)
    end

    pause(0.05)

%% lectura lapso minimo
pause
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('L')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='L')
        lapso_minimo=aux(3)
    end
```

```
pause(0.05)
```

```
%% lectura velocidad_sonido
pause
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('V')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='V')
        vel_sonido=aux(3)*256+aux(4)
    end

    pause(0.05)
    pause

    %% lectura numero de pulsos
configuracion_inicial_sonar
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('N')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='N')
        num_pulsos=aux(3)
    end

    pause(0.05)
    pause
%% lectura temperatura

while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('T')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='T')
        temperatura=aux(3)*256+aux(4);

        %cambiamos de signo la temperatura si está en valores negativos
        if temperatura>2^15
            temperatura=-(2^16 - temperatura);
        end

        %pasamos al formato de °C; dividimos entre 100
```

```
        temperatura=temperatura/100

    end

    pause(0.05)

    %% lectura deriva temperatura
pause
while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('Z')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='Z')
        deriva_temperatura=aux(3)*256 + aux(4);
        if deriva_temperatura>2^15
            deriva_temperatura=-(2^16-deriva_temperatura);
        end
        deriva_temperatura/100
    end

    pause(0.05)
    pause

        %% lectura humedad relativa

while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('H')];


    fwrite(s,mataux);
    pause(0.005);
    aux=fread(s);
    if (aux(1)=='R'&&aux(2)=='H')
        humedad_relativa=aux(3)
    end

    pause(0.05)

%% lectura numero_mediciones

while (1)
        if isempty(fread(s,1)) break
        end
        pause(0.005);
    end
mataux=[uint8('R'),uint8('K')];


    fwrite(s,mataux);
    pause(0.005);
```

```
aux=fread(s);
if (aux(1)=='R'&&aux(2)=='K')
    numero_mediciones=aux(3)
end
```

```
    pause(0.05)

    %% limpieza buffer
        delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
    delete(instrfind)
    clear all
end
```

6.3.4 Fichero medicion_batch.m

```
%%este script realiza la petición de medicion multiple con retorno del conjunto
de datos en bruto  y en formato string (medicion_batch)
configuracion_inicial_sonar
aux=[];
medida=[];
%%

numero_medidas=1111;
while (1)
        if isempty(fread(s))
            break
        end
        pause(0.0001);
end

mataux=[uint8('B'),uint8(numero_medidas)];
fwrite(s,mataux)
pause(0.005);
tic

while (toc<3)
    b=fread(s);
    if ~isempty(b)
    break
    end
end
z=toc

tic


aux=b;
while toc<3
    aux2=fread(s);
       aux=[aux;aux2];
end

string(aux')
medida=[];
z=find(aux==10)';
for i=1:length(z)-1
    ini=z(i)+1;
    fin=z(i+1)-2;
    medida(i)=str2num((string(aux(ini:fin)')));
end

figure(1)
plot(1:length(medida),medida)
[(median(medida)),str2num(string(aux(2:4)))']


    delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
    delete(instrfind)
```

```
    clear all
end
```

6.3.5 Fichero medicion_batch_word.m

```
%%este script realiza la petición de medicion multiple con retorno del conjunto
de datos en bruto  y en formato big-endian (medicion_batch_word)
clear all
configuracion_inicial_sonar
aux=[];
medida=[];
%%

numero_medidas=21;
while (1)
        if isempty(fread(s))
            break
        end
        pause(0.0001);
end

mataux=[uint8('b'),uint8(numero_medidas)];
fwrite(s,mataux)
pause(0.005);
tic

while (toc<3)
    b=fread(s);
    if ~isempty(b)
    break
    end
end
z=toc
%pause(1)
tic

aux=b;
while toc<3
    aux2=fread(s);
       aux=[aux;aux2];
       pause(0.001)

end
medida=[];
j=1;
for i=3:2:length(aux)
    medida(j)=aux(i)*256+aux(i+1);
    j=j+1;
end
figure(1)
plot(1:length(medida),medida)
[(median(medida)),(aux(2))]


    delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
    delete(instrfind)
```

```
    clear s
end
```

6.3.6 Fichero medicion_mediana.m

```
%%este script se encarga de pedir al medidor de distancia un conjunto de
mediciones con retorno del valor de mediana del conjunto, formato string
%(medicion_multiple_media)
configuracion_inicial_sonar
aux=[];
medida=[];
%%

numero_medidas=51;
while (1)
        if isempty(fread(s))
            break
        end
        pause(0.0001);
end

mataux=[uint8('N'),uint8(numero_medidas)];
fwrite(s,mataux)
pause(0.005);
tic

while (toc<3)
    b=fread(s);
    if ~isempty(b)
    break
    end
end
z=toc

tic


aux=b;
while toc<3
    aux2=fread(s);
       aux=[aux;aux2];
end

string(aux')

medida=[];


cad=find(aux==10)
if length(cad>2)&&aux(1)=='N'&&aux(end)==10&&aux(end-1)==13
    numero_ceros=str2num(string(aux(2:cad(1)-2)'))
    mediana=str2num(string(aux(cad(1)+1:cad(2)-1)'))
end
```

6.3.7 Fichero medicion_mediana_word

```
%%este script se encarga de pedir al medidor de distancia un conjunto de
mediciones con retorno del valor de mediana del conjunto,
%formato numerico big-endian(medicion_multiple_media)
clear all
configuracion_inicial_sonar
aux=[];
medida=[];
%%

numero_medidas=21;
while (1)
        if isempty(fread(s))
            break
        end
        pause(0.0001);
end

mataux=[uint8('n'),uint8(numero_medidas)];
fwrite(s,mataux)
pause(0.005);
tic

while (toc<3)
    b=fread(s);
    if ~isempty(b)
    break
    end
end
z=toc
%pause(1)
tic

aux=b;
while toc<3
    aux2=fread(s);
       aux=[aux;aux2];
       pause(0.001)

end
medida=[];
j=1;

if(aux(1)=='n')
    numero_ceros=aux(2)
    mediana=256*aux(3)+aux(4)
end
```

```
    delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
    delete(instrfind)
    clear s
end
```

6.3.8 Fichero medicion_bucle.m

```
%%este script se encarga de realizar mil peticiones de medicion sinmple al
dispositivo, en formato string (medicion_simple).
%Los valores obtenidos los muestra por pantalla
configuracion_inicial_sonar
aux=[];
medida=[];
%%
for i=1:1000
%    fprintf(s,'%c','M','async');
    fwrite(s,uint8('M'))
    while (1)
        pause(0.005);
        %aux=fscanf(s,'%*c %d');
        aux=fread(s);
        aux2=find(aux==10); %caracter de fin de linea
        if isempty(aux) || isempty(aux2) break
        end
            aux3=sscanf(string(aux(2:aux2(1)-1)),'%d');

            if ~isempty(aux3)
                medida(i)=aux3
                i=i+1;
            else medida(i)=-1
        end
    end
    pause(0.005)
end

%%

numero_errores=length(find(medida==-1));
mediana=median(sort(medida));
numero_perdidas=length(find(medida==0));
maxlongitud=max(medida);

minlongitud=min(medida(find(medida>0)));
if isempty(minlongitud)
   minlongitud=0
end

figure(1);
clf(figure(1));
hold on
plot([1:length(medida)],medida,'b')
plot([1:length(medida)],minlongitud,'g-')
plot([1:length(medida)],100,'k+')
```

```
hold off

[numero_errores,numero_perdidas,maxlongitud,minlongitud,mediana]
    delete(instrfind)
if ~isempty(instrfind)
    fclose(instrfind)
    delete(instrfind)
    clear all
end
```