UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA POLITÉCNICA SUPERIOR DE GANDIA

I.T TELECOMUNICACIONES (SIST. ELECTRÓNICOS)

"Medida de la temperatura con conexión directa sensor-microcontrolador para estación marina"

TRABAJO FINAL DE CARRERA

Autor:

Xavier Ramón Siscar Escrivà

Directores:

José Pelegrí Sebastiá

Enrique Colomar Pous

GANDIA, 2013

Resumen

En el presente trabajo se ha diseñado e implementado un sistema de medida de la temperatura para una estación marina, usando una interfaz de conexión directa sensor-microcontrolador.

El sistema se basa en un sensor resistivo para medir la temperatura, un microcontrolador para procesar los datos y una memoria donde se almacenan. La interfaz de conexión directa sensor-microcontrolador permite realizar medidas de manera más rápida y con menor consumo respecto al uso de convertidores analógico-digitales. De esta forma logramos un sistema de bajo consumo desarrollado con componentes de bajo coste y que ofrece una precisión adecuada para su uso como instrumento de investigación en aplicaciones marinas.

Palabras clave: interfaz, directa, sensor-microcontrolador, pic18f24j50, pt1000.

Índice de contenidos

1- Introducción y objetivos

1.1- Introducción

En el campo de la investigación marina la monitorización y previsión de la temperatura superficial del mar es un factor muy importante. Midiendo sus variaciones se pueden explicar fenómenos meteorológicos , ya que afecta a la masa de aire que la cubre, y también marinos, ya que el mantenimiento de la población de fitoplancton y todas las especies que se alimentan de él depende en gran medida de las corrientes de agua fría o cálida.

Gracias al estudio del mar conocemos la causa de muchas catástrofes naturales, como por ejemplo los huracanes, y podemos preverlos. Actualmente gracias a los sensores controlamos multitud de factores de nuestro entorno y el objetivo de este trabajo es mejorar el rendimiento y el coste de un sensor de temperatura marino. En el mercado ya existen boyas que cumplen la misma función que la que se presenta pero con un coste mucho más elevado.

En este proyecto se explican las fases de desarrollo de un prototipo desde su diseño, implementación y las pruebas a las que se ha sometido para comprobar si realmente cumple las especificaciones para ser considerado un instrumento de investigación útil y de bajo coste.

En primer lugar, se describe brevemente el método convencional de medida, basado en el acondicionamiento de los sensores mediante amplificadores operacionales. Posteriormente se expone el nuevo sistema, basado en una interfaz directa sensor-microcontrolador y la medida del valor resistivo del sensor mediante cálculos basados en el tiempo de carga y descarga de la tensión de un condensador a través de varias resistencias.

Por último queda reflejado todo el proceso que se ha seguido y los problemas a la hora de diseñar desde cero una placa prototipo y programar el software necesario para el funcionamiento de todos los elementos del sistema.

1.2- Objetivos

1.2.1- Objetivo principal

Implementar la interfaz de conexión y el sistema de medida del sensor de temperatura PT1000 con el microcontrolador a través del software y hardware que sea necesario.

1.2.2- Objetivos secundarios

Calibrar el sensor para mejorar la medida.

Obtener una precisión y resolución adecuada para un estudio científico de la variación de temperatura a lo largo del tiempo en la superficie marina.

Almacenar los datos obtenidos para poder realizar el estudio.

2- Diseño del sistema de medida

2.1- Introducción a los registradores de temperatura marinos

Un registrador de datos (datalogger) es un dispositivo electrónico que registra datos en el tiempo o en relación a la ubicación por medio de instrumentos y sensores propios o conectados externamente. Por lo general son pequeños, con pilas, portátiles, y equipados con un microprocesador, memoria interna para almacenamiento de datos y sensores. Algunos registradores de datos se comunican con un ordenador personal y utilizan software específico para activar el registrador de datos, ver y analizar los datos recogidos, mientras que otros tienen un dispositivo de interfaz local (teclado, pantalla LCD) y puede ser utilizado como un dispositivo independiente.

Uno de los principales beneficios del uso de registradores de datos es la capacidad para recopilar automáticamente datos las 24 horas del día. Tras la activación, los registradores de datos normalmente se dejan sin vigilancia para medir y registrar la información durante toda la duración del período de seguimiento. Esto permite una visión global y precisa de las condiciones ambientales objeto de seguimiento, tales como la temperatura del aire y la humedad relativa.

En el caso de los registradores de temperatura marinos existen en el mercado una gran variedad de opciones pero con un coste relativamente elevado. A continuación podemos ver algunos modelos comerciales y su precio aproximado, ya que se han extraído de una web americana[1].

Fig.1. HOBO Tidbit Waterproof Temperature Data Logger [1].

Características:

Sumergible hasta 300 metros.

Rango de temperatura de -20°C a 70°C con precisión de ±0.20°C a 25°C.

Guarda 42,000 lecturas en una memoria no volátil.

Intervalo ajustable de 1 segundo a 18 horas.

Precio: 133€

Fig.2. HOBO U12 Stainless Steel Temperature Data Logger [1].

Características:

Sumergible hasta 300 metros.

Rango de temperatura de -40°C a 125°C con precisión de ±0.22°C.

Guarda 43,000 lecturas en una memoria no volátil.

Intervalo ajustable de 1 segundo a 18 horas.

Precio: 259€

Para este proyecto se ha tomado como referencia el siguiente modelo:

Fig.3. SBE 56 de Sea-Bird Electronics [1].

Características:

Sumergible hasta 1500 metros.

Rango de temperatura de -5°C a 45°C con precisión de ±0.22°C.

Guarda 15.9 millones de lecturas en una memoria no volátil.

Intervalo ajustable de 0.5 a 15 segundos.

Precio: No se ha encontrado un precio de referencia.

Así que el diseño de este sistema tratará de conseguir unos resultados lo más cercanos a estos que sea posible. Teniendo en cuenta las limitaciones del material empleado, presupuesto y recursos para la ejecución del proyecto.

2.2- Sistema de Medida

El sensor elegido para la medida de la temperatura es el denominado PT-1000. Es un sensor resistivo, es decir, que varía su valor de resistencia según el parámetro a medir, en este caso temperatura. Presenta una impedancia de 1000 Ohmios (1KΩ) a 0 ºC, y ésta varía en un factor de 3.89 Ω por grado Celsius. Conociendo esta relación podemos relacionar directamente el valor de resistencia en un instante de tiempo con la temperatura a la que se encuentra. En el apartado "Componentes del diseño" se explican más ampliamente sus características. A continuación se detalla el sistema de medida empleado para conocer el valor del sensor.

El sistema de medida habitual para un sensor resistivo, como es el PT-1000 utilizado en este proyecto, se basa en un circuito de acondicionamiento y un convertidor analógico-digital que digitaliza el nivel de tensión del sensor y nos permite calcular su valor en Ohmios. Pero este proceso requiere un tiempo de conversión y un consumo de energía considerable, es por eso que el método de conexión directa sensor-microcontrolador mejora las prestaciones del sistema y su eficiencia, alargando el período de trabajo al reducir el consumo de baterías.

Este sistema de medida ha sido tratado en el artículo "Improved Direct Interface Circuit for Resistive Full and Half-Bridge Sensors" [2]. En este artículo se estudian las diferentes formas de conexión directa del sensor al microcontrolador y el resultado concluye que el método más preciso es la medida con tres señales.

Este método expuesto en la Fig. 4 consiste en conectar directamente el sensor a un puerto del microcontrolador y a un condensador. Pero además del sensor y el condensador C, que proporciona un tiempo de descarga relacionado directamente con el valor de resistencia del sensor, se emplean dos resistencias de calibración para una medida más ajustada y estable ante los cambios de temperatura. El funcionamiento del sistema es el siguiente:

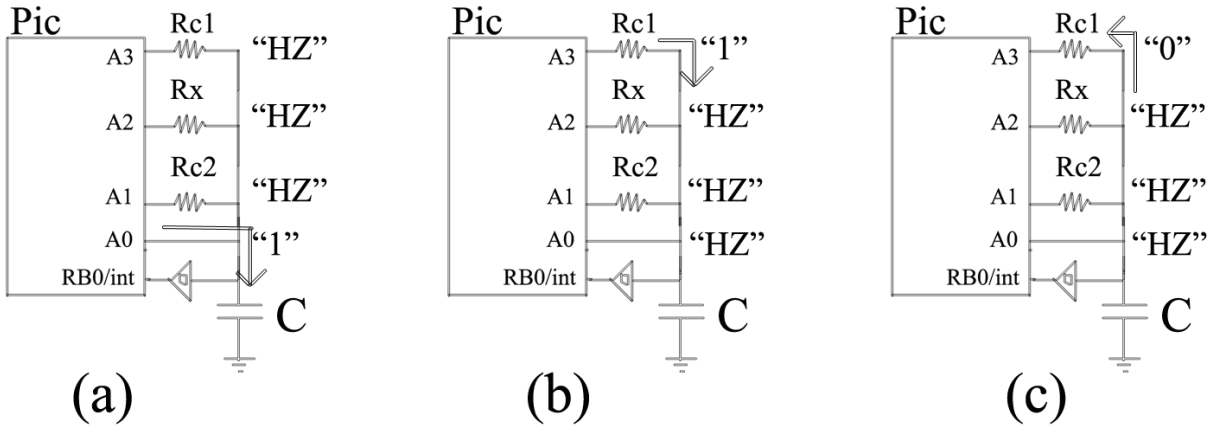

Fig.4. Descripción gráfica del método de medida usado.

(a) Desde la salida A0 se carga el condensador C enviando un “1” lógico (3.3V). Los demás puertos se configuran como entrada o “HZ” (alta impedancia).

(b) Se configura la línea a medir como salida, en este caso la línea de Rc1 que corresponde al puerto A3. Los demás puertos se configuran como entrada o “HZ” (alta impedancia) y se mantiene en la salida A3 un “1” lógico (3.3V) para mantener la carga del condensador.

(c) Se cambia el estado de la salida A3 a “0” para iniciar la descarga del condensador a través de Rc1, y el flanco de bajada provoca la interrupción en RB0/int. Se temporiza el intervalo de tiempo desde que cambia el estado de la salida hasta que se activa la interrupción, obteniendo el tiempo de descarga del condensador.

En el apartado de diseño del software se explica el proceso que sigue el programa ilustrado también en un diagrama de flujo en la Fig. 12.

Estos tres pasos (a b c) se repiten para cada línea (A3, A2, A1), obteniendo los tres tiempos de descarga necesarios para la calibración.

En el diseño final la resistencia Rc1 tiene un valor nulo, por lo que se conecta el cable directamente al condensador. De esta forma obtenemos un tiempo de descarga diferente para cada línea, que depende de la resistencia que tiene asociada. En el artículo citado anteriormente [2] se indica que para simplificar el cálculo debemos utilizar un valor de Rc2 cercano al máximo valor resistivo del sensor, y en Rc1 podemos prescindir de la resistencia conectando directamente la línea al condensador. Así pues después de realizar la medida obtenemos tres valores de tiempo que llamaremos, tRc1, tRc2 y tRx según la línea por la que se ha medido. Con estos tres valores y sabiendo que Rc2 tiene un valor fijo conocido, en este caso 1.2 kΩ, podemos calcular el valor de Rx gracias a la siguiente relación:

$$Rx = \frac{tRx - tRc1}{tRc2 - tRc1} * Rc2$$

Para el cálculo de la temperatura, sabiendo que el sensor tiene una impedancia de 1000Ω a 0º y un factor de 3.85 Ω por cada grado, el cálculo que hay que realizar una vez conocido el valor resistivo Rx es el siguiente:

Temperatura=((Rx-1000)/3.85)

De esta forma logramos medir un parámetro físico como la temperatura mediante una conexión directa sensor-microcontrolador. Pero para poder llevarlo a cabo es necesario un programa para automatizar todas estas tareas, en el apartado 8) se explica la forma en que se ha implementado todo este proceso de medida en lenguaje C++ para programar el microcontrolador.

2.3- Componentes del Diseño

La elección de los componentes es una tarea muy importante, ya que de ella depende el buen funcionamiento del sistema final. A continuación se detallan las características por las cuales ha sido elegido cada componente, aunque hay muchas opciones válidas en el mercado. Los principales factores de adecuación al proyecto son: el cumplimiento de los requisitos de tensión, las especificaciones propias del diseño y de los demás componentes, bajo consumo y bajo coste. Los principales componentes necesarios para el diseño son: un microcontrolador, un sensor de temperatura, una memoria, componentes pasivos y osciladores de cristal. Estos son los componentes elegidos y sus principales características.

2.3.1- PIC18f24j50

Este dispositivo [3] se ha escogido como pieza fundamental del proyecto ante la necesidad de un microcontrolador programable con unas características adecuadas a las especificaciones del diseño. Se han tenido en cuenta para este proyecto las siguientes características:

- 3 puertos de entrada/salida, suficientes para el programa usado en el diseño.
- Puertos I2C/SPI necesarios para enviar los datos de las mediciones a la memoria flash externa.
- Puerto USB V2.0 que usaremos para extraer la información de la memoria.
- Reloj en tiempo real y calendario integrado para asignar el tiempo a cada medida.
- Modo "Deep Sleep" que reduce el consumo a 850nA usando el RTCC (reloj en tiempo real y calendario) integrado para despertar el dispositivo en el intervalo de medida deseado.

2.3.2- Sensor PT1000

Para realizar la mediciones es necesario elegir un sensor adecuado de entre todas las posibilidades que ofrece el mercado. Para la medida de la temperatura, como en este caso, el tipo de sensor más utilizado son los denominados RTD (Resistance Temperature Detector), son sensores resistivos que permiten medir la temperatura mediante un cambio en su valor resistivo. Hay diversos tipos de RTDs y se clasifican según el material del cual están fabricados y su valor de impedancia a 0º. El sensor empleado en este diseño es, como se ha comentado anteriormente, el PT1000 (IST DIN 60751, class B). Debe su nombre al material usado en su fabricación, el platino (PT), y a la impedancia que presenta a 0º, 1000 Ω. Para que el sensor sea efectivo y podamos realizar una buena medida, se ha de mantener siempre una relación lineal de la variación de la magnitud desde un valor conocido, y por este motivo normalmente se escogen metales que mantengan esta linealidad como el platino, el cobre y el níquel. Cada material tiene asociado un coeficiente de temperatura que para el platino es de 0.00385, y utilizando este coeficiente de temperatura podemos hallar la relación de cualquier valor de temperatura con su valor resistivo mediante la siguiente ecuación:

$$R = Ro \cdot (1 + \alpha \cdot \Delta T)$$

donde:
Ro es la resistencia del metal a la temperatura de referencia To=0ºC, en el caso de la PT1000 se tiene Ro=1000Ω=RPT1000.
ΔT es la desviación de temperatura respecto a To (ΔT=T-To).
α es el coeficiente de temperatura del conductor especificado a 0 °C, interesa que sea de gran valor y constante con la temperatura.

De la misma forma que se obtiene el valor resistivo a partir de la temperatura, podemos emplear esta relación para conocer la temperatura a la que se encuentra el sensor midiendo su valor resistivo. Así es como funciona este diseño ya que lo que realmente está midiendo el sistema es el valor Rx del sensor y a partir de este valor, con un sencillo cálculo, se obtiene la temperatura. En el apartado "Sistema de medida" se explica el procedimiento a seguir para obtener el valor resistivo del sensor de manera fiable.

2.3.3- Memoria Flash 32Mbit SPI

Para almacenar los datos recogidos se necesita una memoria no volátil, ya que en el caso de agotarse las baterías del sistema la memoria queda grabada y se puede leer posteriormente. Para esta tarea se ha elegido una memoria de tipo flash teniendo en cuenta su capacidad y facilidad de uso en cuanto a programación se refiere. La memoria elegida es el modelo SST25VF032B [5]. Esta memoria flash de 32Mbit se programa mediante el protocolo SPI (Serial to Paralel Interface). Sus 32Mbit de capacidad permiten almacenar más de 2 millones de medidas, utilizando 8 bytes para almacenar los datos de cada medida: 2 bytes para la temperatura, 3 bytes para la fecha y 3 bytes para la hora.

Medida de Temperatura		Fecha			Hora		
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Parte entera	Parte decimal	Día	Mes	Año	Hora	Minutos	Segundos

En el apartado "Diseño del software" se explica ampliamente en el punto "Escritura en memoria" todo el proceso que se sigue para implementar el protocolo SPI y poder escribir y leer datos de la memoria.

2.3.4- Inversor 74HCT1G14SE (Trigger-Schmitt)

La función del inversor es evitar interferencias en el control del flanco de bajada que produce la descarga del condensador del circuito de medida. El inversor transforma este flanco de bajada en un flanco de subida y evita que se active la interrupción que controla la temporización por error. En el diseño final no ha sido incluido porque en el momento de realizar la compra de componentes no se encontraba disponible. Este cambio afecta al software ya que se debe tener en cuenta al configurar la interrupción que detecta la descarga del condensador.

2.3.5- Resistencias y Condensadores

Los componentes pasivos usados en el diseño han sido elegidos según las indicaciones de las hojas de características de cada componente. Además de éstos son necesarias dos resistencias de precisión del 0.1 % de tolerancia para el circuito de medida de la temperatura, ya que su valor se emplea como factor en los cálculos del programa.

2.3.6 Osciladores de cristal

En el circuito se ubican dos osciladores de cristal, uno principal de 40 Mhz que controla la ejecución del programa en el microcontrolador, y otro secundario de 32.768 Khz que sirve de referencia para el RTCC (Reloj en tiempo real y calendario) del Timer 1.

2.3.7- Lista de componentes

Para comprobar que el diseño se ajusta a su principal característica de bajo coste se expone aquí un listado con todos los componentes necesarios que han sido justificados en este apartado. De esta forma podemos ver que el precio total de un prototipo está alrededor de los 20€ por lo que se puede afirmar que cumple el requisito de bajo coste, teniendo en cuenta que el sensor tiene un coste elevado en relación con los demás componentes. Este es el listado de componentes, con su precio correspondiente y el coste total. El precio por unidad está basado en una producción de cien unidades.

Componente	Cantidad	Precio (ud.)
PT1000	1	6.18€
PIC18f24j50	1	2.38€
Memoria Flash 32Mbit SPI	1	1.93€
Inversor 74HCT1G14SE	1	0.118€
R 1.2K 0.1%	1	0.30€
R 330 0.1%	1	0.30€
R 10K	1	0.031€
R 1Meg	1	0.014€
R 220K	1	0.017€
C 10u	1	0.21€
C 2.2u	1	0.28€
C 0.1u	2	0.124€
C 33p	4	0.118€
Micro usb conector	1	0.9€
Cristal 40Mhz	1	0.51€
Cristal 32.768 Khz	1	0.91€
Fabricación PCB	1	4.5€
Montaje PCB	1	5€
Total		24€

El precio que falta por determinar es el ensamblaje sumergible de 100 a 300 metros de profundidad.

2.4- Diseño del Circuito

Las directivas que se han seguido para el diseño del circuito son varias. Primero se ha diseñado la red de componentes que acompaña al microcontrolador siguiendo las recomendaciones que se indican en su hoja de características. Estas son las conexiones mínimas imprescindibles para su funcionamiento que recomienda el fabricante.

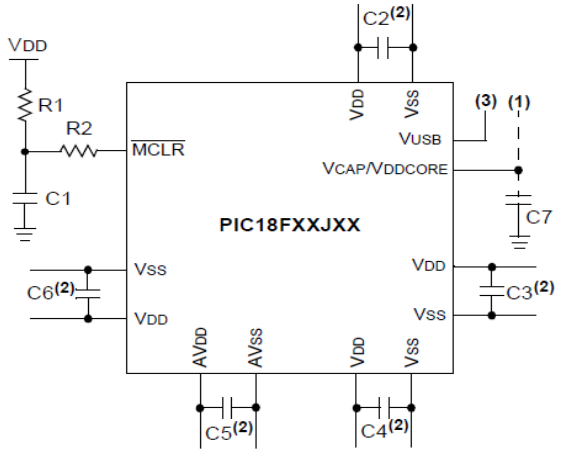

Fig.5. Recomendación de conexiones [3].

Condensadores: C1-C6 0.1uF, 20V cerámicos.
C7 10uF de 6.3 V de tántalo o de 10 V cerámico.

Resistencias: R1 10KΩ.
R2 de 100 Ω a 470 Ω.

A continuación se describen las conexiones del esquemático representado en la Fig.6, que es el circuito definitivo que se ha tomado como referencia para construir el prototipo, después de haber sufrido varias modificaciones causadas por errores en el diseño y montaje de los componentes.

Los osciladores se conectan siguiendo la nota de aplicación estándar del fabricante [3], que recomienda el valor de los condensadores que acompañan al cristal, en este caso de 22pF. El oscilador principal de 40 Mhz se conecta a las entradas OSC1 y OSC2 (Fig.6), mientras que el secundario, de 32.768 Khz a las entradas T1OSC y T1OSI (Fig.6).

Además de estos componentes, el microcontrolador necesita un puerto de programación (CON6), que se conecta al puerto de reset MCLR (Fig.6), a los pines de programación PGD y PGC (Fig.6), y además proporciona una fuente de alimentación alternativa a través del mismo puerto usb por el que se programa. El puerto usb se conecta a los pines d+ y d- (Fig.6), además de los de alimentación y el de alimentación externa (POWER) es la entrada para alimentar el sistema a través de una fuente o batería.

El sistema de medida descrito en el apartado 4) requiere de 4 puertos de entrada/salida y una entrada de interrupción. Se ha utilizado el puerto A para las entradas/salidas y la entrada RB0/INT (Fig.6) para el disparo de la interrupción. El inversor debería ir conectado entre el condensador que se descarga en las medidas y la entrada de interrupción, como se ve en el esquema, aunque el circuito funciona correctamente prescindiendo de él.

Por último, la memoria flash necesita dos pines para la comunicación SPI y uno para controlar la inhabilitación de la memoria, el CE (“Chip Enable”). Los pines de protección WP (“Write Protect”) y HOLD (Fig.6) se mantienen a nivel alto para permanecer inactivos.

El esquemático del circuito se ha diseñado usando el programa Orcad Capture que permite exportar estas conexiones al programa Orcad Layout para implementar la placa de circuito impreso, que será el próximo paso a seguir. Este es el resultado del diseño del esquema del circuito.

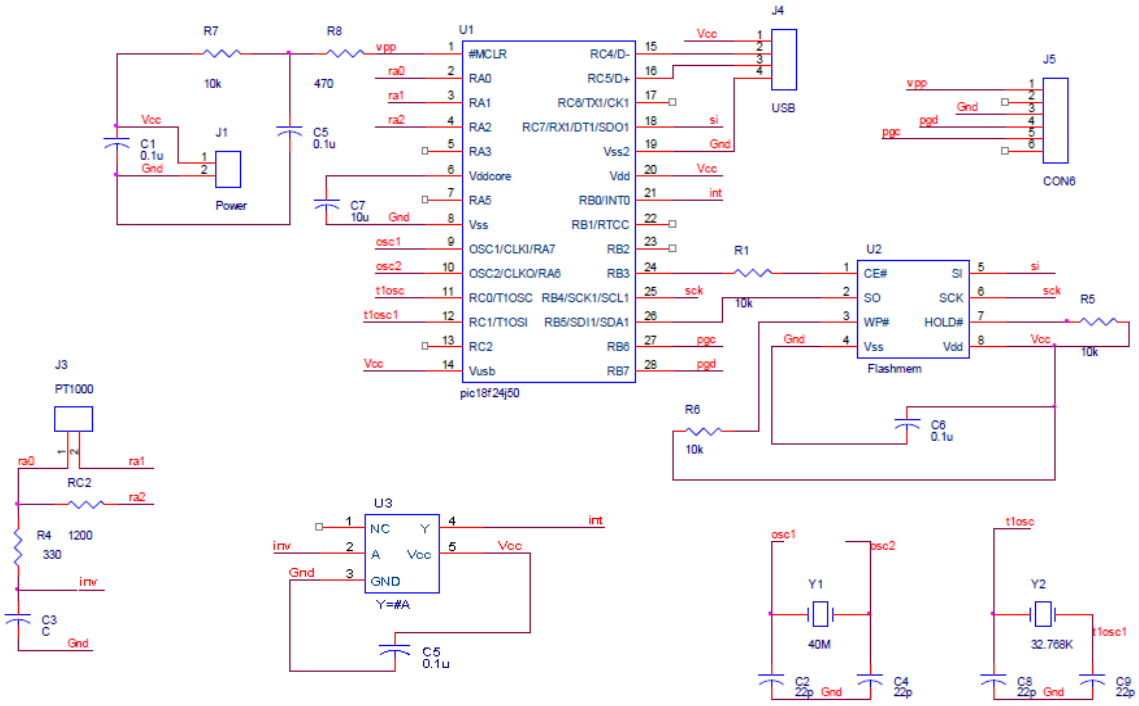

Fig. 5. Esquemático del circuito montado.

2.5- Diseño de la placa

El diseño de la placa se ha realizado mediante el programa Orcad Layout, que importa los datos de las conexiones del esquema diseñado en Orcad Capture y permite la implementación física del circuito mediante una placa insolada con pistas de cobre.

Para este diseño ha sido necesario crear los modelos de los componentes, utilizando las medidas que proporcionan sus hojas de características. Exceptuando los componentes pasivos que tienen un tamaño estandarizado, los demás componentes han sido creados expresamente para el proyecto. Un buen ejemplo de cómo se toman las medidas para un componente, es el conector usb que podemos ver en la imagen.

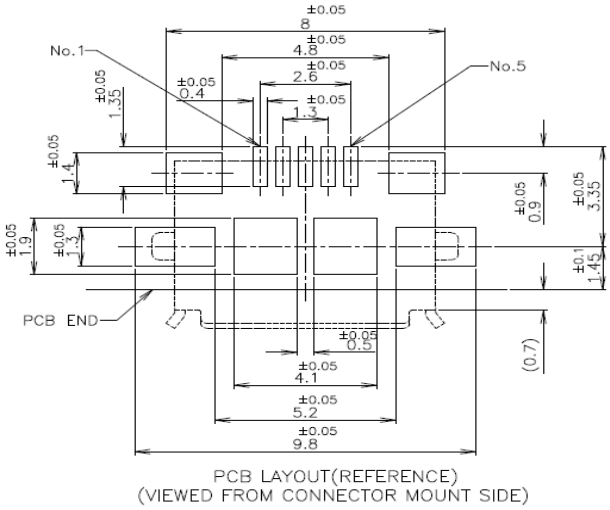

Fig.7. Huella del conector USB [4].

De esta forma se crea la huella o *footprint* de cada componente para que encaje perfectamente en sus puntos de soldadura. Una vez creadas las huellas de todos los componentes éstos se ubican siguiendo unas directrices que dependen de varios factores. Los conectores deberán ubicarse en los bordes de la placa para ser accesibles. Los demás componentes se ubican siguiendo una distribución lógica que depende de la ubicación de los pines en el microcontrolador para facilitar el trazado de pistas. En el caso de los osciladores también existe una recomendación por parte del fabricante [3] que consiste en crear un plano de masa que rodea los cristales y que se encuentren lo más cerca posible de las entradas del microcontrolador.

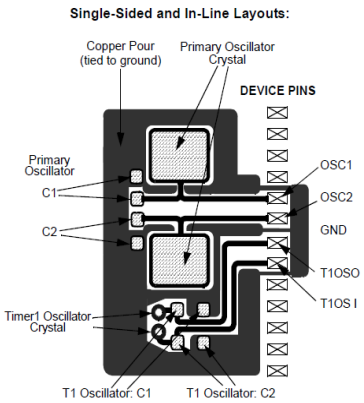

Fig.8. Recomendación de ubicación de los osciladores [3].

Al final de este proceso el resultado es una placa con pistas a doble cara y muy compacta, lo cual es una ventaja a la hora de encapsularla para su uso en el mar. Una vez terminado el diseño de la placa se crean los patrones para insolar en papel vegetal y se procede a la implementación física del circuito creando el prototipo y soldando los componentes a la placa. En las siguientes imágenes se ve el resultado del diseño que produce Orcad Layout y el resultado final con todos los componentes ensamblados.

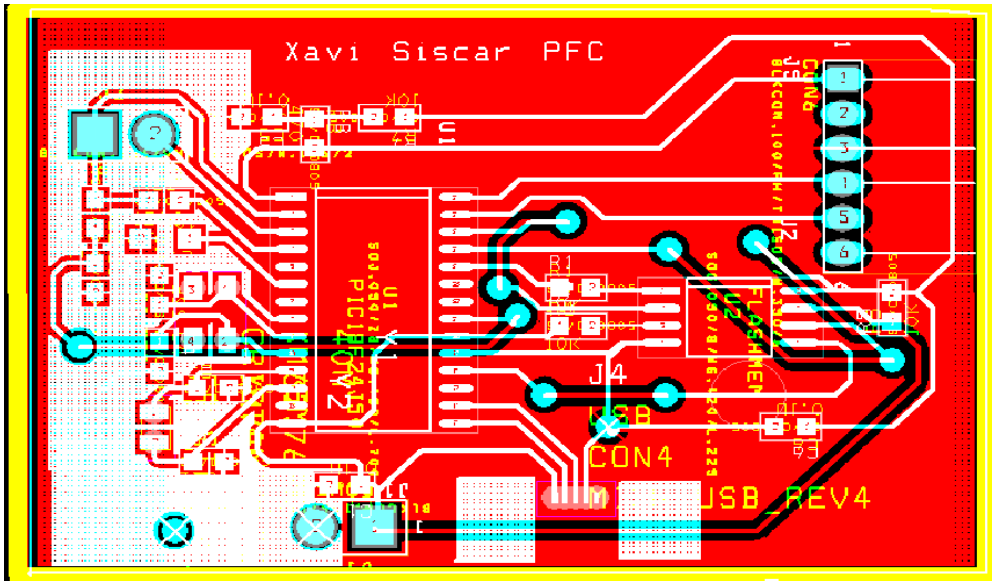

Fig.9. Diseño en Orcad Layout.

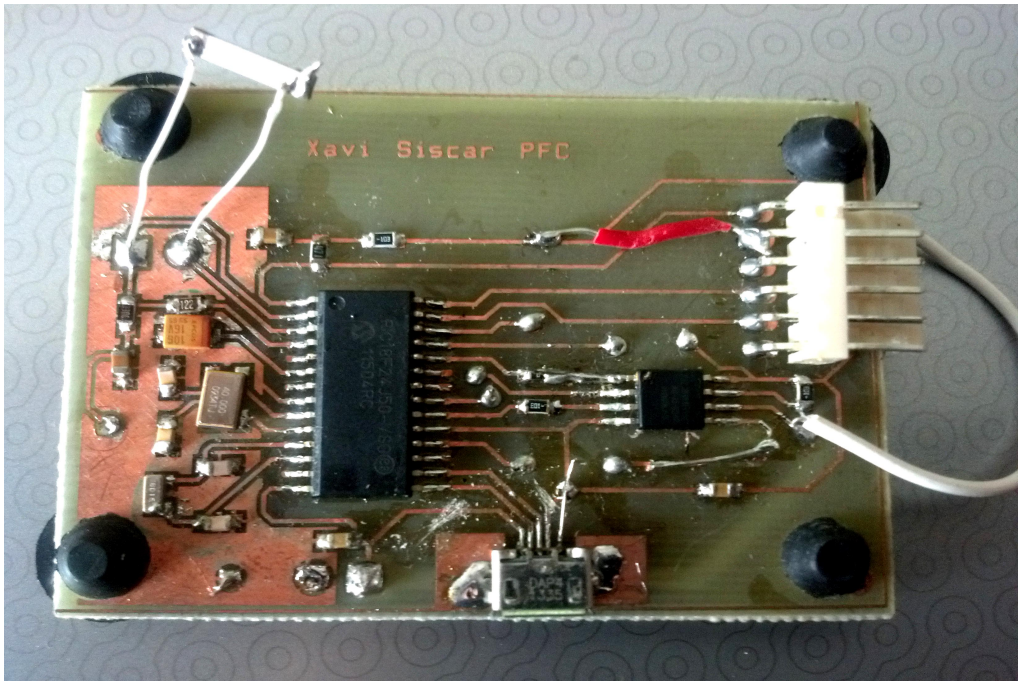

Fig.10. Placa ensamblada.

3- Diseño del software

3.1 Desarrollo del programa

Una vez ensamblada la placa de circuito impreso, hay que programar el microcontrolador para que realice las tareas necesarias para el diseño. El programa, ha sido creado en lenguaje C++ a través del software Mplab v8.85. Su estructura principal se basa en el siguiente diagrama de flujo.

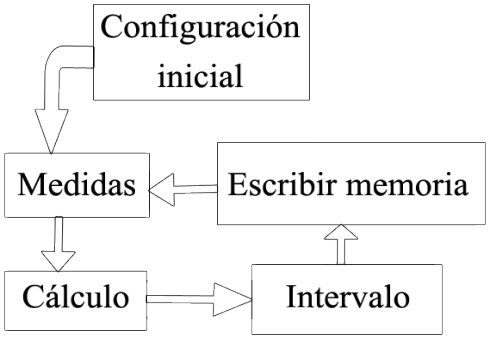

Fig.11. Diagrama del programa principal.

3.1.1- Funciones, variables y definiciones

En la cabecera del programa se incluyen las definiciones de funciones, variables y etiquetas que se asignan a nombres o valores para facilitar su uso en el programa. También se incluyen las librerías aunque en este caso solo se utiliza una librería genérica ya que todas las funciones estas integradas en el programa. Las funciones que utiliza el programa son las siguientes:

- Función de configuración de inicio (ini). En esta función se configuran todos los registros y se actualizan los valores de las variables globales antes de ejecutar el programa principal.

- Funciones de carga y descarga del condensador (carga_rc1, carga_rc2, carga_rx). Estas funciones se usan para controlar la carga y descarga del condensador C a través de tres puertos diferentes que se han de configurar como entradas o salidas dependiendo del estado de carga o descarga y de la línea a medir. En el apartado “Medidas” se explica su funcionamiento.

- Funciones de cálculo de las temporizaciones (calcula_trc1, calcula_trc2, calcula_tx, calculfinal). Estas funciones calculan el tiempo de descarga del condensador de cada línea usando el valor capturado por el Timer 3, y después se calcula la temperatura con estos datos.

- Función para el cálculo de la media de las medidas (media). Esta función almacena un número predeterminado de medidas entre 5 y 10 para hacer una media en el mismo instante de tiempo.

- Función para leer el valor de tiempo del RTCC (rtcc). Cuando se ejecuta esta función se actualizan las variables globales correspondientes a la hora y la fecha para poder ser usadas por el programa en el cálculo del intervalo y la escritura en memoria.

- Función para guardar las medidas en la memoria (guardar). Esta función escribe los datos de la medida, hora y fecha en la memoria.

- Función para leer la memoria (leermem). Cuando se configura el programa en modo lectura de memoria solo se ejecuta esta función que lee la memoria y guarda los datos en variables para poder leerlos mediante el *Pickit 3*.

- Función que temporiza el intervalo entre medidas (intervala). Esta función calcula el tiempo que espera el programa entre cada medida acorde al valor definido en la cabecera.

- Funciones para el manejo de la memoria flash por SPI (init, Send_Byte, Get_Byte, Poll_SO, CE_High, CE_Low, Read_Status_Register, send_ewsr, write_sr, send_wren, send_wrdi, Jedec_ID_Read, Read, Byte_Program, Chip_Erase). Estas funciones implementan el protocolo SPI para controlar las operaciones de memoria, su configuración inicial y la lectura y escritura.

- Variables. Las variables globales se pueden usar en cualquier parte del programa y modificar su valor, de esta forma se usan para almacenar los datos de las temporizaciones, hora y fecha.

- Etiquetas o *defines.* Mediante estas etiquetas podemos asignar equivalencias entre nombres o entre nombres y valores numéricos. De esta forma se pueden configurar los parámetros del programa de forma sencilla modificando los valores de la cabecera. Las etiquetas que se han utilizado son las siguientes:

- Lectura de memoria (lecturamem). Dependiendo de este valor el programa realiza medidas cuando se asigna el valor 0, o lee memoria al escribir un 1.

- Intervalo (intervaloseg, intervalomin, intervalohor). Para programar el intervalo entre medidas se utilizan estas tres definiciones, dos de ellas deben ser 0, y en una se puede escribir el valor de intervalo desado. Por ejemplo para un intervalo entre medidas de 10 segundos se escribe un 10 en intervaloseg y las demás a 0.

- Reloj y calendario (ANY, DIA, MES, HORA, DIASEM, SEG, MIN). Los valores de inicio del RTCC (Reloj en tiempo real y calendario) se programan en estas etiquetas y de esta forma se programa el reloj para contar a partir de la hora y fecha seleccionada.

- Comunicación por SPI (SO, SI, SCK, CE). Para facilitar la programación de las funciones de comunicación SPI, se asignan mediante estas etiquetas los puertos correspondientes con la función de cada pin. En el esquemático (Fig.4) se observan las conexiones de cada pin con la memoria y así se asignan la etiquetas de entrada de datos (SI), salida (SO), reloj (SCK) y *Chip Enable* (CE) a los puertos RC7, RB5, RB4 y RB3, respectivamente.

3.1.2- Configuración inicial

En la configuración inicial se asignan los valores a cada registro para seleccionar la opción mas adecuada al programa. En el anexo se incluyen las tablas de los registros utilizados con todas las opciones posibles de configuración. A continuación se exponen los motivos por que se ha elegido cada configuración en los registros más significativos.

- Registros CONFIG2L y CONFIG3L. En estos registros se encuentra la configuración del tipo de cristal y los osciladores conectados al microcontrolador. Se ha elegido el tipo de cristal HSPLL para el oscilador principal de 40Mhz que controla la ejecución del programa. Como fuente para el Timer 1 que se usa como referencia para el RTCC se ha seleccionado un oscilador externo conectado a T1OSC/T1CKI, que corresponde con el oscilador secundario de 32.768 kHz.

- Registros ANCON0, ANCON1, TRISA, TRISB y TRISC. Estos registros controlan la configuración de los puertos en cuanto a modo digital o analógico y los pines de entrada o salida. Se han configurado todos los puertos en modo digital, ya que uno de los objetivos del proyecto es eliminar el uso de puertos analógicos que requieren convertidores. Los pines de entrada y salida se han configurado de la siguiente forma:

- Puerto A. Se han configurado todos los pines como entrada de alta impedancia aunque ésta configuración varía durante la ejecución el programa. Como se ha explicado en el apartado “Sistema de medida” es necesario alternar los pines de entrada y salida de este puerto para conseguir la carga y descarga del condensador por el pin correspondiente.

- Puerto B. Los pines de este puerto han sido configurados combinando entradas y salidas. Las salidas elegidas son los pines 3 y 4 que se usan en la comunicación SPI y corresponden al *Chip Enable* y la señal de reloj, respectivamente. El pin RB0 se ha configurado como entrada porque se usa en modo de interrupción por flanco para detectar el momento en que se descarga el condensador.

- Puerto C. Este puerto se ha configurado como entrada exceptuando el pin 8 que corresponde a la salida de datos en la comunicación SPI.

- Registros INTCON y PIE. En estos registros se encuentra la configuración de las interrupciones del programa. Se activan las interrupciones globales y las del Timer 0, Timer 3 y RB0. La interrupción RB0 se configura en modo flanco descendente para detectar la descarga del condensador y poder temporizarla.

3.1.3- Configuración RTCC

En este apartado se explica el proceso a seguir para habilitar el Reloj en tiempo en tiempo real y calendario y programar sus valores iniciales en los campos año, día, mes, hora, día de la semana, segundos y minutos. Se usan valores definidos al principio del programa para facilitar su cambio. Para introducir los valores deseados en cada registro se utiliza un puntero denominado RTCVAL y mediante una secuencia de comandos se envían los valores siguiendo un orden determinado. En el anexo se incluye el programa completo con la secuencia de comandos necesaria para programar el reloj.

EXAMPLE 17-1: SETTING THE RTCWREN BIT

```
movlb   0x0F            ;RTCCFG is banked
bcf     INTCON, GIE     ;Disable interrupts
movlw   0x55
movwf   EECON2
movlw   0xAA
movwf   EECON2
bsf     RTCCFG,RTCWREN
```

Fig.12. Secuencia de comandos para habilitar el reloj [3].

3.1.4- Configuración memoria flash

A través de estas funciones se habilita la memoria para poder escribir en ella. En el apartado correspondiente a la programación de la memoria se explica detalladamente su funcionamiento. El proceso a seguir es el siguiente:

a. Deshabilitar cualquier ajuste previo enviando el comando WRDI.

b. Habilitar la escritura del registro enviando el comando EWSR.

c. Escribir un 0 en el registro mediante el comando WSR para quitar la protección de todos los bloques de la memoria

d. Habilitar la escritura enviando el comando WREN.

3.1.5- Medidas

En el apartado “Sistema de medida” se explica detalladamente el proceso a seguir para realizar las medidas, la forma de implementarlo en este ejemplo se repite para cada uno de los tres puertos que realizan la misma función de medida del tiempo de descarga de un condensador. El valor obtenido por cada puerto depende de la resistencia por la que se descarga el condensador, cuyo valor es conocido exceptuando el caso del sensor que es el que queremos obtener. El diagrama de flujo del sistema de medida es el siguiente:

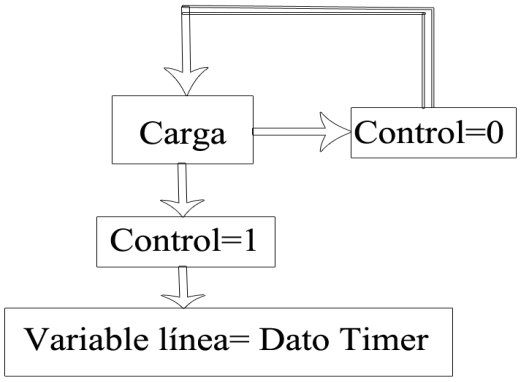

Fig.13. Diagrama del sistema de medida.

En el programa principal se ejecuta la función de carga, y después se espera hasta el final de la descarga que corresponde a la activación de la interrupción RB0 por flanco descendente. Para asegurar la captura del dato en el momento posterior a la interrupción se utiliza la variable "control" que se activa dentro de la rutina de atención a la interrupción y se desactiva cada vez que se produce la carga. Finalmente se captura el dato de temporización y se guarda en la variable correspondiente a la línea medida.

La función de carga realiza la secuencia de comandos siguiendo los pasos descritos en el apartado "Sistema de medida" cambiando la configuración de los puertos, los niveles, el tiempo de carga, interrupciones y la variable de control. Primero se deshabilita la interrupción RB0/INT para evitar una falsa interrupción y se pone a 0 la variable de control de la descarga. Después se configura el puerto como entrada exceptuando la línea a medir que es necesario que sea una salida para poder cambiar su valor. Posteriormente se envía un 1 lógico por la línea a medir y se espera un tiempo para cargar el condensador, una vez cargado se activan las interrupciones y el temporizador y se cambia el valor del puerto a 0 para descargar el condensador. De esta forma cuando se descarga el condensador se activa la interrupción RB0 en cuya rutina de atención se captura el dato de tiempo y se activa la variable de control para que el programa principal pueda guardar el dato en la variable correspondiente.

3.1.6- Interrupciones

Las interrupciones se asignan a una función prioritaria que se activa mediante un indicador o "flag", cuando se activa algún indicador de interrupción el programa se detiene para ejecutar otros comandos englobados en una función, una vez finalizada la ejecución de esta función, se volverá a ejecutar la línea donde se detuvo anteriormente, por eso reciben el nombre de interrupciones.

En este diseño se utilizan tres rutinas de atención a la interrupción, dos para los temporizadores y una para el flanco descendente de RB0. Cada rutina se ejecuta dependiendo del *flag* o indicador que haya activado la interrupción. Estos son los tres *flags* que se pueden activar y su función:

- *Flag* TMR3IF. Si el Timer 3 se desborda se activa esta interrupción, como el Timer 3 controla el tiempo de descarga si se activa significa que el temporizador a llegado a su límite d medida y por lo tanto se ha producido un error ya que el temporizador está configurado para que los tiempos de medida no superen su límite.

- *Flag* INT0IF. Cuando se produce un flanco descendente en RB0 se activa esta interrupción, significa que se ha descargado el condensador y por lo tanto se para el Timer 3 y se guarda su valor en una variable que el programa principal asignará a la línea que se ha medido. Como se ha comentado anteriormente es aquí donde se activa la variable de control que indica al programa principal que se ha descargado el condensador.

- *Flag* T0IF. Si el Timer 0 se desborda se activa esta interrupción, este temporizador se usa para el retardo de carga del condensador. Con una variable se controla cuantas veces ha de ejecutarse este retardo por lo que cada vez que se activa esta interrupción simplemente se incrementa esta variable y se vuelve a cargar el temporizador.

3.1.7- Cálculo

Una vez obtenidos los valores de temporización timer_rc1, timer_rc2 y timerx, correspondientes a cada línea, se realiza el cálculo para obtener Rx, el valor resistivo del sensor, y así determinar la temperatura. En el apartado "Sistema de medida" se explican los cálculos necesarios para obtener estos valores, son los siguientes:

$$Rx = \frac{tRx - tRc1}{tRc2 - tRc1} * Rc2$$

Temperatura=((Rx-1000)/3.85)

Además de éstos cálculos, el sistema realiza cinco medidas consecutivas y calcula el valor promedio que es el que se guarda en la memoria.

3.1.8- Intervalo

Para programar el intervalo de tiempo entre medidas se define al inicio del programa un valor para intervalohor, intervalomin o intervaloseg. Sólo es posible programar un tipo de intervalo en horas, minutos o segundos, de esta forma si elegimos un intervalo de 5 segundos, los demás valores deben ser 0, y así ocurre también con los minutos y las horas. En el programa se llama a la función intervala() cada vez que se realiza una medida, después de guardar los datos en la memoria. El cometido de esta función es esperar el tiempo programado hasta la siguiente medida y para ello se usa la función "rtcc" que actualiza las variables del reloj para calcular el tiempo transcurrido desde la última medida. Cuando el tiempo coincide con el intervalo finaliza la función y el programa principal continua realizando medidas. Por lo tanto no hay límite de programación del intervalo.

3.1.9- Escritura en memoria

La escritura de los datos de las mediciones en la memoria flash se hace mediante el protocolo SPI (Serial to Paralel Interface) o Interfaz Serie a Paralelo. El Bus SPI es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. Su funcionamiento es relativamente sencillo y se basa en el envío de un flujo de bits en serie regulado por un reloj.
Para controlar el envío de datos se necesitan las líneas de: reloj, dato entrante, dato saliente y un pin de chip enable, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. En este diseño las líneas se han asignado de la siguiente forma:

- Dato saliente. SO salida por el pin RB5.
- Dato entrante. SI entrada por el pin RC7.
- Reloj. SCK salida por el pin RB4.
- Chip enable. CE salida por el pin RB3.

En las definiciones iniciales del programa se han establecido estas correspondencias para facilitar su uso en las funciones que controlan la memoria. Estas funciones permiten enviar y recibir datos de la memoria, de esta forma es posible configurarla enviando unos comandos determinados.

La forma en que se envían y reciben los datos de la memoria está definida por el protocolo SPI, primero hay que activar el dispositivo con un "0" lógico en el pin correspondiente al Chip Enable y luego enviar el comando correspondiente sincronizado con la señal de reloj. En la siguiente imagen se describe gráficamente el proceso.

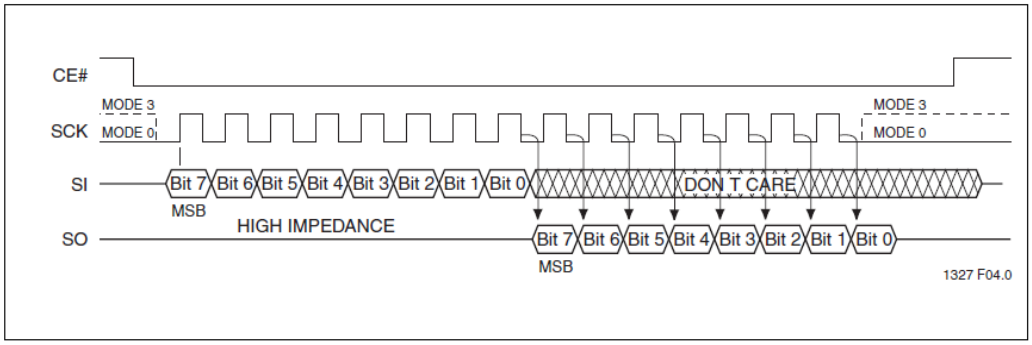

Fig.14. Protocolo SPI [5].

Ejemplos de comandos (hexadecimal):

Tabla 1: Códigos de memoria.

Instruction	Description	Op Code Cycle[1]	Address Cycle(s)[2]	Dummy Cycle(s)	Data Cycle(s)	Maximum Frequency
Read	Read Memory	0000 0011b (03H)	3	0	1 to ∞	25 MHz
High-Speed Read	Read Memory at higher speed	0000 1011b (0BH)	3	1	1 to ∞	80 MHz
4 KByte Sector-Erase[3]	Erase 4 KByte of memory array	0010 0000b (20H)	3	0	0	80 MHz
32 KByte Block-Erase[4]	Erase 32KByte block of memory array	0101 0010b (52H)	3	0	0	80 MHz
64 KByte Block-Erase[5]	Erase 64 KByte block of memory array	1101 1000b (D8H)	3	0	0	80 MHz
Chip-Erase	Erase Full Memory Array	0110 0000b (60H) or 1100 0111b (C7H)	0	0	0	80 MHz
Byte-Program	To Program One Data Byte	0000 0010b (02H)	3	0	1	80 MHz
AAI-Word-Program[6]	Auto Address Increment Programming	1010 1101b (ADH)	3	0	2 to ∞	80 MHz
RDSR[7]	Read-Status-Register	0000 0101b (05H)	0	0	1 to ∞	80 MHz
EWSR	Enable-Write-Status-Register	0101 0000b (50H)	0	0	0	80 MHz
WRSR	Write-Status-Register	0000 0001b (01H)	0	0	1	80 MHz
WREN	Write-Enable	0000 0110b (06H)	0	0	0	80 MHz
WRDI	Write-Disable	0000 0100b (04H)	0	0	0	80 MHz
RDID[8]	Read-ID	1001 0000b (90H) or 1010 1011b (ABH)	3	0	1 to ∞	80 MHz
JEDEC-ID	JEDEC ID read	1001 1111b (9FH)	0	0	3 to ∞	80 MHz

Una vez conocido el proceso es necesario incluir en el programa una función que envíe los datos y otra que los reciba siguiendo el protocolo. A partir de estas dos funciones y usando los comandos citados anteriormente es posible manejar la memoria creando funciones más complejas para poder leer y escribir los datos. En este ejemplo se explican estas dos funciones básicas y una más compleja creada a partir de las dos.

Función para enviar un byte: Esta función recibe como dato el byte a enviar y se crea después un bucle que se repite ocho veces, una por cada bit. Este bucle consiste en alternar la salida de reloj para crear la señal necesaria para sincronizar la memoria y ir enviando uno por uno los bits. Se envía primero el bit de mayor peso y luego se desplaza la variable hasta llegar al último bit.

Función para recibir un byte: Esta función, de forma análoga a la función de enviar, alterna la salida de reloj para crear la señal necesaria para la memoria pero en este caso los bits recibidos conforman una variable que se devuelve al programa principal.

Función para leer un byte de la memoria: Esta función combina las dos anteriores ya que para leer la memoria es necesario enviar varios bytes para luego recibir el byte almacenado en una posición de la memoria. Primero se envía el comando *Read* (Fig.13) y la dirección a leer, que ocupa tres bytes. Después con la función para recibir un byte se obtiene el valor de la posición de memoria que se ha enviado.

En este diseño se guardan en la memoria los datos de la temperatura registrada, hora y fecha. Una vez explicado el proceso para escribir en la memoria lo siguiente es que el programa envíe todos estos datos a la memoria usando las funciones explicadas anteriormente. Hay que tener en cuenta que la temperatura se guarda en una variable de tipo float, es decir, con decimales. Antes de guardar su valor en la memoria se separa en dos bytes, uno para el valor entero y otro para el decimal. De esta forma solo queda enviar ordenadamente los datos después de cada medida, en el anexo se puede consultar el código completo del programa comentado para una mejor comprensión.

4- Resultados experimentales

Para comprobar el funcionamiento del sistema se ha utilizado el modo In-circuit debugging del programador Pickit 3. Este dispositivo permite, además de programar el microcontrolador, hacer una depuración del programa ejecutándolo directamente en la placa donde esta montado el sistema.

Fig.15. Pickit 3 [6].

De esta forma es posible ejecutar el programa paso a paso o crear puntos de ruptura donde se para la ejecución y mediante la función Watch del Mplab se puede mostrar el valor de las variables en pantalla. Esto resulta muy útil a la hora de comprobar si el programa esta funcionando correctamente y realizar los cambios necesarios.

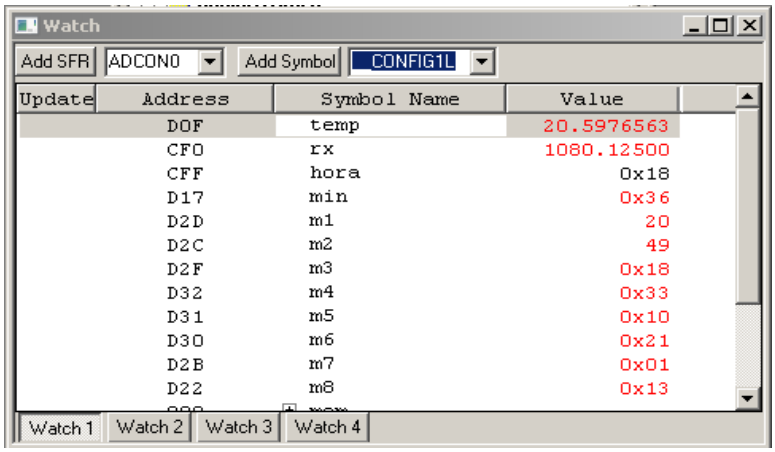

Fig.16. Ventana Watch del Mplab.

Utilizando el método de los puntos de ruptura se ha elaborado una tabla con los valores medidos por el programa al conectar un valor de resistencia determinado por una caja de décadas que permite seleccionar la impedancia. Luego se ha comparado el valor medido por el programa con una medida realizada por un multímetro digital con conexión a 4 hilos. Cabe recordar que al conectar una resistencia en lugar del sensor no se está modificando la temperatura a la que se encuentra el sistema por lo que para un estudio más a fondo sería necesario utilizar una cámara climática ya que es la única forma de comprobar la respuesta frente a los cambios de temperatura.

Se ha utilizado una caja de décadas calibrada modelo ZX99 (Fig.17) que incorpora unos potenciómetros regulables con un valor de resistencia asignado, de esta forma es posible obtener un valor de resistencia desde prácticamente 0 Ω hasta varias decenas de KΩ , aunque en este caso se está simulando un sensor PT1000 que varía su impedancia desde 950 Ω hasta 1.2 KΩ .

Fig.17. Caja de décadas.

La medida a 4 hilos es la forma más precisa de medir una impedancia ya que si se utilizan cuatro conductores, la corriente de salida y la tensión medida se pueden separar. Los terminales del medidor se denominan “Source” (Fuente) para la generación de corriente y “Sense” (Detección) para la entrada de tensión.
La resistencia en serie en los conductores “fuente” no afecta el flujo de corriente. Asimismo, por los conductores de “medida” casi no hay flujo de corriente, debido a la alta impedancia de entrada del medidor. Esto significa que no hay caída de tensión en los conductores de prueba. De este modo, sólo se mide la caída de tensión en la resistencia desconocida debido a la corriente que fluye por él.

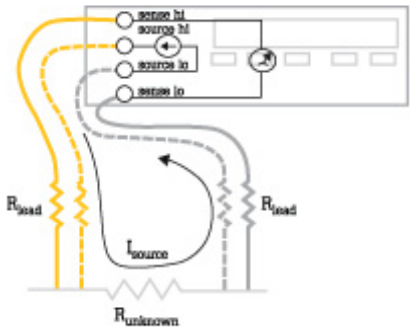

Fig.18. Medida a 4 hilos [7].

Se ha utilizado un multímetro digital modelo 34401A de Hewlett Packard [8] para realizar las medidas de impedancia a 4 hilos. Este dispositivo permite conocer el valor de impedancia con una resolución de dos decimales y una precisión de 0.01 Ω [8]. En la fig. 19 se observa el frontal del multímetro donde se conectan los 4 hilos para medir la impedancia y la pantalla donde se lee el resultado en formato digital.

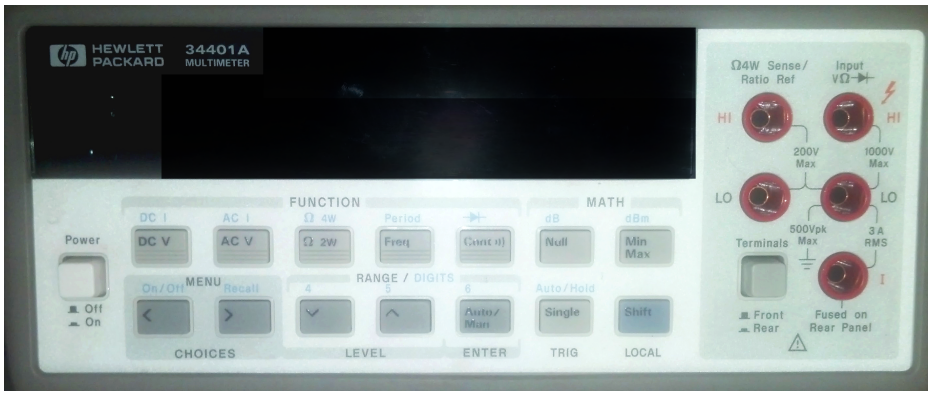

Fig.19. Multímetro digital.

En la siguiente tabla la columna “Rx (Ohm)” corresponde al valor óhmico seleccionado en la caja de décadas, mientras que “Rx Multímetro” es el resultado de la medición a 4 hilos y “Rx MEDIDA” es la que se obtiene del programa capturando la variable “rx”. También se refleja el valor de temperatura que corresponde a cada medición y la diferencia del valor medido por el programa con el valor teórico y el medido por el multímetro. También se han representado en tres gráficas los diferentes valores para ilustrar los resultados.

Tabla 2: Mediciones.

RX (Ohm)	RX MEDIDA	RX Multímetro	TEMP TEORIC ºC	TEMP MEDIDA ºC	Temp diff	dIF MULTIMETRO	Rx diff
950	949,59	950,03	-12,99	-13,09	0,11	-0,44	0,41
970	969,95	970,04	-7,79	-7,81	0,01	-0,09	0,05
990	990,07	990,04	-2,60	-2,58	-0,02	0,03	-0,07
1010	1010,03	1010,06	2,60	2,61	-0,01	-0,03	-0,03
1030	1030,03	1030,06	7,79	7,80	-0,01	-0,03	-0,03
1050	1050,09	1050,05	12,99	13,01	-0,02	0,04	-0,09
1070	1070,06	1070,05	18,18	18,20	-0,02	0,01	-0,06
1090	1090,09	1090,05	23,38	23,40	-0,02	0,04	-0,09
1110	1110,09	1110,05	28,57	28,59	-0,02	0,04	-0,09
1130	1130,03	1130,05	33,77	33,77	-0,01	-0,02	-0,03
1150	1150,09	1150,05	38,96	38,98	-0,02	0,04	-0,09
1170	1170,31	1170,03	44,16	44,24	-0,08	0,28	-0,31
1190	1190,25	1190,03	49,35	49,42	-0,06	0,22	-0,25
1200	1200,15	1199,94	51,95	51,99	-0,04	0,21	-0,15

El primer gráfico (Fig.20) muestra la curva teórica de valores de resistencia aplicados superpuesta con la curva creada con los valores medidos por el programa. Se observa que no existe apenas diferencia entre las dos ya que como se ve en la tabla el coeficiente de correlación “r2” es prácticamente 1, por eso se han creado dos gráficas más para poder apreciar las diferencias.

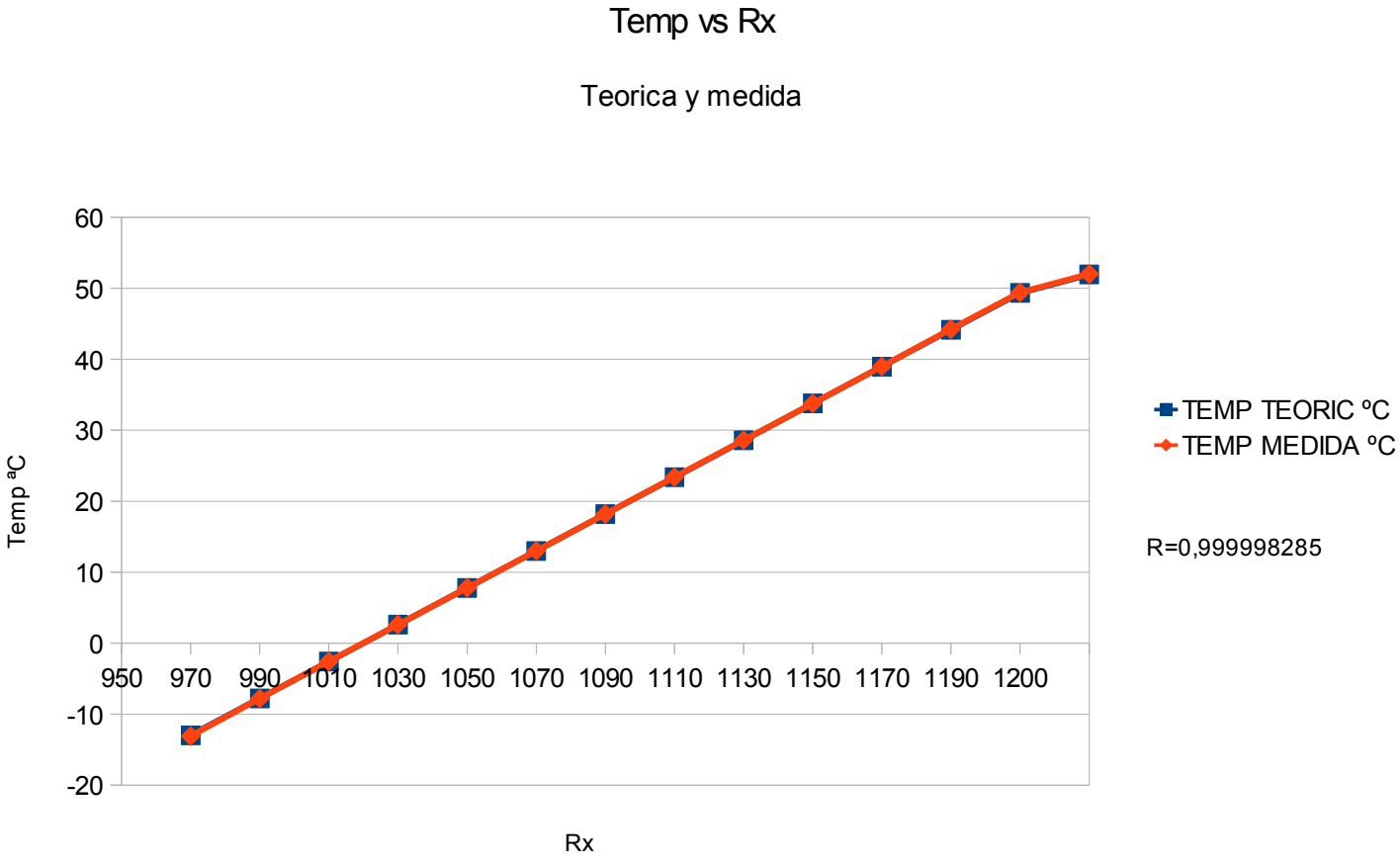

Fig.20. Gráfico temperatura y resistencia.

En el siguiente gráfico (Fig.21) se ve ampliada la diferencia en la medida de Rx entre el valor teórico y el valor medido. Se observa que en los extremos existe una diferencia mayor ya que como se ha comentado anteriormente estos valores equivalen a temperaturas extremas como -10ºC o 50ºC y la resistencia de calibración se encuentra en todo momento a unos 20ºC. Aún así esta diferencia no supera los 0.4 Ohmios y en los valores intermedios se mantiene por debajo de 0.1.

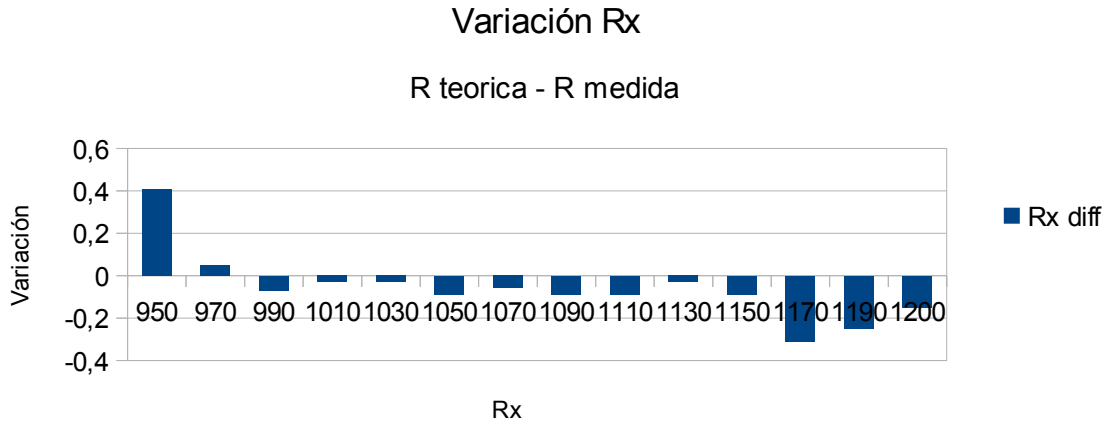

Fig.21. Gráfico diferencia R teórica y R medida.

En el siguiente gráfico (Fig.22) se ha comparado el valor medido por el multímetro con el medido por el programa. Se observa que en los valores intermedios la diferencia esta en el margen de 0.1Ω , esta diferencia puede tener varias causas. Al medir la resistencia con el multímetro digital los dígitos de la parte decimal fluctúan alrededor de 0.05 Ω ya que al medir resistencias tan pequeñas cualquier factor externo que afecte mínimamente a la resistencia varía su valor. También hay que tener en cuenta que el programa realiza cinco medidas antes de dar el resultado, que es el promedio de las cinco así que un error de 0.1 Ω es razonable teniendo en cuenta estos factores.

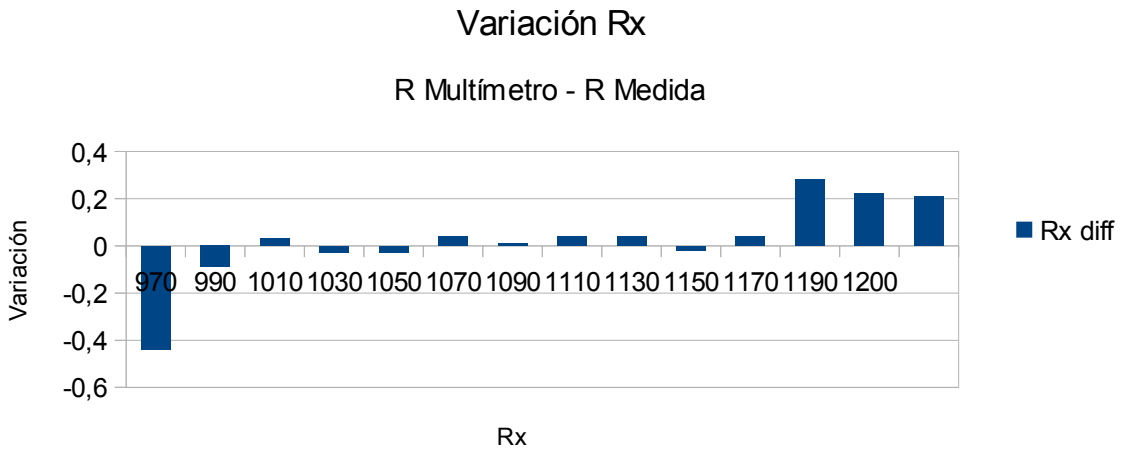

Fig.22. Gráfico diferencia R multímetro y R medida.

Otra prueba a la que ha sido sometido el sistema ha sido programar un intervalo de 5 segundos y dejarlo en funcionamiento haciendo variar la temperatura tocando el sensor para comprobar que registra esta variación. Después se ha leído la memoria y con los datos se ha elaborado esta gráfica donde se comprueba que efectivamente hay una subida de temperatura mientras se estaba calentando el sensor y al dejar de calentarlo vuelve a la temperatura ambiente.

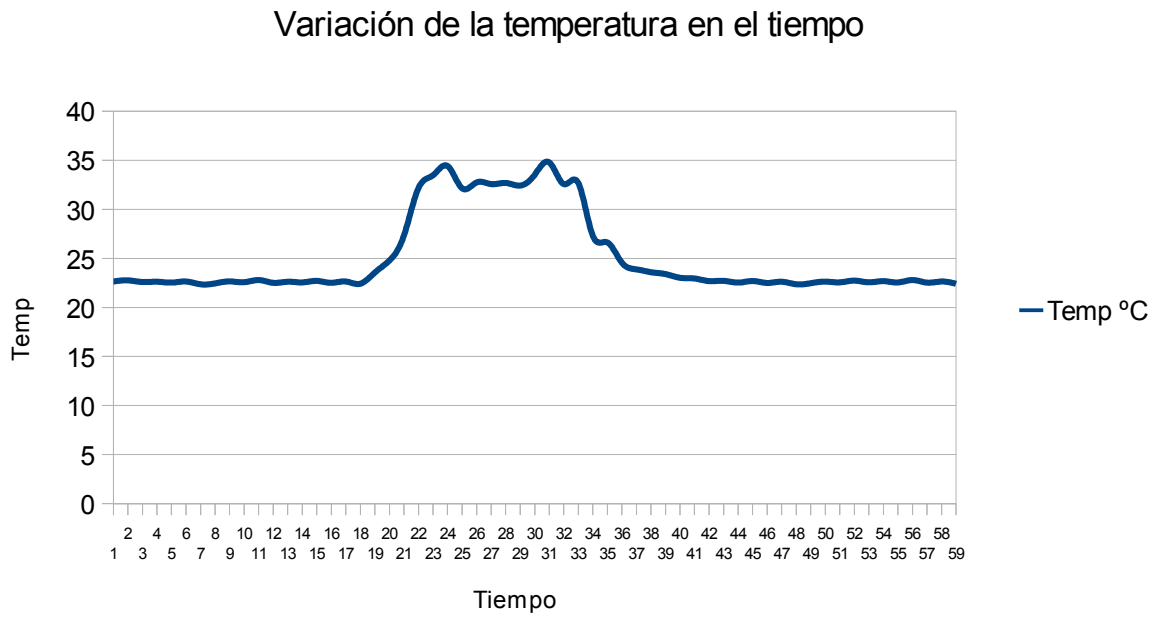

Fig.23. Gráfica variación de temperatura en el tiempo.

Esta es la tabla completa de los resultados utilizados para elaborar el último gráfico. Para obtenerlos se han leído los datos registrados en la memoria después de realizar las medidas.

Tabla 3: Datos leídos de la memoria.

Temp ºC	Hora	Fecha	Temp ºC	Hora	Fecha
22,64	11:25:33	27/03/13	34,78	11:28:03	27/03/13
22,76	11:25:38	27/03/13	32,57	11:28:08	27/03/13
22,59	11:25:43	27/03/13	32,68	11:28:13	27/03/13
22,62	11:25:48	27/03/13	27,3	11:28:18	27/03/13
22,53	11:25:53	27/03/13	26,63	11:28:23	27/03/13
22,64	11:25:58	27/03/13	24,56	11:28:28	27/03/13
22,36	11:26:03	27/03/13	23,88	11:28:33	27/03/13
22,45	11:26:08	27/03/13	23,59	11:28:38	27/03/13
22,65	11:26:13	27/03/13	23,4	11:28:43	27/03/13
22,57	11:26:18	27/03/13	23,02	11:28:48	27/03/13
22,79	11:26:23	27/03/13	22,95	11:28:53	27/03/13
22,51	11:26:28	27/03/13	22,68	11:28:58	27/03/13
22,62	11:26:33	27/03/13	22,71	11:29:03	27/03/13
22,55	11:26:38	27/03/13	22,53	11:29:08	27/03/13
22,7	11:26:43	27/03/13	22,69	11:29:13	27/03/13
22,51	11:26:48	27/03/13	22,5	11:29:18	27/03/13
22,64	11:26:53	27/03/13	22,62	11:29:23	27/03/13
22,42	11:26:58	27/03/13	22,36	11:29:28	27/03/13
23,56	11:27:03	27/03/13	22,47	11:29:33	27/03/13
24,78	11:27:08	27/03/13	22,63	11:29:38	27/03/13
27,34	11:27:13	27/03/13	22,56	11:29:43	27/03/13
32,17	11:27:18	27/03/13	22,73	11:29:48	27/03/13
33,47	11:27:23	27/03/13	22,56	11:29:53	27/03/13
34,42	11:27:28	27/03/13	22,68	11:29:58	27/03/13
32,15	11:27:33	27/03/13	22,55	11:30:03	27/03/13
32,74	11:27:38	27/03/13	22,79	11:30:08	27/03/13
32,57	11:27:43	27/03/13	22,53	11:30:13	27/03/13
32,69	11:27:48	27/03/13	22,64	11:30:18	27/03/13
32,42	11:27:53	27/03/13	22,42	11:30:23	27/03/13
33,53	11:27:58	27/03/13			

4.1 Características del sensor diseñado

Este es un resumen de las características del sensor diseñado, en comparación con el modelo de referencia usado. Para la elaboración de esta comparativa se han tenido en cuenta los resultados experimentales obtenidos. Los resultados señalan que se han cumplido los objetivos iniciales.

	Rango de temperatura	Precisión	Memoria	Intervalo
Modelo referencia[1]	-5ºC a 45ºC	±0.22ºC	15.9 millones de lecturas	De 0.5 a 15 segundos
Sensor diseñado	-10ºC a 50ºC	±0.1ºC	2.5 millones de lecturas	Totalmente ajustable

5- Conclusiones y trabajos futuros

A la vista de los resultados el sistema cumple con los requisitos de diseño iniciales. El coste de fabricación no es elevado y se consiguen unas mediciones aceptables. El inconveniente de este proyecto ha sido la dificultad de tener que diseñar desde cero la placa y programarla. El diseño se ha repetido dos veces debido a los fallos iniciales y a la hora de programar cuando el sistema no funciona correctamente se plantea la duda de si el fallo está en el hardware o en el software. Por este motivo no se ha podido avanzar más en el desarrollo del diseño que se podría haber extendido de varias formas.

El sistema incorpora un puerto USB que permite la comunicación con un PC, de esta forma se podrían descargar los datos de las mediciones y programar datos como el intervalo, hora y fecha. Sería necesario modificar el programa y crear un driver para el PC además de la aplicación que permitiera acceder al PIC. Lo ideal sería una aplicación que permitiera configurar el PIC y además pudiera presentar gráficamente los datos obtenidos.

En el apartado de la alimentación, sería necesario diseñar un sistema y realizar pruebas para comprobar el consumo del circuito para así dotarlo de la fuente de energía que requiera. Para ello el PIC se puede configurar en modo “deep sleep” o bajo consumo en el intervalo entre mediciones para reducir el uso de la batería.

Una vez completo el sistema se debería encapsular para poder ser usado en el mar, esto tal vez requeriría rediseñar la placa dependiendo de la ubicación final de los puertos, el sensor y la batería.

La prueba final sería, una vez realizado todo el proceso, dejarlo en el mar durante un tiempo para comprobar que realmente funciona y corregir los posibles fallos.

6- Bibliografía

[1] Microdaq.com (http://www.microdaq.com/data-logger/temperature/1.php)

[2] Improved Direct Interface Circuit for Resistive Full and Half-Bridge Sensors 2007 4th International Conference on Electrical and Electronics Engineering, México city, E. Sifuentes, O. Casas, F. Reverter, and R. Pallàs-Areny.

[3]Datasheet del PIC18F24J50, Microchip.com (http://ww1.microchip.com/downloads/en/DeviceDoc/39931d.pdf)

[4] Datasheet del conector USB, Farnell.com (http://www.farnell.com/datasheets/1650952.pdf)

[5] Datasheet de la memoria, Farnell.com (http://www.farnell.com/datasheets/1669503.pdf)

[6] Web del Pickit 3 (http://www.microchip.com/pickit3)

[7] Web Total Solutions (http://www.Totalsolutions.fluke.com)

[8] Guía de usuario del multímetro (http://projectlab.engphys.ubc.ca/wp-content/uploads/HP34401A-UsersGuide.pdf)

7- Anexo

A.1 Programa

```
#include <htc.h>              //librería genérica para el PIC18

void ini(void);               //Función de configuración de inicio
void carga_rc1 (void);        //Funciones de carga y descarga del condensador
void carga_rc2 (void);
void carga_rx (void);
void calcula_trc1(void);      //Funciones de cálculo de las temporizaciones
void calcula_trc2(void);
void calcula_tx(void);
void calculfinal(void);
void media(void);             //Función para el cálculo de la media de las medidas
void rtcc(void);              //Función para leer el valor de tiempo del RTCC
void guardar(void);           //Función para guardar las medidas en la memoria
void leermem (void);          //Función para leer la memoria
void intervala(void);         //Función que temporiza el intervalo entre medidas
void init();                  //Funciones para el manejo de la memoria flash por SPI
void Send_Byte(unsigned char out);
unsigned char Get_Byte();
void Poll_SO();
void CE_High();
void CE_Low();
unsigned char Read_Status_Register();
void send_ewsr();
void write_sr(unsigned char byte);
void send_wren();
void send_wrdi();
unsigned long Jedec_ID_Read();
unsigned char Read(unsigned long Dst);
void Byte_Program(unsigned long Dst, unsigned char byte);
void Chip_Erase();

//Variables

unsigned int n;
unsigned char any, mes, dia, hora, min, aux, cont,t, segdec;
unsigned char seg, diasem, control, retardo, m1,m2,m3,m4,m5,m6,m7,m8;
float dato, trc1, trc2, tx, rx, temp, temp2, temp3, timerc1, timerc2, timerx, m;
unsigned long dir;      //Variable dirección de memoria flash
unsigned long mem[825],me1,me2,me3,me4;

#define lecturamem    0   //Dependiendo de este valor el programa realiza medidas
                          //(0) o lee memoria (1)
```

```
#define intervaloseg      2             //intervalo en segundos
#define intervalomin      0             //intervalo en minutos
#define intervalohor      0             //intervalo en horas

#define ANY                     0x13    //Valores de inicio del Reloj en tiempo real RTCC
#define DIA                     0x21
#define MES                     0x01
#define HORA                    0x18
#define DIASEM                  0x01    //0=Domingo
#define SEG                     0x00
#define MIN                     0x33

#define SO          RB5                 //Puertos SPI para comunicación con la memoria
#define SI          RC7
#define SCK         RB4
#define CE          RB3

void main (void)
{
        ini();                          //inicializa
        do
        {

        carrega_rc1();
        while(control==0);      //Espera el final de la descarga
        timerc1=dato;           //Guarda el dato capturado por la interrupción en la variable
                                //correspondiente a la línea medida, en este caso Rc1

        carrega_rc2();
        while(control==0);      //espera final descarga
        timerc2=dato;

        carrega_rx();
        while(control==0);      //espera final descarga
        timerx=dato;

        calculfinal();          //calcula T(ºC) a partir de datos timers

        mitja();                        //promedio de 5 medidas

        }
        while(1);
}
void interrupt medida(void)
{

        if(TMR3IF==1)           //Si el Timer 3 se desborda se activa esta interrupción
        {
                TMR3IF=0;               //Se ha producido un error en la medida
```

```
                control=0;
        }
        if(INT0IF==1)           //Cuando se produce un flanco descendente en RB0
                                //se activa esta interrupción
        {
                INT0IF=0;                       //Se ha descargado el condensador
                dato=((TMR3H<<8)|TMR3L);        //Se guarda el valor de tiempo en una variable
                TMR3ON=0;                       //Deshabilita el Timer 3
                control=1;                  //Se activa la variable que controla la descarga
        }
        if(T0IF==1)    //Si el Timer 0 se desborda se activa esta interrupción
        {
                T0IF=0;
                retardo=retardo+1;          //Es el retardo que se usa para cargar el condensador
                TMR0=90;                    //Se vuelve a cargar el Timer 0 para temporizar
        }
}

void ini (void)
{
        __CONFIG2L=0X0D; //Selecciona el tipo de cristal HSPLL.

        __CONFIG3L=0X02; //Selecciona el oscilador que usa el RTCC como referencia, en este
                                caso, un oscilador externo conectado a T1OSC/T1CKI.

        ANCON1=0x1F
        ANCON0=0xFF; // Configura los puertos en modo digital.

        T1CON=0X89; //Selecciona la fuente de reloj del Timer 1, que es el que usa el RTCC
                        como referencia, un oscilador externo de cristal.

        INTCON=0xB0; // Habilita las interrupciones globales y del Timer 0.

        PIE1=0x01; // Habilita la interrupción del Timer 1.

        T0CON=0xC7; // Configura el Timer 0 que se usa para temporización de retardos.

        RBPU=1; // Habilita las resistencias de pull-up del puerto B.

        INTEDG0=0; // Configura el disparo de la interrupción RB0 en un flanco descendente.

        T3CON=0x00; // Configura el Timer 3 que se usa para temporizar el tiempo de descarga
                                del condensador.

        TMR3IE=1; // Habilita la interrupción del Timer 3.

        TRISA=0xFF; // Configura el puerto A como entrada de alta impedancia.

        TRISB=0xE7; // Configura el puerto B como entrada excepto CE y SCK (RB4,RB5).
```

```
	TRISC=0x7F; // Configura el puerto puerto C como entrada excepto SI (RC7).
	CE=1;
	SCK=0;
	TMR0ON=1;
	TMR0=100;
	retardo=0;			//programa retardo
	n=5;
	m=0;
	RTCWREN=1;
	RTCEN=1;
	RTCCFG=0xA7;			//rtcoe=1
	GIE=0;
	EECON2=0X55;			//Configuración RTCC
	EECON2=0XAA;
	RTCWREN=1;
	RTCEN=1;
	GIE=1;
	RTCCFG=0xA3;
	RTCVALL=ANY;				//año
	RTCVALH=0;
	RTCVALL=DIA;				//dia
	RTCVALH=MES;				//mes
	RTCVALL=HORA;				//hora
	RTCVALH=DIASEM;				//dia de la semana 0=Domingo
	RTCVALL=SEG;				//segundos
	RTCVALH=MIN;				//minutos
	GIE=0;
	EECON2=0X55;
	EECON2=0XAA;
	RTCWREN=0;
	GIE=1;
	T0CON=0xC7;				//C7 INTEDG0=1 flanco ascendente x inversor prescaler 7
	TMR0=10;
	SCK=0;
	send_wrdi();   /* Deshabilita previo*/
	SCK=0;
	send_ewsr();   /* Habilita escritura de registro */

	SCK=0;
write_sr(0x00);      /* Desbloquea registro */

	SCK=0;
	send_wren();            /* Habilita escritura */

	if(lecturamem==1)leermem();
	SCK=0;
	Chip_Erase(); /* Borrar memoria */

	dir=0x000005; //dirección de inicio de escritura en memoria
```

```
}

void carrega_rc1 (void)
{

        INT0IE=0;           //Deshabilita la interrupción de cambio de nivel en RB0
        control=0;          //Es la variable que controla la descarga
        TRISA=0xFE;         //puerto A entrada (HZ) menos RA0 salida carga/descarga
        RA0=1;              //Se empieza a cargar el condensador

        while(retardo!=2);
        retardo=0;          //Mantiene el condensador cargado

        TMR3ON=1;           //Se activa el Timer 3
        TMR3H=0;            //Timer 3 a 0 para temporizar
        TMR3L=0;            //
        INT0IE=1;           //Habilita la interrupción de cambio de nivel en RB0
        RA0=0;              //Empieza la descarga

}
void carrega_rc2 (void)
{
        INT0IE=0;
        control=0;
        TRISA=0xFE;         //puerto A entrada (HZ) menos RA0 salida carga/descarga
        RA0=1;                      //aguanta carga

        while(retardo!=2);
        retardo=0;

        TRISA=0xFB;                 //puerto A entrada (HZ) menos RA2 salida carga/descarga
        RA2=1;                      //aguanta carga
        TMR3ON=1;           //activar timer
        TMR3H=0;            //timer a 0 para temporizar
        TMR3L=0;            //timer a 0 para temporizar
        INT0IE=1;
        RA2=0;                      //empieza descarga


}
void carrega_rx (void)
{
        INT0IE=0;
        control=0;
        TRISA=0xFE;         //puerto A entrada (HZ) menos RA0 salida carga
        RA0=1;                      //aguanta carga

        while(retardo!=2);
        retardo=0;
```

```
	TRISA=0xFD;		//puerto A entrada (HZ) menos RA1 salida carga/descarga
	RA1=1;			//aguanta carga
	TMR3ON=1;	//activar timer
	TMR3H=0;	//timer a 0 para temporizar
	TMR3L=0;	//timer a 0 para temporizar
	INT0IE=1;
	RA1=0;			//empieza descarga

}

void calculfinal(void)
{
	rx=timerx-timerc1;
	rx=(rx/(timerc2-timerc1))*1199;
	rx=rx-(rx*0.0229);  //rc2


}

void mitja(void)
{
	n=n-1;
	m=m+rx;	//se guarda en m el total del sumatorio rx
	if(n==0)
	{
		rx=m/5;			//rx ahora es la media n=5
		temp=((rx-1000)/3.85);	//alpha=3.85
		n=5;			//vuelvo a contar desde 5
		m=0;			//y el sumatorio a 0
		rtcc();			//carga los valores del reloj
		intervala();		//guarda en memoria y espera hasta la próxima medida

	}
 }
void rtcc(void)
{
	RTCCFG=0xA7;
	any=RTCVALL;		//año
	RTCVALH=0;
	dia=RTCVALL;			//dia
	mes=RTCVALH;			//mes
	hora=RTCVALL;			//hora
	diasem=RTCVALH;			//weekday
	seg=RTCVALL;			//segundos
	min=RTCVALH;			//minutos

}
void guardar(void)
{
	unsigned int tempint,tempdec;
```

```
        temp3=temp;
        tempint=temp3;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,tempint);   //Enviar valor T(ºC)entero
        dir++;
    temp3=temp3-tempint+0.004;
        tempdec=(temp3*100);                //Separar entero y decimal redondeo
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,tempdec); //Enviar valor T(ºC)decimal
    dir++;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,hora);                 //Enviar valor hora
    dir++;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,min);              //Enviar valor min
    dir++;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,seg);                  //Enviar valor seg
    dir++;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,dia);                  //Enviar valor dia
    dir++;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,mes);                  //Enviar valor mes
    dir++;
        SCK=0;
        send_wren();            /* Habilitar escritura */
        SCK=0;
        Byte_Program(dir,any);                  //Enviar valor año
        dir++;
    if(dir>150)
        {
            SCK=0;
            m1=Read(0x00000D);
            SCK=0;
            m2=Read(0x00000E);
```

```
                SCK=0;
                m3=Read(0x00000F);
                SCK=0;
                m4=Read(0x000010);
                SCK=0;
                m5=Read(0x000011);
                SCK=0;
                m6=Read(0x000012);
                SCK=0;
                m7=Read(0x000013);
                SCK=0;
                m8=Read(0x000014);
                SCK=0;
                CE=1;
        }
}
void leermem(void)
{
                n=0;
        dir=5;
                for(n=0;n<825;n++)
                {
                SCK=0;
                me1=Read(dir);
                dir++;
                SCK=0;
                me2=Read(dir);
                dir++;
                SCK=0;
                me3=Read(dir);
                dir++;
                SCK=0;
                me4=Read(dir);
                dir++;
                mem[n]=( (me1<<24)|(me2<<16)|(me3<<8)|(me4));
                }
                CE=1;

}

void intervala (void)
{

cont=0;                 //variable para contabilizar el paso del tiempo
        rtcc();                 //lee el valor de tiempo actual y lo guarda en hor, min y seg.
        aux=seg;                //en aux queda guardado el valor de segundos actual
        do                      //Bucle que lee el valor de tiempo
        {
                rtcc();
                if(aux!=seg)    //Cuando cambia el valor de los segundos
```

```
		{
			cont++;		//se incrementa la variable para contabilizar
			aux++;		//y la auxiliar para comparar de nuevo
		}
	}while(cont<intervaloseg);	//este bucle se repite hasta que se contabiliza el intervalo
		guardar();		//deseado, luego se guarda la ultima lectura en memoria
}



/*************************************************************************/
/* PROCEDURE: init                                                       */
/*                                                                       */
/* This procedure initializes the SCK to low. Must be called prior to    */
/* setting up mode 0.                                                    */
/*                                                                       */
/* Input:                                                                */
/*		None                                                     */
/*                                                                       */
/* Output:                                                               */
/*		SCK                                                      */
/*************************************************************************/
void init()
{
	SCK = 0;	/* set clock to low initial state */
}

/*************************************************************************/
/* PROCEDURE: Send_Byte                                                  */
/*                                                                       */
/* This procedure outputs a byte shifting out 1-bit per clock rising     */
/* edge on the the SI pin(LSB 1st).                                      */
/*                                                                       */
/* Input:                                                                */
/*		out                                                      */
/*                                                                       */
/* Output:                                                               */
/*		SI                                                       */
/*************************************************************************/
void Send_Byte(unsigned char out)
{

	unsigned char i = 0;
	for (i = 0; i < 8; i++)
	{
		if ((out & 0x80) == 0x80)	/* check if MSB is high */
			SI = 1;
		else
			SI = 0;			/* if not, set to low */
		SCK = 1;			/* toggle clock high */
```

```
            out = (out << 1);           /* shift 1 place for next bit */
            SCK = 0;                    /* toggle clock low */
        }
}

/*****************************************************************************/
/* PROCEDURE: Get_Byte                                                       */
/*                                                                           */
/* This procedure inputs a byte shifting in 1-bit per clock falling          */
/* edge on the SO pin(LSB 1st).                                              */
/*                                                                           */
/* Input:                                                                    */
/*             SO                                                            */
/*                                                                           */
/* Output:                                                                   */
/*             None                                                          */
/*****************************************************************************/
unsigned char Get_Byte()
{
        unsigned char i = 0, in = 0, temp = 0;
        for (i = 0; i < 8; i++)
        {
            in = (in << 1);         /* shift 1 place to the left or shift in 0 */
            temp = SO;              /* save input */
            SCK = 1;                /* toggle clock high */
        if (temp == 1)              /* check to see if bit is high */
            in = in | 0x01;         /* if high, make bit high */
            SCK = 0;                /* toggle clock low */

        }
        return in;
}

/*****************************************************************************/
/* PROCEDURE: CE_High                                                        */
/*                                                                           */
/* This procedure set CE = High.                                             */
/*                                                                           */
/* Input:                                                                    */
/*             None                                                          */
/*                                                                           */
/* Output:                                                                   */
/*             CE                                                            */
/*                                                                           */
/*****************************************************************************/
void CE_High()
{
        CE = 1;                         /* set CE high */
}
```

```
/*********************************************************************/
/* PROCEDURE: CE_Low                                                 */
/*                                                                   */
/* This procedure drives the CE of the device to low.                */
/*                                                                   */
/* Input:                                                            */
/*              None                                                 */
/*                                                                   */
/* Output:                                                           */
/*              CE                                                   */
/*                                                                   */
/*********************************************************************/
void CE_Low()
{
        CE = 0;                         /* clear CE low */
}

/*********************************************************************/
/* PROCEDURE: Read_Status_Register                                   */
/*                                                                   */
/* This procedure read the status register and returns the byte.     */
/*                                                                   */
/* Input:                                                            */
/*              None                                                 */
/*                                                                   */
/* Returns:                                                          */
/*              byte                                                 */
/*********************************************************************/
unsigned char Read_Status_Register()
{
        unsigned char byte = 0;
        CE_Low();               /* enable device */
        Send_Byte(0x05);        /* send RDSR command */
        byte = Get_Byte();      /* receive byte */
        CE_High();              /* disable device */
        return byte;
}

/*********************************************************************/
/* PROCEDURE: EWSR                                                   */
/*                                                                   */
/* This procedure Enables Write Status Register.                     */
/*                                                                   */
/* Input:                                                            */
/*              None                                                 */
/*                                                                   */
/* Returns:                                                          */
/*              Nothing                                              */
/*********************************************************************/
void send_ewsr()
```

```
{
        CE_Low();               /* enable device */
        Send_Byte(0x50);        /* enable writing to the status register */
        CE_High();              /* disable device */
}


/*****************************************************************************/
/* PROCEDURE: WRSR                                                           */
/*                                                                           */
/* This procedure writes to the Status Register.                             */
/*                                                                           */
/* Input:                                                                    */
/*              byte                                                         */
/*                                                                           */
/* Returns:                                                                  */
/*              Nothing                                                      */
/*****************************************************************************/
void write_sr(unsigned char byte)
{

        CE_Low();               /* enable device */
        Send_Byte(0x01);        /* select write to status register */
        Send_Byte(byte);
        CE_High();              /* disable the device */
}

/*****************************************************************************/
/* PROCEDURE: WREN                                                           */
/*                                                                           */
/* This procedure enables the Write Enable Latch.  It can also be used       */
/* to Enables Write Status Register.                                         */
/*                                                                           */
/* Input:                                                                    */
/*              None                                                         */
/*                                                                           */
/* Returns:                                                                  */
/*              Nothing                                                      */
/*****************************************************************************/
void send_wren()
{
        CE_Low();               /* enable device */
        Send_Byte(0x06);        /* send WREN command */
        CE_High();              /* disable device */
}

/*****************************************************************************/
/* PROCEDURE: WRDI                                                           */
/*                                                                           */
/* This procedure disables the Write Enable Latch.                           */
```

```
/*                                                                    */
/* Input:                                                             */
/*               None                                                 */
/*                                                                    */
/* Returns:                                                           */
/*               Nothing                                              */
/**********************************************************************/
void send_wrdi()
{
        CE_Low();               /* enable device */
        Send_Byte(0x04);        /* send WRDI command */
        CE_High();              /* disable device */
}

/**********************************************************************/
/* PROCEDURE: Jedec_ID_Read                                           */
/*                                                                    */
/* This procedure Reads the manufacturer's ID (BFh), memory type (25h)  */
/* and device ID (05h).  It will use 9Fh as the JEDEC ID command.     */
/* Please see the product datasheet for details.                      */
/*                                                                    */
/* Input:                                                             */
/*               None                                                 */
/*                                                                    */
/* Returns:                                                           */
/*               IDs_Read:ID1(Manufacture's ID = BFh, Memory Type (25h), */
/*                and Device ID (05h)                                 */
/*                                                                    */
/**********************************************************************/
unsigned long Jedec_ID_Read()
{
        unsigned long temp;

        temp = 0;

        CE_Low();               /* enable device */
        Send_Byte(0x9F);        /* send JEDEC ID command (9Fh) */
        temp = (temp | Get_Byte()) << 8; /* receive byte */
        temp = (temp | Get_Byte()) << 8;
        temp = (temp | Get_Byte());   /* temp value = 0xBF2505 */
        CE_High();              /* disable device */

        return temp;
}

/**********************************************************************/
/* PROCEDURE:      Read                                               */
/*                                                                    */
/* This procedure reads one address of the device.  It will return the  */
/* byte read in variable byte.                                        */
```

```
/*                                                                      */
/*                                                                      */
/*                                                                      */
/* Input:                                                               */
/*              Dst:    Destination Address 000000H - fffffH            */
/*                                                              */
/*                                                                      */
/* Returns:                                                             */
/*              byte                                                    */
/*                                                                      */
/************************************************************************/
unsigned char Read(unsigned long Dst)
{
        unsigned char byte = 0;

        CE_Low();               /* enable device */
        Send_Byte(0x03);        /* read command */
        Send_Byte(((Dst & 0xFFFFFF) >> 16));    /* send 3 address bytes */
        Send_Byte(((Dst & 0xFFFF) >> 8));
        Send_Byte(Dst & 0xFF);
        byte = Get_Byte();
        CE_High();              /* disable device */
        return byte;            /* return one byte read */
}


/************************************************************************/
/* PROCEDURE:    Byte_Program                                           */
/*                                                              */
/* This procedure programs one address of the device.                   */
/* Assumption:  Address being programmed is already erased and is NOT */
/*              block protected.                                        */
/*                                                                      */
/*                                                                      */
/*                                                                      */
/* Input:                                                               */
/*              Dst:            Destination Address 000000H - fffffH    */
/*              byte:           byte to be programmed                   */
/*                                                              */
/*                                                                      */
/* Returns:                                                             */
/*              Nothing                                                 */
/*                                                                      */
/************************************************************************/
void Byte_Program(unsigned long Dst, unsigned char byte)
{
        CE_Low();                       /* enable device */
        Send_Byte(0x02);                /* send Byte Program command */
        Send_Byte(((Dst & 0xFFFFFF) >> 16));    /* send 3 address bytes */
        Send_Byte(((Dst & 0xFFFF) >> 8));
```

```
        Send_Byte(Dst & 0xFF);
        Send_Byte(byte);                /* send byte to be programmed */
        CE_High();                      /* disable device */
}

/*********************************************************************/
/* PROCEDURE: Chip_Erase                                             */
/*                                                                   */
/* This procedure erases the entire Chip.                            */
/*                                                                   */
/* Input:                                                            */
/*              None                                                 */
/*                                                                   */
/* Returns:                                                          */
/*              Nothing                                              */
/*********************************************************************/
void Chip_Erase()
{
        CE_Low();                       /* enable device */
        Send_Byte(0x60);                /* send Chip Erase command (60h or C7h) */
        CE_High();                      /* disable device */
}
```

A.2 Registros de PIC18f24j50

REGISTER 27-3: CONFIG2L: CONFIGURATION REGISTER 2 LOW (BYTE ADDRESS 300002h)

R/WO-1	R/WO-1	U-0	R/WO-1	R/WO-1	R/WO-1	R/WO-1	R/WO-1
IESO	FCMEN	—	LPT1OSC	T1DIG	FOSC2	FOSC1	FOSC0
bit 7							bit 0

Legend:			
R = Readable bit	WO = Write-Once bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **IESO:** Two-Speed Start-up (Internal/External Oscillator Switchover) Control bit
1 = Two-Speed Start-up is enabled
0 = Two-Speed Start-up is disabled

bit 6 **FCMEN:** Fail-Safe Clock Monitor Enable bit
1 = Fail-Safe Clock Monitor is enabled
0 = Fail-Safe Clock Monitor is disabled

bit 5 **Unimplemented:** Read as '0'

bit 4 **LPT1OSC:** Low-Power Timer1 Oscillator Enable bit
1 = Timer1 oscillator is configured for high-power operation
0 = Timer1 oscillator is configured for low-power operation

bit 3 **T1DIG:** Secondary Clock Source T1OSCEN Enforcement bit
1 = Secondary oscillator clock source may be selected (OSCCON<1:0> = 01) regardless of the (T1CON<3>) T1OSCEN state
0 = Secondary oscillator clock source may not be selected unless T1CON<3> = 1

bit 2-0 **FOSC<2:0>:** Oscillator Selection bits
111 = ECPLL oscillator with PLL software controlled, CLKO on RA6
110 = EC oscillator with CLKO on RA6
101 = HSPLL oscillator with PLL software controlled
100 = HS oscillator
011 = INTOSCPLLO, internal oscillator with PLL software controlled, CLKO on RA6, port function on RA7
010 = INTOSCPLL, internal oscillator with PLL software controlled, port function on RA6 and RA7
001 = INTOSCO internal oscillator block (INTRC/INTOSC) with CLKO on RA6, port function on RA7
000 = INTOSC internal oscillator block (INTRC/INTOSC), port function on RA6 and RA7

REGISTER 27-5: CONFIG3L: CONFIGURATION REGISTER 3 LOW (BYTE ADDRESS 300004h)

R/WO-1	R/WO-1	R/WO-1	R/WO-1	R/WO-1	R/WO-1	R/WO-1	R/WO-1
DSWDTPS3(1)	DSWDTPS2(1)	DSWDTPS1(1)	DSWDTPS0(1)	DSWDTEN(1)	DSBOREN	RTCOSC	DSWDTOSC(1)
bit 7							bit 0

Legend:			
R = Readable bit	WO = Write-Once bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-4 **DSWDTPS<3:0>**: Deep Sleep Watchdog Timer Postscale Select bits(1)

The DSWDT prescaler is 32. This creates an approximate base time unit of 1 ms.

1111 = 1:2,147,483,648 (25.7 days)
1110 = 1:536,870,912 (6.4 days)
1101 = 1:134,217,728 (38.5 hours)
1100 = 1:33,554,432 (9.6 hours)
1011 = 1:8,388,608 (2.4 hours)
1010 = 1:2,097,152 (36 minutes)
1001 = 1:524,288 (9 minutes)
1000 = 1:131,072 (135 seconds)
0111 = 1:32,768 (34 seconds)
0110 = 1:8,192 (8.5 seconds)
0101 = 1:2,048 (2.1 seconds)
0100 = 1:512 (528 ms)
0011 = 1:128 (132 ms)
0010 = 1:32 (33 ms)
0001 = 1:8 (8.3 ms)
0000 = 1:2 (2.1 ms)

bit 3 **DSWDTEN**: Deep Sleep Watchdog Timer Enable bit(1)

1 = DSWDT is enabled
0 = DSWDT is disabled

bit 2 **DSBOREN**: "F" Device Deep Sleep BOR Enable bit, "LF" Device VDD BOR Enable bit

For "F" Devices:
1 = VDD sensing BOR is enabled in Deep Sleep
0 = VDD sensing BOR circuit is always disabled

For "LF" Devices:
1 = VDD sensing BOR circuit is always enabled
0 = VDD sensing BOR circuit is always disabled

bit 1 **RTCOSC**: RTCC Reference Clock Select bit

1 = RTCC uses T1OSC/T1CKI as reference clock
0 = RTCC uses INTRC as reference clock

bit 0 **DSWDTOSC**: DSWDT Reference Clock Select bit(1)

1 = DSWDT uses INTRC as reference clock
0 = DSWDT uses T1OSC/T1CKI as reference clock

Note 1: These functions are not available on "LF" devices.

REGISTER 21-3: ANCON0: A/D PORT CONFIGURATION REGISTER 2 (BANKED F48h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG7(1)	PCFG6(1)	PCFG5(1)	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **PCFG<7:0>:** Analog Port Configuration bits (AN7-AN0)
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input is disabled and reads '0'

Note 1: These bits are only available on 44-pin devices.

REGISTER 21-4: ANCON1: A/D PORT CONFIGURATION REGISTER 1 (BANKED F49h)

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
VBGEN	r	—	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
bit 7							bit 0

Legend:	r = Reserved bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **VBGEN:** 1.2V Band Gap Reference Enable bit
1 = 1.2V band gap reference is powered on
0 = 1.2V band gap reference is turned off to save power (if no other modules are requesting it)

bit 6 **Reserved:** Always maintain as '0' for lowest power consumption

bit 5 **Unimplemented:** Read as '0'

bit 4-0 **PCFG<12:8>:** Analog Port Configuration bits (AN12-AN8)
1 = Pin configured as a digital port
0 = Pin configured as an analog channel – digital input is disabled and reads '0'

REGISTER 13-1: T1CON: TIMER1 CONTROL REGISTER (ACCESS FCDh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR1CS1	TMR1CS0	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	RD16	TMR1ON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 7-6 **TMR1CS<1:0>:** Timer1 Clock Source Select bits
10 = Timer1 clock source is the T1OSC or T1CKI pin
01 = Timer1 clock source is the system clock (FOSC)(1)
00 = Timer1 clock source is the instruction clock (FOSC/4)

bit 5-4 **T1CKPS<1:0>:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Source Select bit
When TMR1CS<1:0> = 10:
1 = Power up the Timer1 crystal driver and supply the Timer1 clock from the crystal output
0 = Timer1 crystal driver is off, Timer1 clock is from the T1CKI input pin(2)
When TMR1CS<1:0> = 0x:
1 = Power up the Timer1 crystal driver
0 = Timer1 crystal driver is off(2)

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
TMR1CS<1:0> = 10:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
TMR1CS<1:0> = 0x:
This bit is ignored. Timer1 uses the internal clock when TMR1CS<1:0> = 0x.

Note 1: The FOSC clock source should not be selected if the timer will be used with the ECCP capture/compare features.

2: The Timer1 oscillator crystal driver is powered whenever T1OSCEN (T1CON) or T3OSCEN (T3CON) = 1. The circuit is enabled by the logical OR of these two bits. When disabled, the inverter and feedback resistor are disabled to eliminate power drain. The TMR1ON and TMR3ON bits do not have to be enabled to power up the crystal driver.

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER (ACCESS FF2h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE/GIEH**: Global Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts

When IPEN = 1:
1 = Enables all high-priority interrupts (also enables low-priority interrupts when GIEL is also set)
0 = Disables all interrupts

bit 6 **PEIE/GIEL**: Peripheral/Low-Priority Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked peripheral interrupts (when GIE is also set)
0 = Disables all peripheral interrupts

When IPEN = 1:
1 = Enables all interrupts configured for low priority (when GIEH is also set)
0 = Disables all interrupts configured for low priority

bit 5 **TMR0IE**: TMR0 Overflow Interrupt Enable bit

1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE**: INT0 External Interrupt Enable bit

1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit

1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF**: TMR0 Overflow Interrupt Flag bit

1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF**: INT0 External Interrupt Flag bit

1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit(1)

1 = At least one of the RB<7:4> pins changed state (must be cleared in software)
0 = None of the RB<7:4> pins have changed state

Note 1: A mismatch condition will continue to set this bit. Reading PORTB and waiting 1 TCY will end the mismatch condition and allow the bit to be cleared.

REGISTER 9-7: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1 (ACCESS F9Dh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PMPIE(1)	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 7 **PMPIE:** Parallel Master Port Read/Write Interrupt Enable bit(1)
1 = Enables the PMP read/write interrupt
0 = Disables the PMP read/write interrupt

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RC1IE:** EUSART1 Receive Interrupt Enable bit
1 = Enables the EUSART1 receive interrupt
0 = Disables the EUSART1 receive interrupt

bit 4 **TX1IE:** EUSART1 Transmit Interrupt Enable bit
1 = Enables the EUSART1 transmit interrupt
0 = Disables the EUSART1 transmit interrupt

bit 3 **SSP1IE:** Master Synchronous Serial Port 1 Interrupt Enable bit
1 = Enables the MSSP1 interrupt
0 = Disables the MSSP1 interrupt

bit 2 **CCP1IE:** ECCP1 Interrupt Enable bit
1 = Enables the ECCP1 interrupt
0 = Disables the ECCP1 interrupt

bit 1 **TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE:** TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: These bits are unimplemented on 28-pin devices.

REGISTER 12-1: T0CON: TIMER0 CONTROL REGISTER (ACCESS FD5h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin input edge (set by T0SE)
0 = Internal clock (Fosc/4)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is not assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS<2:0>**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

REGISTER 15-1: T3CON: TIMER3 CONTROL REGISTER (ACCESS F79h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TMR3CS1	TMR3CS0	T3CKPS1	T3CKPS0	T3OSCEN	$\overline{\text{T3SYNC}}$	RD16	TMR3ON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 7-6 **TMR3CS<1:0>:** Timer3 Clock Source Select bits
10 = Timer3 clock source is the Timer1 oscillator or the T3CKI digital input pin (assigned in PPS module)
01 = Timer3 clock source is the system clock (Fosc)(1)
00 = Timer3 clock source is the instruction clock (Fosc/4)

bit 5-4 **T3CKPS<1:0>**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T3OSCEN:** Timer3 Oscillator Source Select bit
When TMR3CS<1:0> = 10:
1 = Power up the Timer1 crystal driver (T1OSC) and supply the Timer3 clock from the crystal output
0 = Timer1 crystal driver is off, Timer3 clock is from the T3CKI digital input pin assigned in PPS module(2)
When TMR3CS<1:0> = 0x:
1 = Power up the Timer1 crystal driver (T1OSC)
0 = Timer1 crystal driver is off(2)

bit 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
When TMR3CS<1:0> = 10:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS<1:0> = 0x:
This bit is ignored; Timer3 uses the internal clock.

bit 1 **RD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

Note 1: The Fosc clock source should not be selected if the timer will be used with the ECCP capture/compare features.
2: The Timer1 oscillator crystal driver is powered whenever T1OSCEN (T1CON) or T3OSCEN (T3CON) = 1. The circuit is enabled by the logical OR of these two bits. When disabled, the inverter and feedback resistor are disabled to eliminate power drain. The TMR1ON and TMR3ON bits do not have to be enabled to power up the crystal driver.

“Medida de la temperatura con conexión directa sensor-microcontrolador para estación marina”