Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

ADQUISICIÓN Y MONITORIZACIÓN DE VARIABLES FÍSICAS CON UN MICROCONTROLADOR

Alumno/Alumna: Martín Contreras, Asier

Director/Directora (1): Oleagordia Aguirre, Iñigo Javier

Director/Directora (2): Fernández Rodríguez, Pablo

Curso: 2018-2019

Fecha: 17, 06, 2019

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Resumen

La programación de microcontroladores puede ofrecer un gran abanico de oportunidades, una de ellas, la analizada en este proyecto, consiste en diseños de circuitos eléctricos basados en sensores que miden diferentes variables físicas con diversos métodos de transmisión de datos. Para ello, se ha dado uso a transductores de varios tipos diferentes (temperatura, humedad…), los cuales han sido acoplados a sus correspondientes montajes. Para el funcionamiento de estos se ha utilizado un microcontrolador PIC18F87J11, y mediante diferentes protocolos de comunicación se consigue obtener los valores deseados de los transductores mencionados anteriormente. Estos protocolos son el I2C, SPI y la conversión analógico-digital. Los pequeños proyectos realizados podrían ser la base que dan pie a futuras aplicaciones mayores.

Gaur egun mikrokontroladoreek aukera asko zabaltzen dituzte. Proiektu honetan aukera horietako bat ikertu da: hainbat datu transmisio metodo erabiltzen dituzten eta ingurumeneko aldagai fisikoak aztertzen dituzten sentsoreekin zirkuitu elektrikoak sortzea. Horretarako, mota askotako transduktoreak (tenperaturakoak, hezetasunekoak…) erabili eta haiei dagozkien zirkuitoetara erantsi dira. Transduktore horiek erabili ahal izateko PIC18F87J11 mikrokontroladorea erabili da, haren bidez, nahi dugun baloreak lor ditzakegulako sentsoreetan, zenbait komunikazio protokolo tarteko. Protokolo horiek I2C, SPI eta konbertsio analogiko-ditala dira. Honakoa bezalako proiektu txikiek etorkizunean aplikazio handiagoetarako balio dezakete.

The programming of microcontrollers can offer a wide range of opportunities, one of them, the one analyzed in this project, consists of designs of electrical circuits based on sensors that measure different physical variables with different methods of data transmission. For this purpose, transducers of several different types (temperature, humidity...) have been used, which have been coupled to their corresponding assemblies. A PIC18F87J11 microcontroller has been used for their operation, and by means of different communication protocols it is possible to obtain the desired values of the transducers mentioned above. These protocols are I2C, SPI and analog-to-digital conversion. The small projects carried out could be the basis for future major applications.

INDICE

Tabla de figuras

Tabla de ecuaciones

Tablas

Tabla de anexos

1 INTRODUCCION

El objetivo de este proyecto de fin de grado consiste en la adquisición y procesamiento de datos mediante diferentes métodos a través de un microcontrolador.

El proyecto se basa en el montaje de varios circuitos que se encargan en capturar variables físicas del entorno y mediante el uso de un microcontrolador procesarlas y enviarlas al PC mediante USB.

El microcontrolador utilizado es el PIC18F87J11, de Microchip, montado en una tarjeta Explorer PICDEM, al cual se adjunta un ICD3, que se encarga de la depuración y la programación. Todo ello junto a un PC permitirá, a través del programa MPLAB X IDE, programar los diversos programas deseados y así cumplir con el objetivo del proyecto, que es la adquisición de variables físicas del entorno con los correctos montajes de los sensores.

Se podría separar el proyecto en dos partes, que serían la parte del Hardware y la del Software.

Hardware: Se referiría al montaje de los circuitos con sus correspondientes sensores, encargados de capturar las variables físicas del entorno.

Software: Consiste en la programación del microcontrolador con los correspondientes programas en el PC para la correcta adquisición y visualización de los datos.

Los datos adquiridos se visualizarán principalmente en la pantalla LCD del Explorer y en el PC.

2 CONTEXTO

El conocimiento de variables físicas se puede dar de diversos métodos, uno de ellos es a través de sensores. Dependiendo de lo que se desea medir existen diferentes dispositivos, cuya funcionalidad varia. En este proyecto se ha dado uso a un microcontrolador y un PC, que junto a los sensores nos permitirá recoger y procesar variables del entorno. Existen varios métodos para llevar a cabo la correcta transmisión, los utilizados en este trabajo son el método de transmisión serie I2C, SPI y la conversión analógico-digital. Dependiendo de las características de cada sensor se da uso a uno u otro método, siendo la base del proyecto la programación de cada uno de estos. El proyecto tiene el objetivo de tener aplicaciones útiles para posibles aplicaciones futuras, es decir, la ampliación de los montajes realizados, la adjunción de varios de estos sensores en un mismo circuito o incluso la ampliación del propio código programado podría ser de gran utilidad para programas automatizados mayores que recojan variables físicas que se deseen. Además de ello, se muestra la programación de los métodos mencionados anteriormente (I2C, SPI y ADC) que podrían ser utilizados en sensores de otro tipo que no se hayan utilizado en este proyecto.

3 OBJETIVOS

El objetivo del proyecto es la realización de circuitos de adquisición de datos del ambiente.

Se realizarán varios circuitos:

LCD: El primer programa a realizar será uno que nos permita el uso de la LCD, puesto que no es posible utilizarla directamente, se detallará mas adelante.

Temperatura: se encarará de medir la temperatura del ambiente mediante el sensor TCN75A, que envía los datos de temperatura mediante I2C.

Temperatura y humedad: mediante este montaje se podrá visualizar tanto la temperatura como la humedad del ambiente, mediante el sensor HIH6121-021-001, que también emplea I2C.

Presión: realizando un montaje con el sensor de presión MPX2050DP se podrá capturar la diferencia de presión ejercida entre dos tubos pertenecientes al sensor, el valor será recibido en tensión y mediante el convertidor ADC del PIC se podrá lograr el valor de la presión.

Acelerómetro: el sensor MMA8451 permite medir la posición de los ejes X, Y y Z de este, utilizando tecnología I2C.

Temperatura y temperatura y humedad: este montaje se ha realizado para demostrar que es posible utilizar más de un sensor en un mismo programa, puesto que cada sensor dispone de su propia dirección. Consiste en realizar la llamada a la dirección del sensor de la cual deseamos obtener el valor y este se encargará de enviar el valor correspondiente.

En este caso he realizado el montaje de los sensores de temperatura TCNA75A y del sensor de temperatura y humedad HIH6121-021-001.

Todos los transductores: por último, se ha realizado un programa que contenga todos los transductores en un mismo circuito.

Cada uno de los circuitos será descrito con mayor detalle más adelante.

4 BENEFICIOS

Los beneficios del proyecto se basan en tres partes:

Conocimiento de variables

Cada uno de los programas que se realizan en el proyecto permiten conocer variables del ambiente automáticamente, los programas están diseñados para ofrecer las variables que adquieran cada uno de ellos en la pantalla LCD de la tarjeta Explorer, donde se encuentra el PIC18F87J11.

I2C, SPI y ADC

El proyecto permite la adquisición de conocimientos y facilidades en estos tres campos, para así resultar mas sencilla la programación de otros dispositivos que se deseen utilizar que funcionen con alguno de estos métodos.

Futuros proyectos

El objetivo principal del proyecto es que los programas realizados sean utilizados en futuras aplicaciones mayores, en los que se podría dar uso a combinaciones de estos sensores que den lugar a infinitas utilidades, ya que, la captura de variables físicas es indispensable en infinidad de casos.

5 METODOLOGIA

Para la realización de este trabajo será necesario tener conocimiento sobre el entorno y las características de la familia de microcontroladores PIC18, incluyendo el estudio sobre el lenguaje de programación C para programar dicho microcontrolador.

5.1 SOFTWARE

Para la realización del Software serán necesarios:

- PC
- Microcontrolador PIC18F87J11
- Programador ICD3
- MPLAB X IDE

5.1.1 MPLAB X IDE

MPLAB X es un ambiente de desarrollo integrado (IDE) gratuito creado por Microchip Technology (Microchip, 1989)para dar soporte a la realización de proyectos basados en microcontroladores PIC.

Para la realización de proyectos se deberán seguir los siguientes pasos:

- Configuraciones iniciales, así como la elección del microcontrolador, el programa de depuración, nombre y ubicación del proyecto…
- Realizar el "config.h", consiste en la elección de algunas de las características disponibles del Microcontrolador, en nuestro caso lo mas importante es la elección del oscilador. Los diferentes osciladores se pueden ver en el datasheet del PIC18F87J11.

```
// CONFIG1L
#pragma config WDTEN = OFF      // Watchdog Timer Enable bit (WDT disabled (control is placed on SWDTEN bit))
#pragma config STVREN = OFF     // Stack Overflow/Underflow Reset Enable bit (Reset on stack overflow/underflow disabled)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode dis

// CONFIG1H
#pragma config CP0 = OFF        // Code Protection bit (Program memory is not code-protected)

// CONFIG2L
#pragma config FOSC = INTOSC    // Oscillator Selection bits (Internal oscillator, port function on RA6 and RA7 )
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = ON        // Two-Speed Start-up (Internal/External Oscillator Switchover) Control bit (Two-Speed Start-up e

// CONFIG2H
#pragma config WDTPS = 32768    // Watchdog Timer Postscaler Select bits (1:32768)

// CONFIG3L
#pragma config EASHFT = ON      // External Address Bus Shift Enable bit (Address shifting enabled, address on external bus is of
#pragma config MODE = MM        // External Memory Bus Configuration bits (Microcontroller mode - External bus disabled)
#pragma config BW = 16          // Data Bus Width Select bit (16-bit external bus mode)
#pragma config WAIT = OFF       // External Bus Wait Enable bit (Wait states on the external bus are disabled)

// CONFIG3H
#pragma config CCP2MX = DEFAULT // ECCP2 MUX bit (ECCP2/P2A is multiplexed with RC1)
#pragma config ECCPMX = DEFAULT // ECCPx MUX bit (ECCP1 outputs (P1B/P1C) are multiplexed with RE6 and RE5; ECCP3 outputs (P3B/P3
#pragma config PMPMX = DEFAULT  // PMP Pin Multiplex bit (PMP port pins connected to EMB (PORTD and PORTE))
#pragma config MSSPMSK = MSK7   // MSSP Address Masking Mode Select bit (7-Bit Address Masking mode enable)
```

Figura 1. Config.h.

- Realización del main, funciones… (programa principal). Aquí se debe programar lo que se quiere realizar.

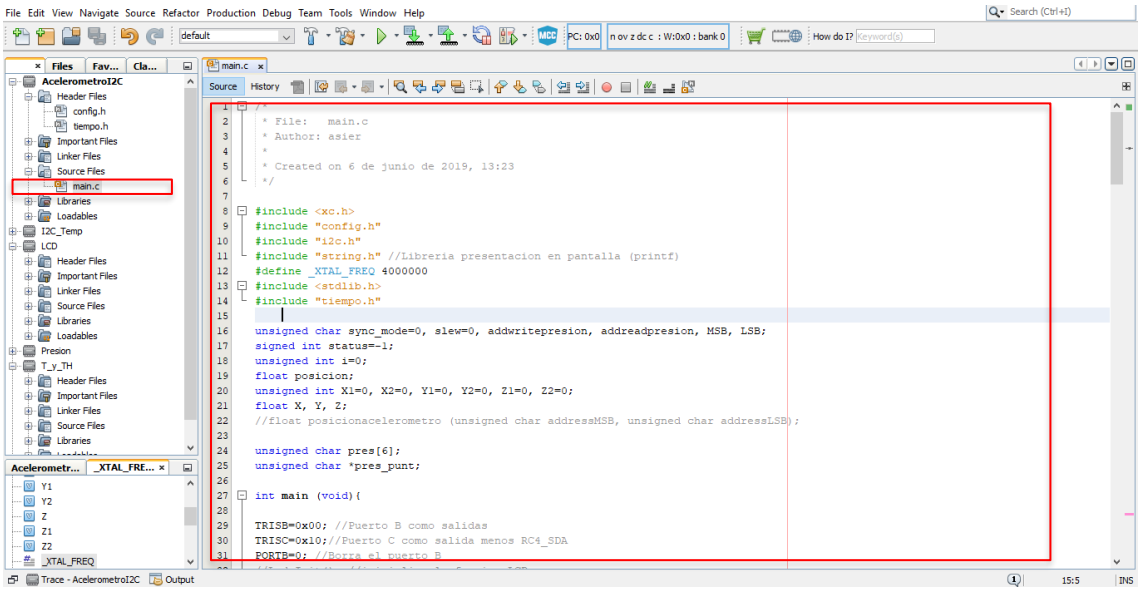


Figura 2. Main.

- El siguiente paso, una vez realizado el proyecto, sería la compilación de este para ver si tiene algún error y generar el archivo que la interfaz de programación necesita para programar el microcontrolador.
- Simulación: Este es un paso opcional. Lo que nos permite es simular la ejecución del programa paso a paso, tal y como lo haría el propio microcontrolador pero con varias ventajas, tales como consultar los registros de memoria, simular entradas externas, etc.
- Programación: Se trata del último paso antes de comenzar a utilizar el microcontrolador. En este apartado se necesitará una herramienta hardware externa. Se utilizará el ICD3, pero MPLAB permite el uso de distintas herramientas de programación que se muestran en el propio menú de programación al seleccionar la herramienta que se utilizará.
- El ICD3 se conecta por un extremo al PC vía USB y por el otro al microcontrolador con un conector RJ. Para la programación es necesario tener previamente conexionado el microcontrolador para que el ICD3 identifique el microcontrolador.

Figura 3. ICD3.

5.1.2 Compilador C XC8

El MPLAB X es un programa que utiliza lenguaje ensamblador para su funcionamiento. Para facilitar la programación existe el compilador XC8. Este compilador trata de convertir un código de programación en un formato legible para el programa. Nosotros escribiremos el programa en lenguaje C y después compilaremos el programa mediante XC8 para que este código sea legible por el programa. El XC8 también incluye unas librerías periféricas que tienen como objetivo facilitar la programación del código, estas librerías son sumamente útiles para este proyecto, ya que, ayudan a prevenir errores en la programación de los métodos utilizados (I2C, ADC y SPI).

5.1.3 I2C

I2C es un bus de comunicaciones en serie, su nombre viene de Inter-Integrated Circuit.

La principal característica de I^2C es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. Las líneas se llaman:

SDA: datos

SCL: reloj

Los dispositivos conectados al bus I2C tienen una dirección única para cada uno. El dispositivo maestro inicia la transferencia de datos y además genera la señal de reloj, en el caso de este proyecto el maestro siempre será el microcontrolador, aunque no siempre ha de serlo.

El proceso de comunicación en el bus I2C es:

- El maestro comienza la comunicación enviando un patrón llamado "start condition". Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transacción.
- El maestro se dirige al dispositivo con el que quiere comunicarse, enviando un byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con el que se quiere comunicar, y el octavo bit (A0) de menor peso se corresponde con la operación deseada ($R/\overline{W}$)
- La dirección enviada es comparada por cada esclavo del bus con su propia dirección, si ambas coinciden, el esclavo se considera direccionado como esclavo-transmisor o esclavo-receptor dependiendo del bit $R/\overline{W}$.
- Cada byte leído/escrito por el maestro debe ser obligatoriamente respondido con un bit de ACK por el dispositivo maestro/esclavo.
- Cuando la comunicación finaliza, el maestro transmite una "stop condition" para dejar libre el bus.

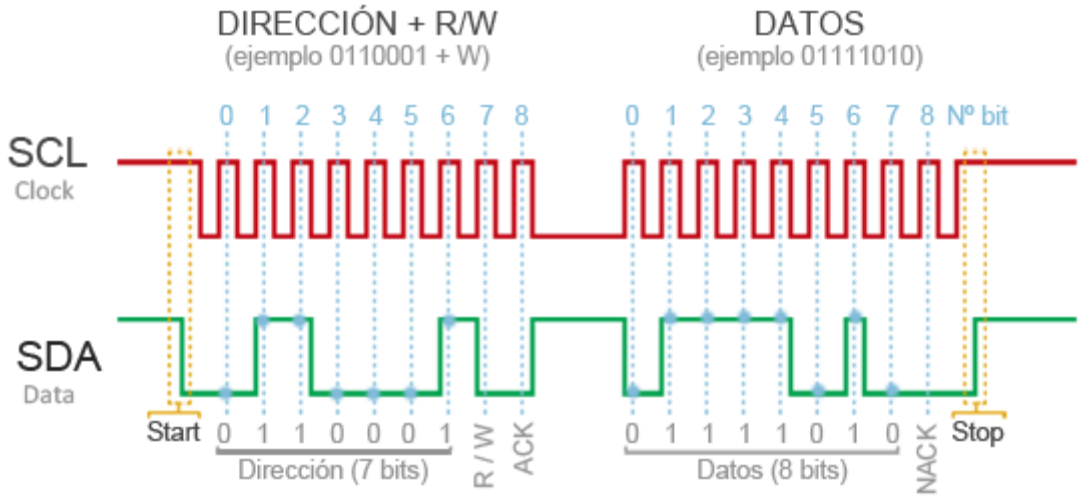


Figura 4. Funcionamiento del I2C (1).

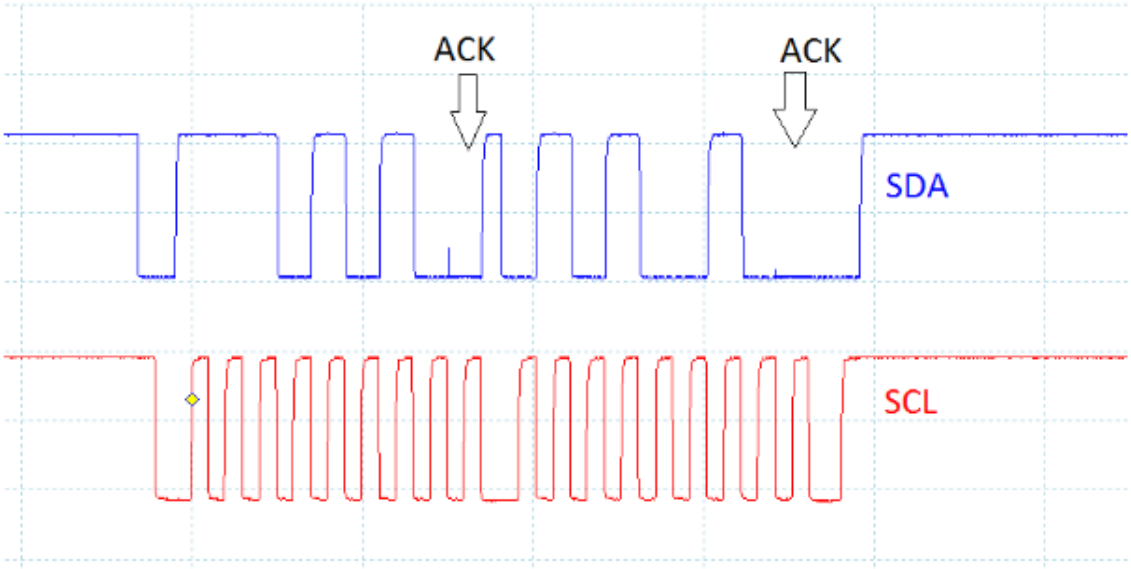


Figura 5. Funcionamiento del I2C (2).

Información obtenida del siguiente blog:

(Crespo, s.f.)

https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/

Tabla 1. Frecuencias del I2C.

FOSC	FCY	FCY * 2	BRG Value	FSCL (2 Rollovers of BRG)
40 MHz	10 MHz	20 MHz	18h	400 kHz(1)
40 MHz	10 MHz	20 MHz	1Fh	312.5 kHz
40 MHz	10 MHz	20 MHz	63h	100 kHz
16 MHz	4 MHz	8 MHz	09h	400 kHz(1)
16 MHz	4 MHz	8 MHz	0Ch	308 kHz
16 MHz	4 MHz	8 MHz	27h	100 kHz
4 MHz	1 MHz	2 MHz	02h	333 kHz(1)
4 MHz	1 MHz	2 MHz	09h	100 kHz
16 MHz	4 MHz	8 MHz	03h	1 MHz(1,2)

El I2C funciona a unas frecuencias determinadas, las cuales dependen del dispositivo que se esté utilizando, para ello, el datasheet del microcontrolador PIC18F87J11 (Bibliografía) nos proporciona la *Tabla 1.* Fijando un valor en SSPADD indicamos esta velocidad.

Una de las librerías periféricas que proporciona el compilador XC8 es la librería del I2C, de la cual se usarán las siguientes funciones:

- **OpenI2C:** Sirve para abrir el I2C y configurarlo. Para ello debemos incluir dos parámetros:

sync_mode: configura el registro SSPCON, debemos indicar en que modo queremos utilizar el I2C, en este caso será MASTER.

Slew: sirve para habilitar o deshabilitar el modo de rápido (fast-mode).

- **IdleI2C**: Crea una condición de espera.
- **StartI2C**: Comienza la comunicación I2C.
- **I2CWrite**: Escribe un byte en el bus I2C. Devuelve un valor que sirve para saber si la comunicación se ha completado, es decir, si se ha recibido el bit ACK.
- **GetsI2C**: Sirve para leer un array de bytes. Indicando el puntero y la longitud de este la función devuelve los X datos exigidos.
- **StopI2C**: Detiene la comunicación I2C.

5.1.4 ADC

La conversión analógica-digital consiste en la recepción de una señal de carácter analógica y su transformación en formato digital. En el caso de este trabajo sirve para convertir un valor de tensión en un valor digital, lo cual permite conocer el valor numérico mediante una correcta programación.

Librería ADC:

- **OpenADC:** Sirve para abrir y configurar el convertidor ADC.

Config1**:** Se indica la conversión del reloj A/D, el tiempo de adquisición del A/D y el formato del resultado.

Config2: Se indica el canal por el que se realizará la conexión analógica, para posteriormente realizar la conversión. También se indica si se desea la activación de las interrupciones y las referencias de tensión.

Portconfig: Se seleccionan los canales que se desea que sean analógicos.

- **CloseADC:** Sirve para cerrar el convertidor A/D.
- **ConvertADC:** Convierte los valores analógicos recibidos en valores digitales.

Para mas información sobre la conversión analógico-digital acuda al siguiente blog:

(Wikipedia, 2019)
https://es.wikipedia.org/wiki/Conversi%C3%B3n_anal%C3%B3gica-digital

5.1.5 SPI

El Bus SPI (Serial Peripheral Interface) es un estándar de comunicaciones, similar al I2C.

Se ha utilizado este protocolo para realizar la comunicación entre el microcontrolador y el expansor MCP23S17 de la tarjeta, lo que permite la comunicación con la pantalla LCD.

El SPI, a diferencia del I2C, utiliza cuatro canales:

- **SCK:** Señal de reloj, proporcionado por el master, es decir, el microcontrolador.
- **SI:** Salida de datos del master y entrada de datos al esclavo.
- **SO:** Entrada de datos al master y salida de datos del esclavo. Este pin no se utilizará, puesto que no se dispone de la posibilidad de leer datos del expansor MCP23S17.
- **CS:** Chip select, sirve para que el esclavo se active y pueda comenzar la comunicación.

La comunicación será similar a la del I2C.

- Primero se realiza una configuración inicial (en la que se indican los parámetros como la frecuencia de transmisión) y se inicia el módulo SPI.
- Después, para comenzar con la transmisión, se debe activar el pin CS, para así activar el dispositivo esclavo.
- Se envía el dato deseado al bus y se espera a que se realice la transmisión correctamente.
- Por último, se desactiva el pin CS.

En el caso de la programación de la LCD mediante SPI no ha sido necesario dar uso a la librería SPI, aunque se podría haber hecho.

Si desea conocer mas información sobre el SPI consulte el siguiente blog:

https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

(Grusin, s.f.)

5.2 PROGRAMAS

5.2.1 LCD

El microcontrolador utilizado, como ya se ha mencionado anteriormente, es el PIC18F87J11. Este es un microcontrolador que utiliza direcciones de 8 bits, lo cual imposibilita la opción de utilizar la LCD de la Explorer PICDEM 18 (la utilizada) directamente. La pantalla LCD es de dos filas por 16 caracteres en cada fila y para poder dar uso a esta serán necesarios 10 bits, el byte del dato enviado (8 bits) mas el bit ``RS`` el bit ``E``.

El pin “RS” controla en que parte de la memoria LCD se están escribiendo los datos. Es aquí donde se mantiene la información que sale en la pantalla, o donde el controlador de esta busca los siguientes datos a mostrar. El pin de “lectura/escritura”(R/W) selecciona el modo de lectura o de escritura. En el caso de la Explorer utilizada, este pin se encuentra cableado a tierra, por lo tanto, no es posible poder realizar lecturas. El pin ``E`` es el pin de habilitación “enable”, este habilita los registros.

Para poder utilizar la LCD, por lo tanto, ésta está conectada al expansor MCP23S17. Este expansor utiliza el módulo SPI, mediante el cual se envían los datos a la LCD. Este módulo requiere de una programación para poder ser utilizada, para ello hay que acceder a los registros del expansor. Para poder acceder a los registros, primero se debe activar el chip select (poner a 0 el pin RA2), después se debe enviar a éste la dirección (0x40) del propio MCP23S17, después la dirección del registro al que se desea acceder y por último el dato que se desea enviar. El expansor requiere de una configuración para poder comunicarse con la LCD, una vez configurado se realiza la inicialización de la LCD y a partir de ese momento es posible enviar los datos deseado para plasmarlos en la pantalla.

Este programa nos es de gran ayuda en los siguientes, puesto que, sin este, no nos sería posible visualizar las variables adquiridas en la pantalla LCD.

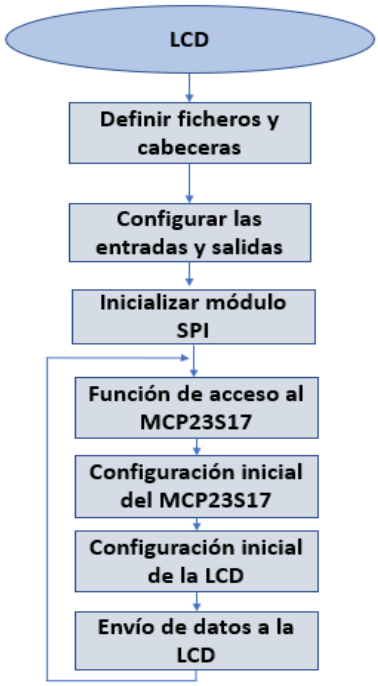


Figura 6. Diagrama de flujo LCD.

La programación viene referida en el *anexo I* en el apartado Anexos.

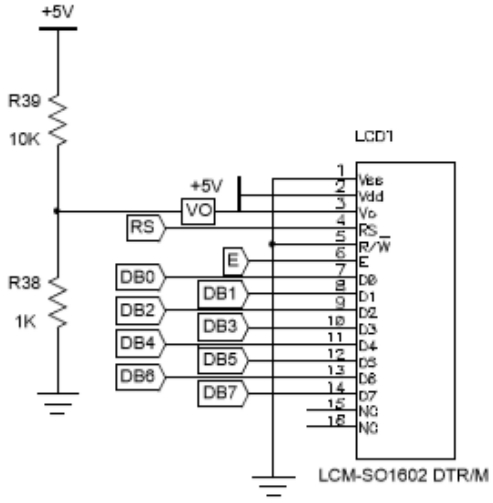


Figura 7. Conexiones de los pines LCD.

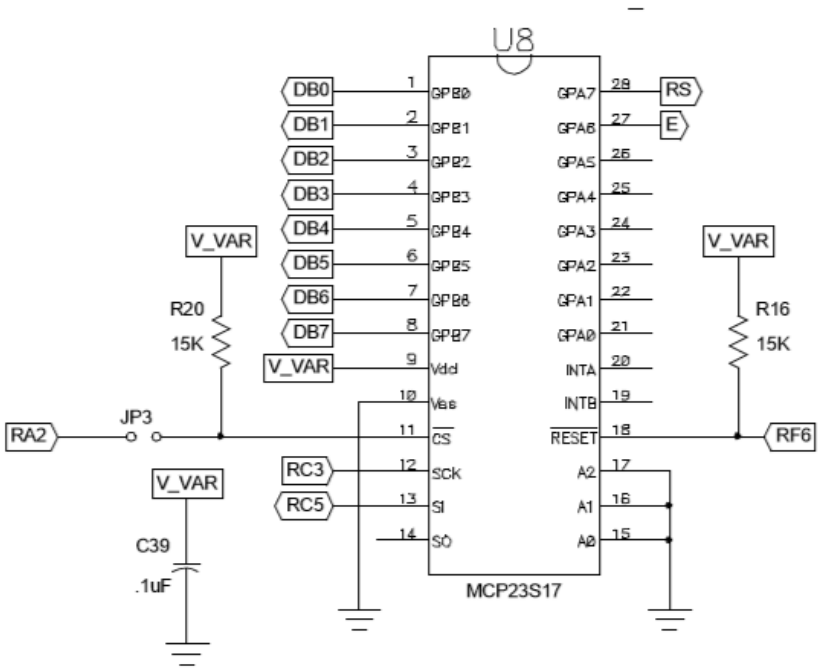


Figura 8. Conexiones de los pines del MCP23S17.

5.2.2 Sensores de temperatura integrados. PMOD TMP3,

Existen unos dispositivos integrados cuyo comportamiento depende de las variaciones de la temperatura, a estos dispositivos se les llama sensores de temperatura. Esta función se puede medir a raíz de determinados fenómenos: dilatación de cuerpos, cambio de resistencia o de la conductividad, emisión y de adsorción de rayos infrarrojos, etc.

En el caso analizado en este trabajo se da uso a un sensor cuya variación de la temperatura depende de la modificación de la resistencia eléctrica o conductividad de su material.

A diferencia de los termistores (NTC y PTC), estos elementos no funcionan únicamente como consecuencia de la variación de su resistencia con la temperatura, sino que experimentan determinados cambios en su conductividad, de manera que proporcionan a su salida variaciones de magnitudes de tipo analógico como tensión o corriente y también de tipo digital, es decir, magnitudes expresadas en binario con un determinado número de bits.

En este proyecto se ha dado uso al PMOD TMP3. El PMOD TMP3 es un módulo de sensor de temperatura construido alrededor del sensor TCN75A.

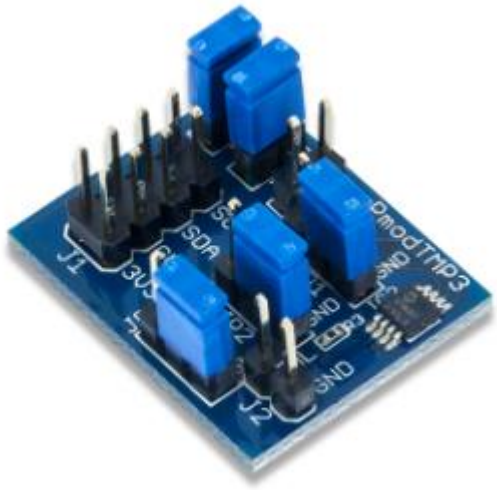


Figura 9. PMOD TMP3.

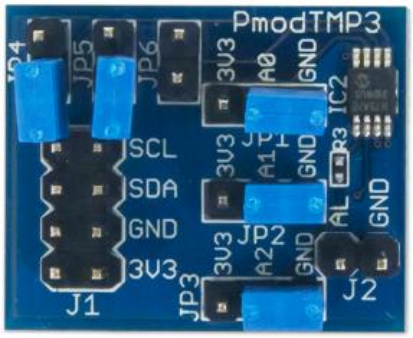


Figura 10. PMOD TMP3.

El sensor TCN75A es un sensor de temperatura que utiliza tecnología I2C para enviar datos de la temperatura en modo digital. Dispone de 8 pines:

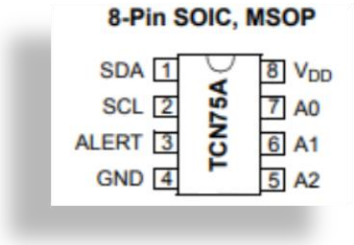


Figura 11. Pines del TCN75A.

Los pines A0, A1 y A2 son los pines de dirección que sirven para identificar el dispositivo. Como son tres pines, esto indica que se podría utilizar hasta 8 sensores en un mismo circuito (2^3=8). La dirección se decidirá poniendo dichos pines a alimentación o a tierra (1 o 0). La elección de la dirección le corresponde al propio usuario, es decir, variando los Jumpers JP1, JP2 y JP3 (como se puede observar en la figura 10) con lo que se realizan las conexiones de los pines A0, A1 y A2 a tierra o a alimentación. En mi caso los 3 pines están conectados a tierra, por lo tanto, la dirección del sensor será 0x48.

SDA es la línea por la cual se realiza la transmisión de los datos, siendo esta del tipo bidireccional.

SCL es la línea de entrada de reloj.

ALERT es la línea de alerta que es programable por el usuario para producir una alerta cuando la temperatura supere dicho valor.

V_{DD} es la línea de alimentación.

GND es la línea de tierra.

La resolución de la temperatura es seleccionable por el usuario y es del rango de 0,5° C hasta 0,0615° C. En el proyecto se utilizará la resolución por defecto, 0,5° C.

La alimentación que soporta el dispositivo va desde 2,7 V hasta 5,5 V, por lo tanto se utilizará la fuente de V_{CC} de 3,3 V proporcionada por el propio microcontrolador.

La resolución ofrecida por el sensor varía desde 9 bits hasta 12 bits, en el programa se utilizará la resolución por defecto (9 bits). Para poder realizar la lectura del dispositivo primero se debe que envíar la dirección del sensor mas el último bit a 0 (este corresponde al bit R/W, el cual es un 0 si se desea escribir en el sensor y un 1 si se desea leer desde el sensor) desde el microcontrolador, que será el dispositivo maestro de la conexión, siendo el sensor de temperatura el esclavo. Entonces, si se desea escribir se enviará el byte 0x90 y si se desea leer se enviará 0x91.

A continuación se debe enviar la dirección del registro del cual se desea leer el dato, que consistiría en el registro donde está almacenado el valor de la temperatura (0x00).

Una vez enviado la dirección del registro realizaremos un stop y acto seguido un start (reestart), después se enviará el valor de la dirección mas un último bit a 1, puesto que esta vez se desea leer el valor. Este valor será enviado en dos bytes, primero el MSB (Most Significant Byte) y luego el LSB (Less Significant Byte), los bits válidos serán los 9 primeros, es decir, los 8 bits del MSB y el Most Significant bit del LSB, el resto de los bits serán 0. El valor será enviado en formato A2, ya que la temperatura puede tener un valor negativo. También se ha programado una alarma, la cual tiene como función dar un aviso cuando la temperatura exceda los 30°C. En caso de que esto ocurra, se encenderá el LED de la posición RD1 (el segundo LED empezando por la derecha) durante un segundo.

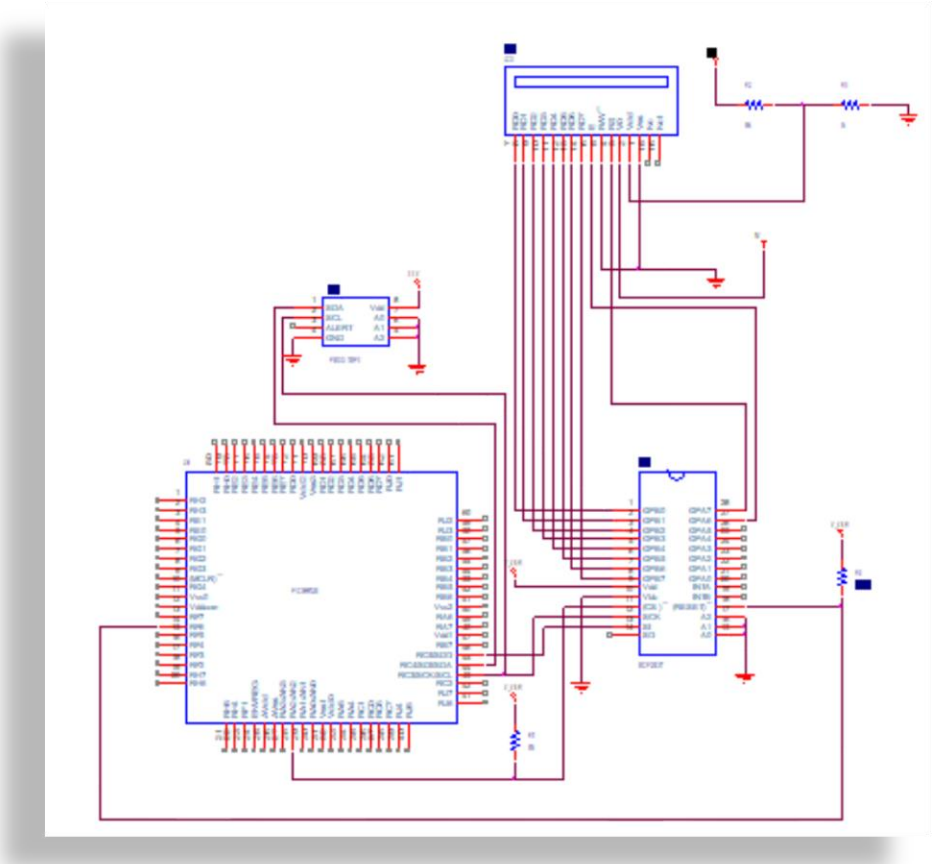


Figura 12. Esquemático PMOD TMP3.

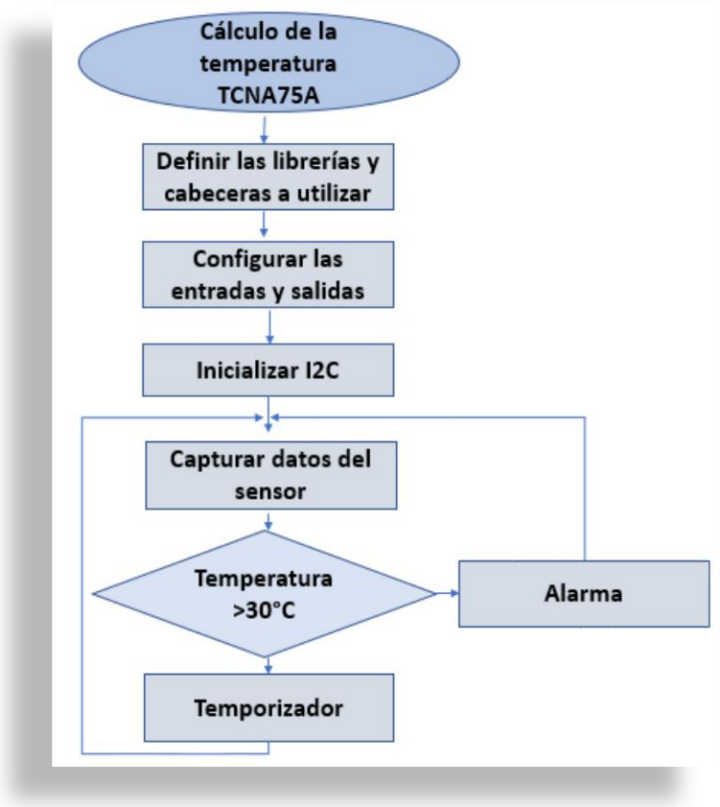


Figura 13. Diagrama de flujo TCN75A.

La programación viene referida en el anexo J en el apartado Anexos.

5.2.3 SENSOR DE TEMPERATURA Y HUMEDAD. HIH 6121-021-001

La humedad expresa la cantidad de agua que existe en un medio determinado, ya sea líquido, sólido o gaseoso. Conocer esta cantidad o concentración de agua puede ser muy útil para determinados propósitos, ya que podría influir en el funcionamiento de procesos industriales, alimentarios, agrícolas, o en la duración de ciertos elementos incorporados a electrodomésticos, vehículos, telefonía…

La forma de referirse a esta magnitud es de diversas maneras:

Humedad absoluta: es la relación entre la masa de agua presente en el medio (kg), y el volumen (en m^3) de dicho medio.

Humedad específica: es la relación entre la masa de agua y la masa de sustancia seca presentes en el medio (ambos expresados en kg).

Humedad relativa: es la cantidad de agua que contiene un gas expresada en tanto por ciento de la cantidad que el gas tendría en estado de saturación, a la misma temperatura y presión absoluta.

$$Hr = \frac{Pv}{p_{sat}} x100$$

Para obtener información y medidas de la humedad, sobre todo de la relativa, se han desarrollado sensores de condensación, electrolíticos, resistivos, capacitivos…

Los sensores de tipo capacitivo son los más utilizados en electrónica.

El transductor empleado en este proyecto es el sensor HIH6121. El sensor HIH6121-021-001 es un sensor de humedad relativa y temperatura en un mismo encapsulado. El sensor ofrece una resolución de humedad ±4.0 %RH y temperatura de 0.5°C y un rango de humedad que parte de ±10%RH hasta 90%RH y temperatura que parte desde 5°C hasta 50°C.

El sensor dispone de módulo I2C, lo cual nos permite capturar la temperatura y la humedad por los canales SCL y SDA.

Las líneas SDA (4) y SCL (3) son bidireccionales, las cuales necesitan ir conectadas a unas resistencias de Pull-Up. También será necesario utilizar un condensador de desacoplo entre los pines GND (2) y V_{DD} (1).

El módulo I2C del sensor trabaja a una frecuencia de 400 kHz y los datos pueden ser transmitidos a unas velocidades de hasta 400 kbit/s en el modo rápido (fast mode), pero yo utilizo el modo estándar a 100 kbit/s, ya que no necesito tales velocidades.

El sensor trabaja en modo esclavo, es decir, el microcontrolador será el Master de la comunicación.

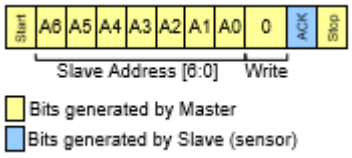
Dirección del sensor: 0x27

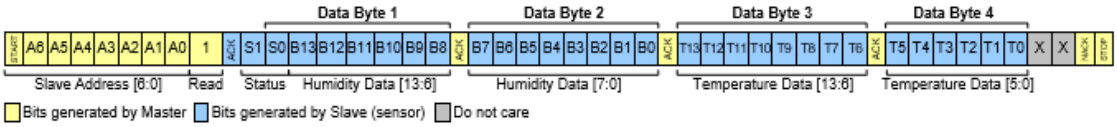
Bit R/W en modo escritura: 0

Bit R/W en modo lectura: 1

Byte a enviar para escritura: 0x4E

Byte a enviar para lectura: 0x4F





Para realizar la comunicación el Master primero debe enviar el byte para escritura. Cuando el sensor reciba el dato, éste responderá con el bit de acknowledge, y acto seguido se produce un Stop. El Master realiza un Reestart, envía el byte para lectura, y lee los bytes con datos de humedad y temperatura.

Los dos primeros bytes leídos corresponden a la humedad y los dos últimos corresponden a la temperatura.

También se programa una alarma que se activará cuando la temperatura exceda los 30°C y la humedad exceda el 80%RH, está se activará durante 1 segundo.

Humedad

Primero se recibe el MSB, del cual los dos most significant bits corresponden a los bits de estado, que serán 0 cuando trabaje en modo normal. Después se recibe el LSB.

Este valor será de 14 bits, que mediante la ecuación 1 se convierte en valor relativo de la humedad (%RH).

$$Humedad\ (\%RH) = \frac{Valor\ de\ 14\ bits\ recibido}{2^{14} - 2} x100$$

Ecuación 1. Cálculo de la humedad relativa.

Temperatura

Primero se recibe el MSB y después el LSB (del cual los dos less significant bits serán 0).

Mediante la ecuación 2 se consigue transformar el valor de 14 bits en °C.

$$Temperatura\ (°C) = \frac{Valor\ de\ 14\ bits\ recibido}{2^{14} - 2} x165 - 40$$

Ecuación 2. Cálculo de la temperatura.

Para la temperatura:

Partimos de: T_{13} T_{12} T_{11} T_{10} T_9 T_8 T_7 T_6 T_5 T_4 T_3 T_2 T_1 T_0 0 0

Para obtener: 0 0 T_{13} T_{12} T_{11} T_{10} T_9 T_8 T_7 T_6 T_5 T_4 T_3 T_2 T_1 T_0

Después se realiza el cálculo de la ecuación 2 por Software y obtendremos el valor de la temperatura.

Para la humedad:

Con tan solo realizar el cálculo de la ecuación 1 por Software se obtendrá el valor de la humedad relativa.

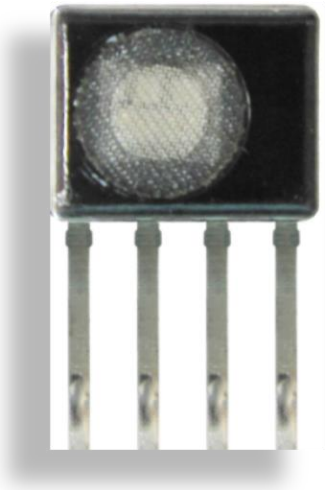


Figura 14. HIH6121.

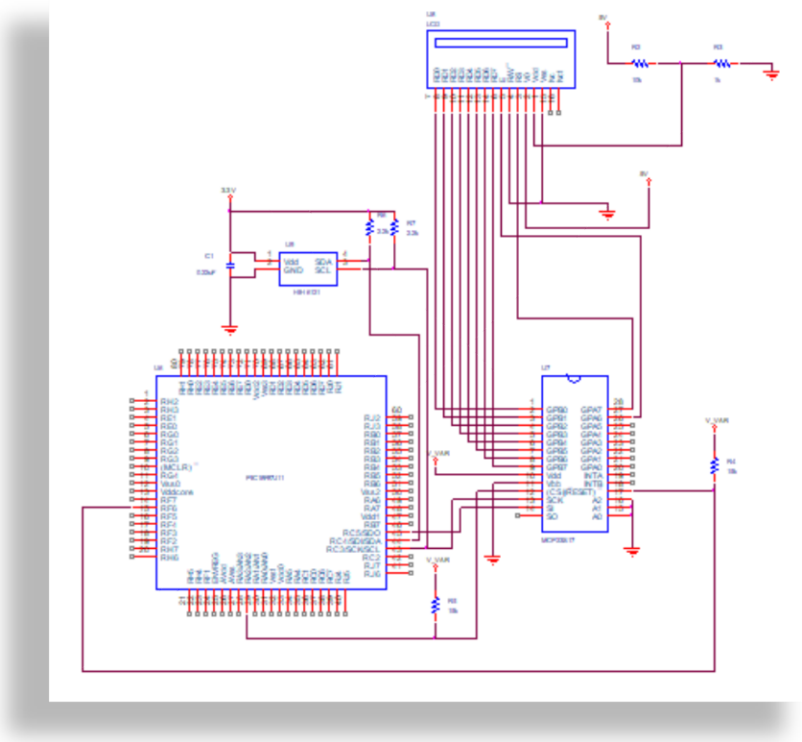


Figura 15. Esquemático HIH6121.

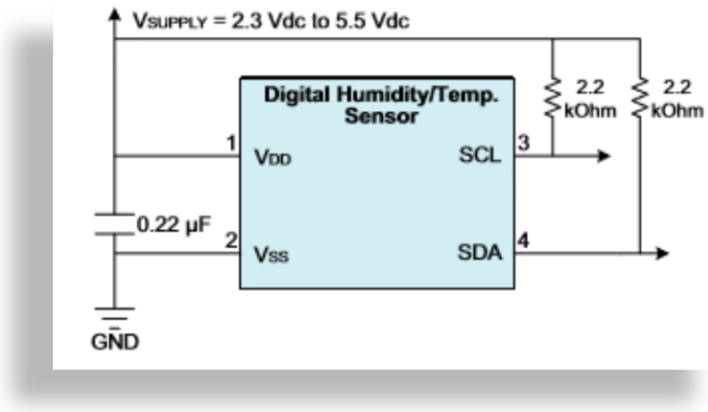


Figura 16. Pines del sensor HIH6121.

.

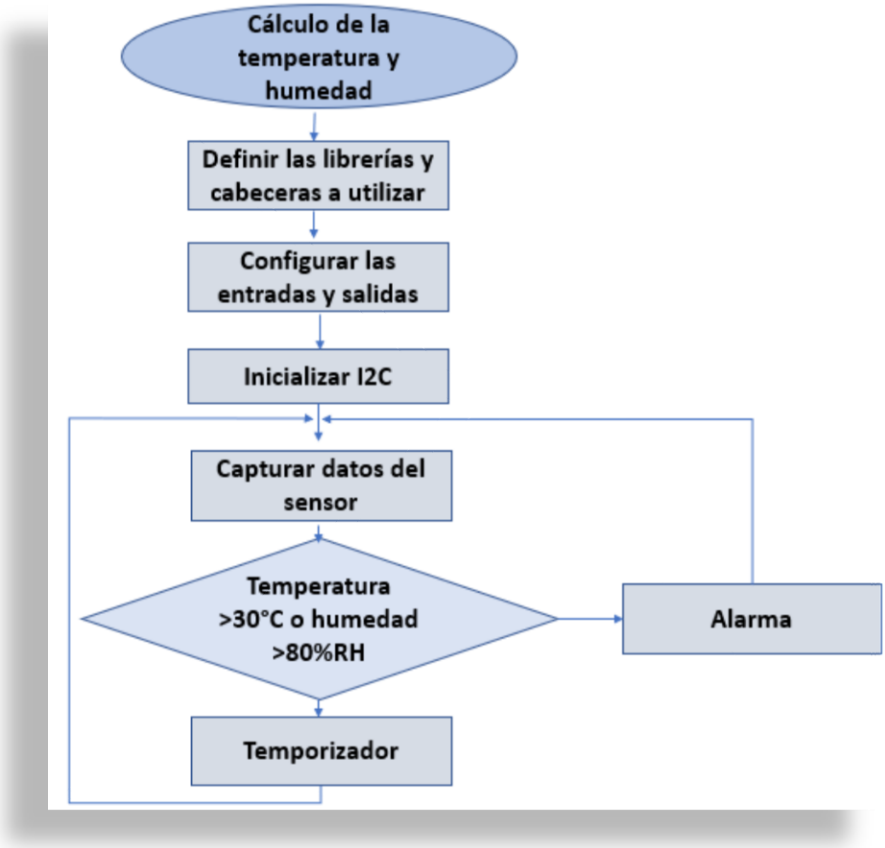


Figura 17. Diagrama de flujo HIH6121.

La programación viene referida en el *anexo K* en el apartado Anexos.

5.2.4 Sensor de presión. MPX2050DP.

Para medir la presión se utilizan sensores dotados de un elemento especifico sensible a la presión, el cual proporciona una señal eléctrica al variar dicha presión, además de un transductor o elemento acondicionador que facilita la medida de la deformación producida.

Los sensores electrónicos son aquellos que necesitan de una fuente de energía eléctrica para proporcionar su señal. Para detectar la deformación que la presión produce, generalmente sobre una membrana, se utilizan diferentes principios físicos, resistivos, inductivos, capacitivos, piezorresistivos… posteriormente es necesario un circuito acondicionador para procesar la lectura.

El sensor empleado este proyecto es el MPX2050DP. Este sensor de presión de silicio es del fabricante Motorola, del tipo piezoeléctrico y capaz de soportar presiones de hasta 7,25 PSI (Pounds per Square Inch o libras por pulgada cuadrada) o 50 kPa (1 kiloPascal=0,145 psi) dentro de un margen de temperatura que va desde 0°C hasta +85°C. Proporciona una tensión máxima a su salida de 40mV, que es directamente proporcional a la diferencia de presión aplicada a sus entradas (P1>P2).

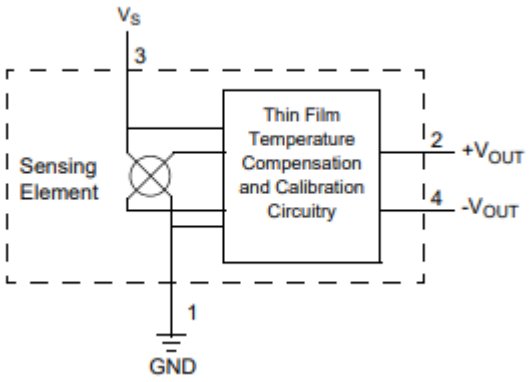


Figura 18. Pines del MPX2050DP.

Se realizará un circuito con el sensor de presión MPX2050DP, al cual va asociado un circuito acondicionador mediante amplificadores operacionales y que conectado al convertidor analógico digital del microcontrolador, permita conocer la presión aplicada en el sensor en kPa. Para ello se tomará la característica de la tensión de salida en función de la diferencia de presión, facilitada por el fabricante y que se puede ver en la figura 22. Esta presión diferencial viene referida a la diferencia entre dos presiones absolutas (P1-P2).

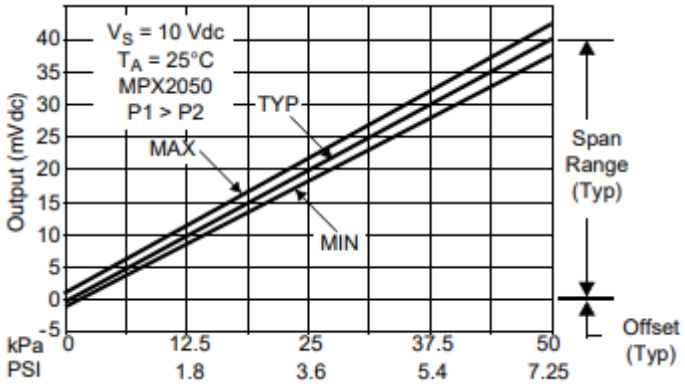


Figura 19. Tensión de salida vs presion diferencial.

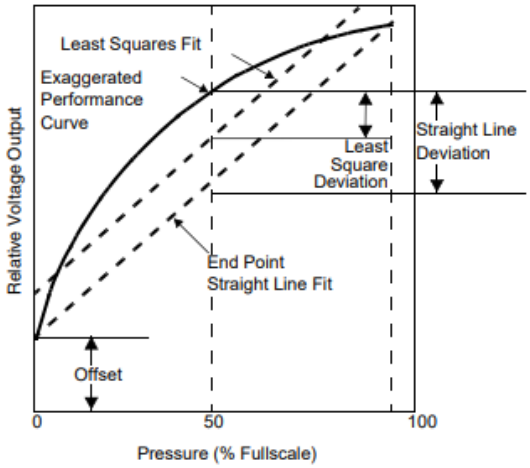


Figura 20. Linealidad del MPX2050DP.

MONTAJE

Como ya se ha visto, la tensión diferencial máxima que dará el sensor es de 40mV, por lo tanto, se necesitará un circuito acondicionador capaz de adaptar esta señal tan pequeña para poder ser procesada. Para ello se utiliza un amplificador operacional (AD620), ver anexo en la Bibliografía

Para probar el funcionamiento se introduce aire por uno de los tubos que dispone el sensor, entonces se produce una salida diferencial de tensión que es adaptada por el amplificador operacional. La tensión de salida del amplificador entrará por el pin RA0 (entrada analógica) del microcontrolador, donde será procesada y visualizada.

La conversión que se realiza en el microcontrolador será de 8 bits y a 5V (tensión que recibe el microcontrolador). Por lo tanto, la resolución será de 5:256 mV/bit, entonces existe un factor de 20 para adaptar el valor. Expresaremos la lectura en kPa (1mV=1,5kPa), por lo tanto, tenemos que en la conversión debemos multiplicar la lectura por 30. Por último, se hace uso del módulo ADC del microcontrolador, que nos convertirá el valor de la tensión en un valor binario.

Figura 21. MPX2050DP.

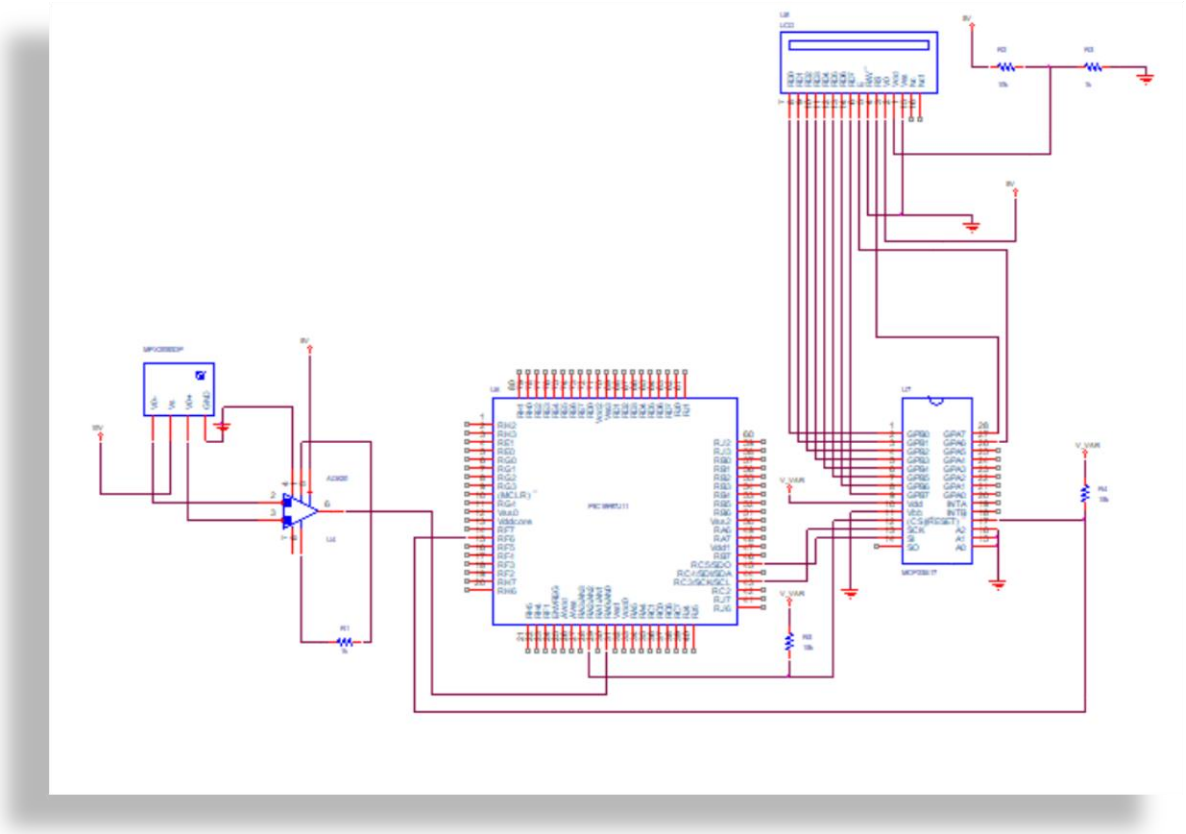


Figura 22. Esquemático MPX2050DP.

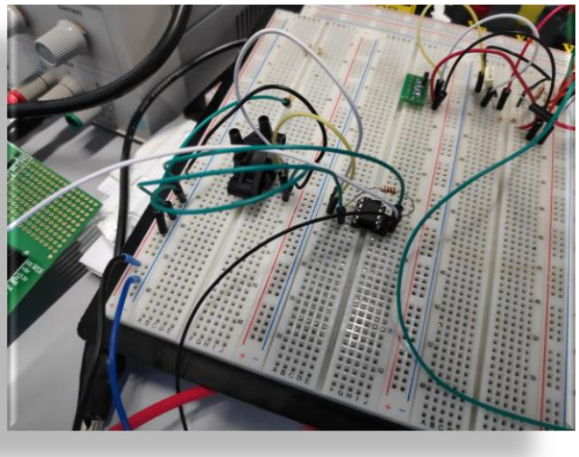


Figura 23. Montaje MPX2050DP(1).

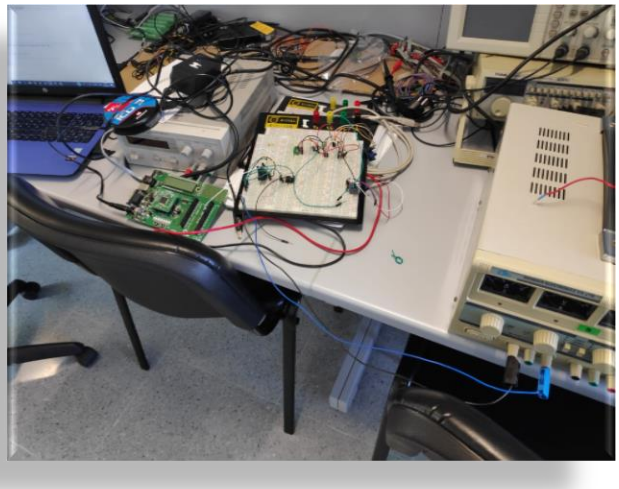


Figura 24. Montaje MPX2050DP(2).

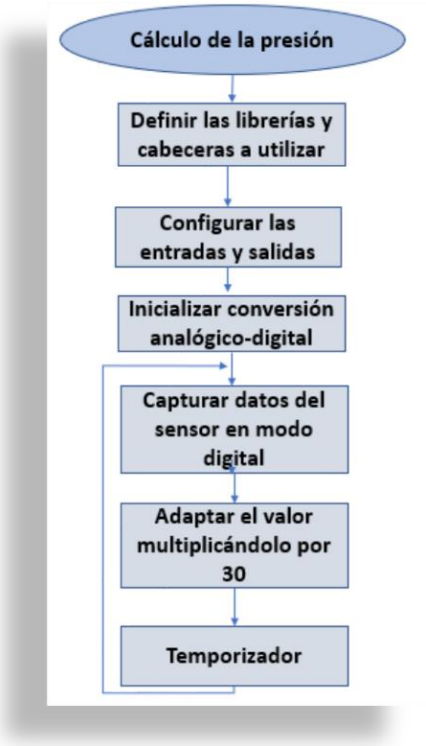


Figura 25. Diagrama de flujo MPX2050DP.

La programación viene referida en el *anexo L* en el apartado Anexos.

5.2.5 Sensor de aceleración. MMA8451Q.

Los sensores de aceleración o acelerómetros son dispositivos que basan su funcionamiento en el principio de la aceleración de Newton (Fuerza=Masa x Aceleración), por lo que deben estar provistos de una masa móvil, sobre el cual se aplicará la fuerza que producirá la aceleración o deceleración.

Los acelerómetros, en su modo de medición dinámico, permiten medir el movimiento y las vibraciones a las que está sometido un cuerpo, y en su modo estático pueden medir la inclinación con respecto a la gravedad.

El sensor empleado en este proyecto es el MMA8451Q. El MMA8451Q es un sensor de aceleración capacitivo micromecanizado que dispone de 3 ejes (X, Y, Z) y 14 bits de resolución. Este acelerómetro dispone de varias opciones configurables y dos pines de interrupción. La alimentación parte desde 1,8 V hasta 3,6 V. Tiene muy

buena compensación respecto de las variaciones de temperatura y opera en un rango de ±2g, 42g, ±8g. La sensibilidad del sensor ante el rango de ±8g es de 1024 cuentas/g, este es el rango que se utilizará, puesto que es el predeterminado y es adecuado para la aplicación.

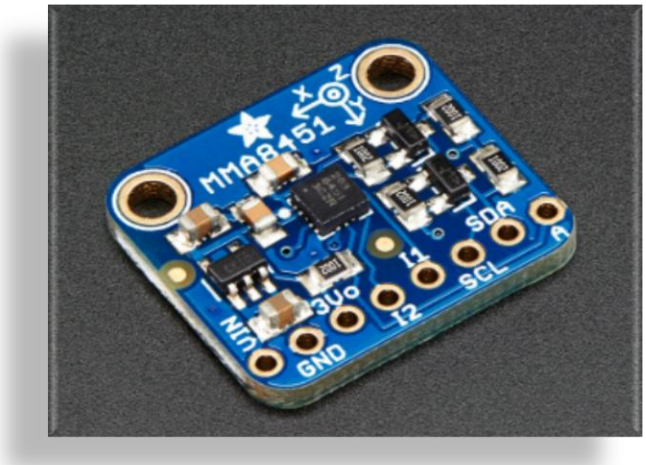


Figura 26. MMA8451Q.

Mediante los resultados obtenidos de los ejes X, Y y Z se puede detectar la aceleración del sensor.

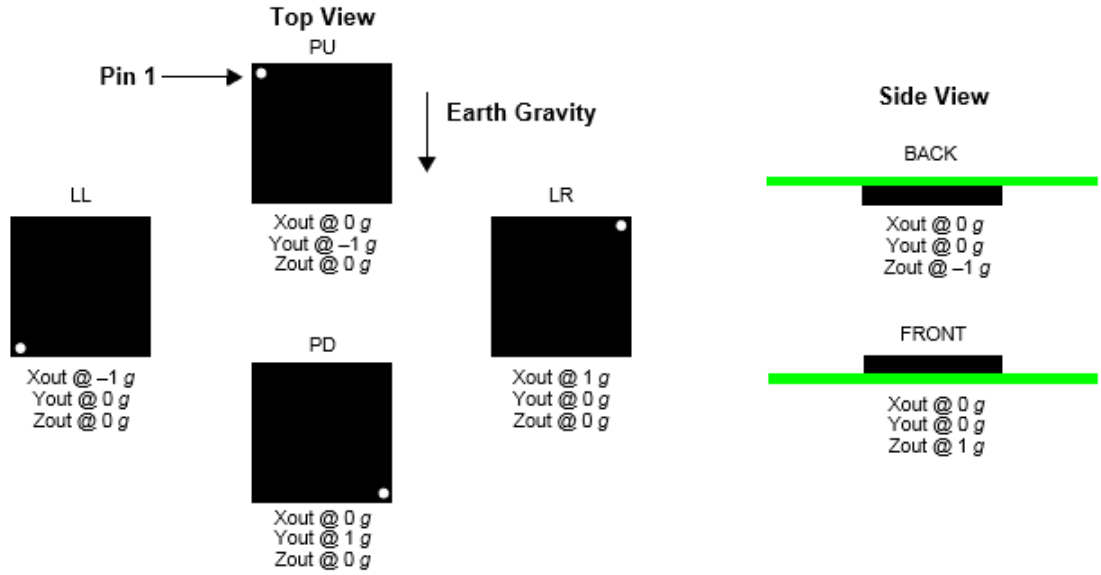


Figura 27. Posiciones del sensor MMA8451Q.

El acelerómetro está provisto de un bus I2C, el cual permite comunicarse con el microcontrolador, con el que obtendremos los datos en complemento A2, para así poder saber si la posición de unos de los ejes es positivo o negativo.

El sensor dispone de un pin (A0), mediante el cual se puede variar la dirección del sensor (*ver figura 29*).

Tabla 2. Direcciones del MMA8451Q.

Command	[6:1] Device address	[0] SA0	[6:0] Device address	R/W	8-bit final value
Read	001110	0	0x1C	1	0x39
Write	001110	0	0x1C	0	0x38
Read	001110	1	0x1D	1	0x3B
Write	001110	1	0x1D	0	0x3A

El funcionamiento es el mismo a los I2C vistos anteriormente, primero se envía la dirección del dispositivo con el bit de escritura (en nuestro caso 0x3A), después se envía la dirección del registro del que se desea leer el dato. Una vez realizada la

primera parte se realiza un reestart y se envía la dirección del sensor mas el bit de lectura (ox3B) y después se leen los datos.

El dispositivo dispone de varios registros que parten desde el 0x00 hasta el 0x31.

Los registros en donde se almacenan los datos de la posición son los siguientes:

0x01: OUT_X_MSB

0x02: OUT_X_LSB

0x03: OUT_Y_MSB

0x04: OUT_Y_LSB

0x05: OUT_Z_MSB

0x06: OUT_Z_LSB

El MMA8451Q dispone de un modo de ejecución llamado Multiple-byte read, este modo es idóneo para la aplicación realizada, ya que, consiste en realizar una lectura continuada de los registros sin tener que volver a realizar la llamada a estos. Una vez habiendo llamado al registro del primer byte que se desea leer, automáticamente se aumentará una posición en los registros, lo cual nos permite leer los seis bytes de datos seguidos y conocer la posición realizando un solo ciclo.

Una vez capturado los datos se debe realizar una serie de conversiones para poder obtener la posición de los tres ejes en una variable cada una, ya que, los dos less significant bits de los bytes OUT_X_LSB, OUT_Y_LSB, OUT_Z_LSB serán 0.

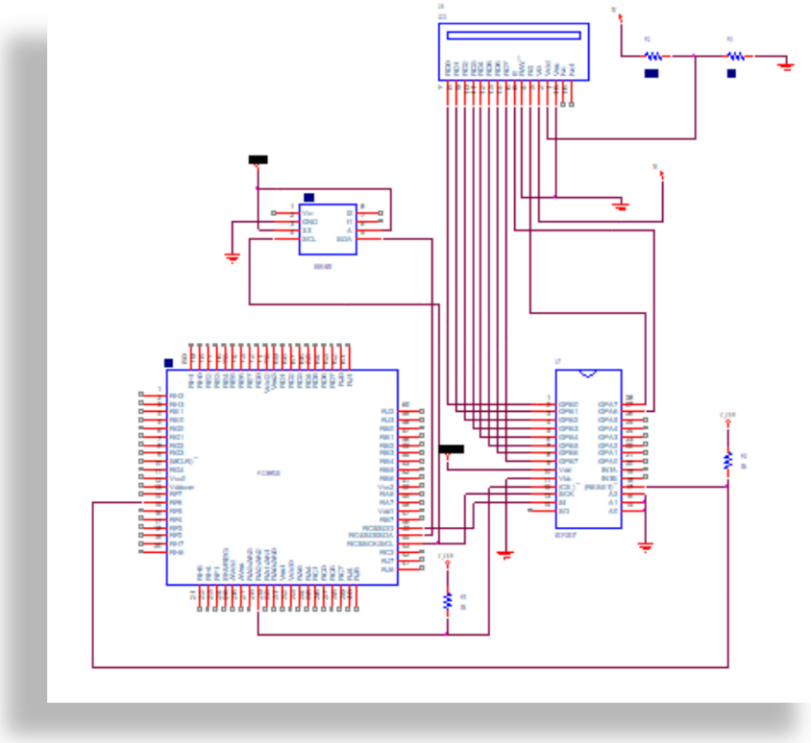


Figura 28. Esquemático MMA8451Q.

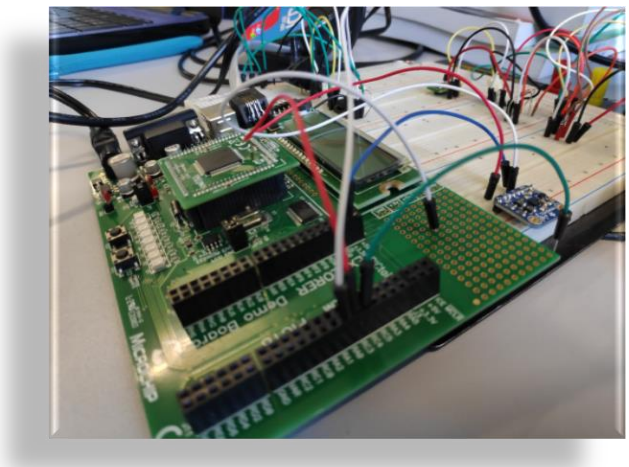


Figura 29. Montaje MMA8451Q.

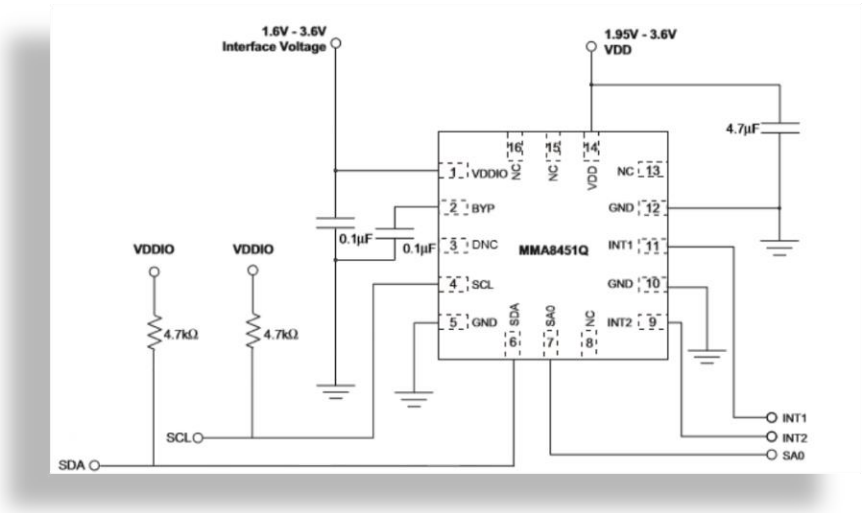


Figura 30. Pines del MMA8451Q.

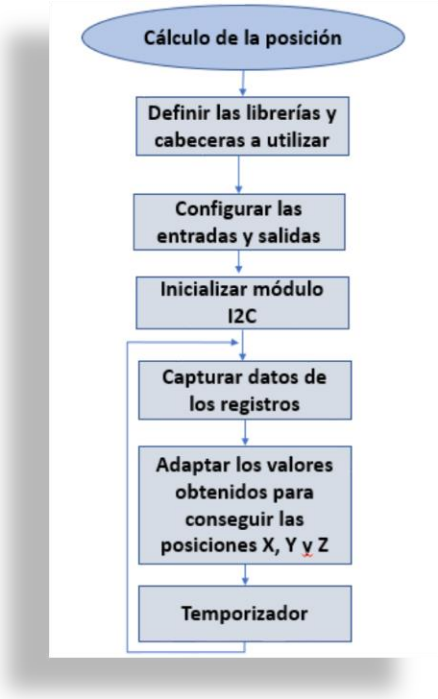


Figura 31. Diagrama de flujo MMA8451Q.

La programación viene referida en el *anexo M* en el apartado Anexos.

5.2.6 Temperatura y humedad y temperatura

Para demostrar que es posible realizar un mismo programa en el que estén integrados mas de un sensor se ha realizado un programa en el que se han implementado el sensor de temperatura TCNA75A (TMP3) y el sensor de humedad y temperatura (HIH6121-021-001). Esto es posible porque los sensores disponen de unas direcciones determinadas (o en algunos casos variables por el propio usuario). En el caso que se plantea ahora el sensor TCN75A tiene la dirección 0x48 y el HIH6121-021-001 0x27, por lo tanto, realizando la llamada a la dirección deseada conoceremos el valor que nos proporcione el sensor que deseemos.

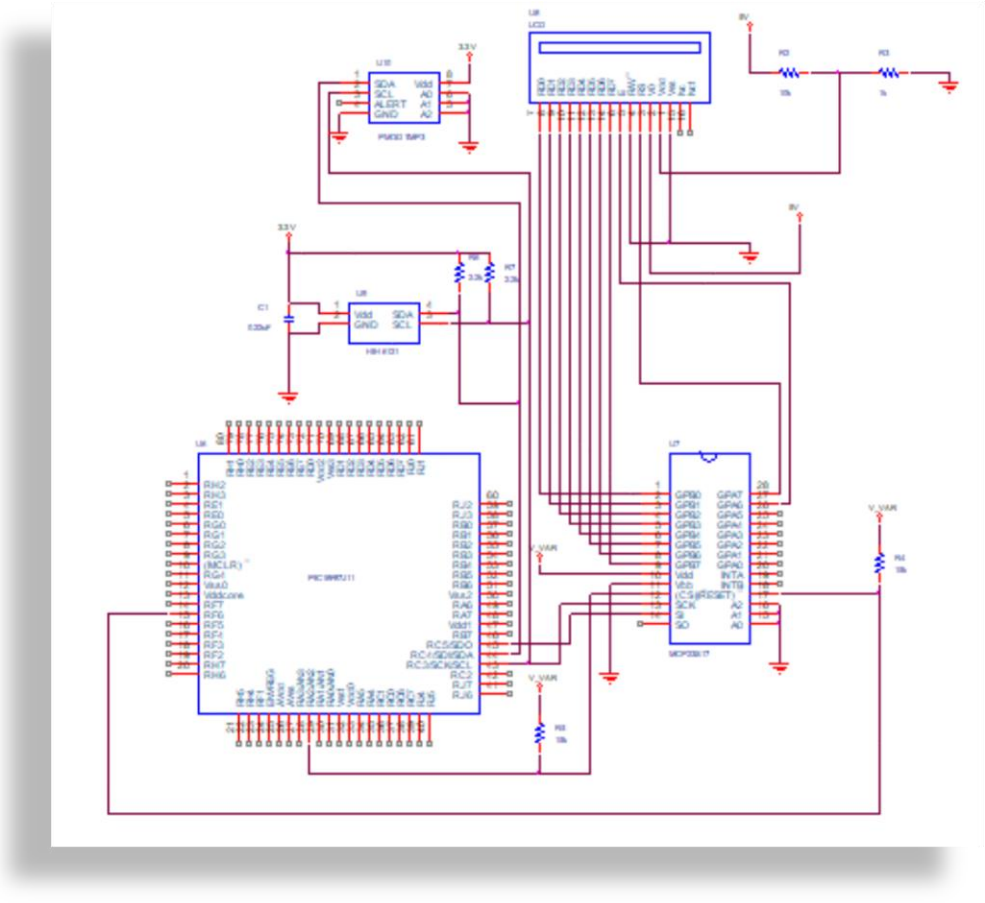


Figura 32. Esquemático PMOD TMP3 y HIH6121.

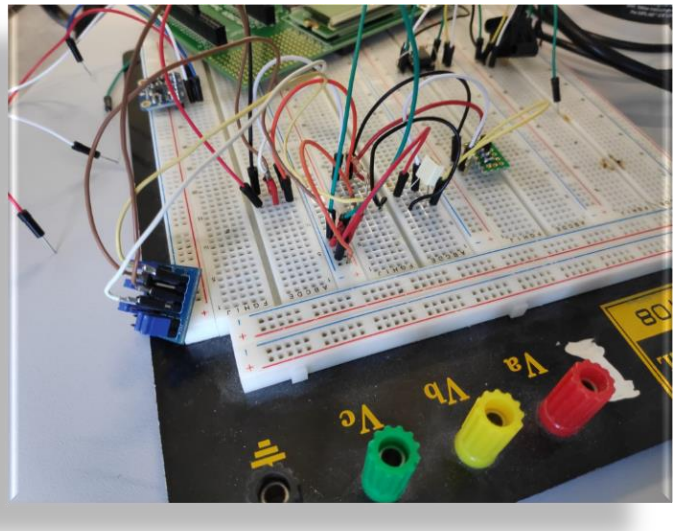


Figura 33. Montaje PMOD TMP3 y HIH 6121.

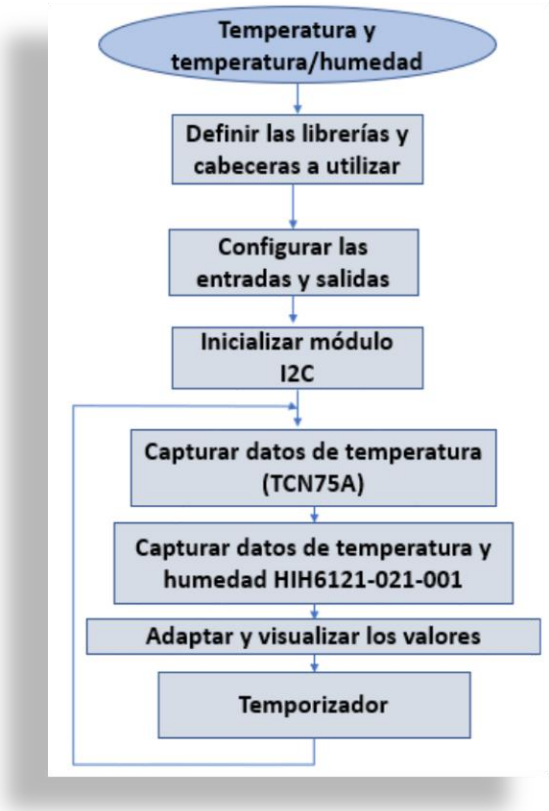


Figura 34. Diagrama de flujo TCNA75A y HIH 6121.

La programación viene referida en el *anexo N* en el apartado Anexos.

5.2.7 Programa conjunto

Por último, se ha realizado un programa que contenga todos los transductores juntos en un mismo código.

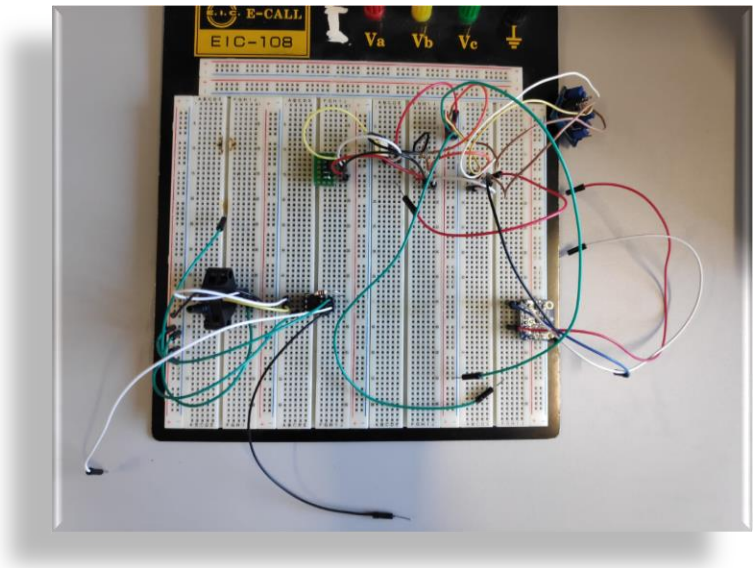


Figura 35. Todos los montajes (1).

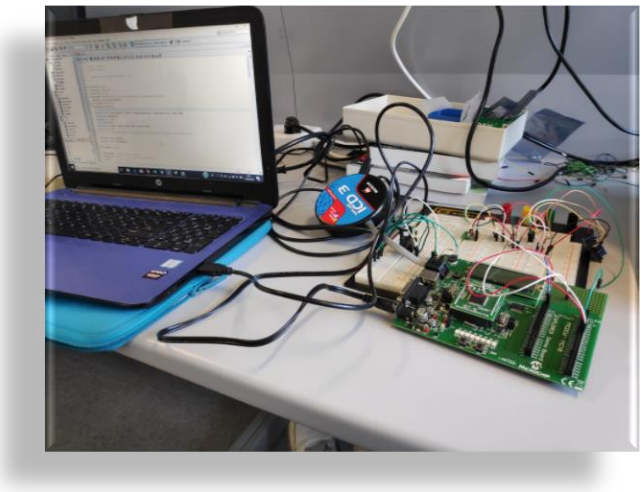


Figura 36. Todos los montajes (2).

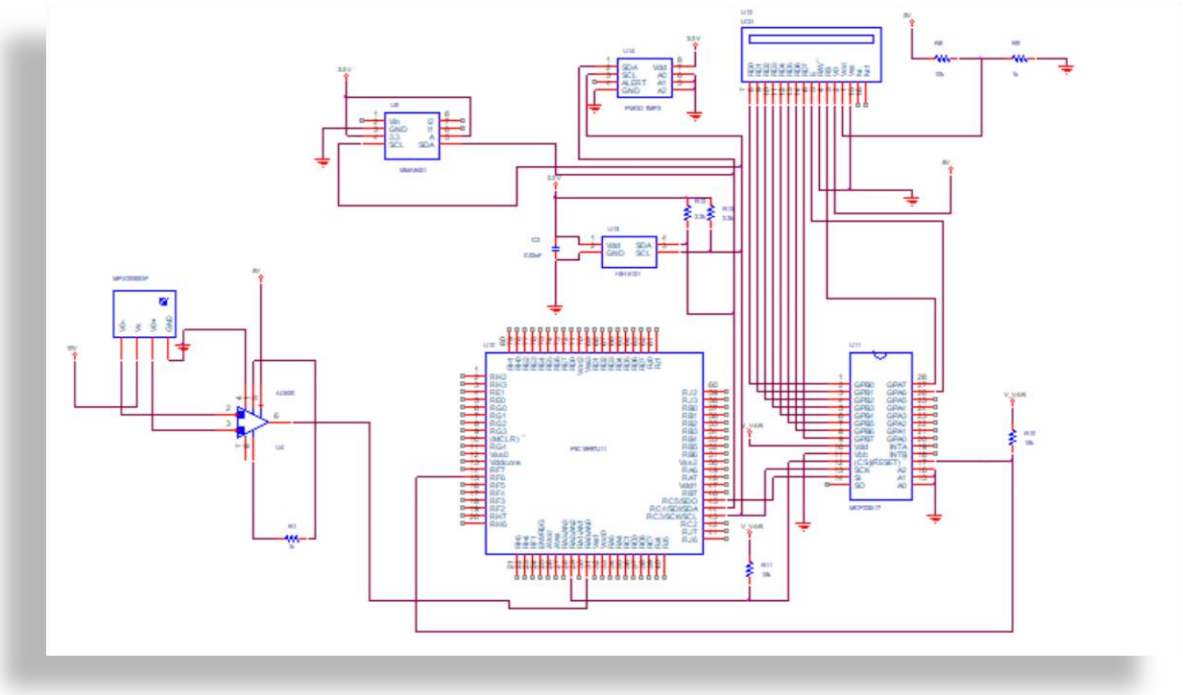


Figura 37. Esquemático todos los transductores.

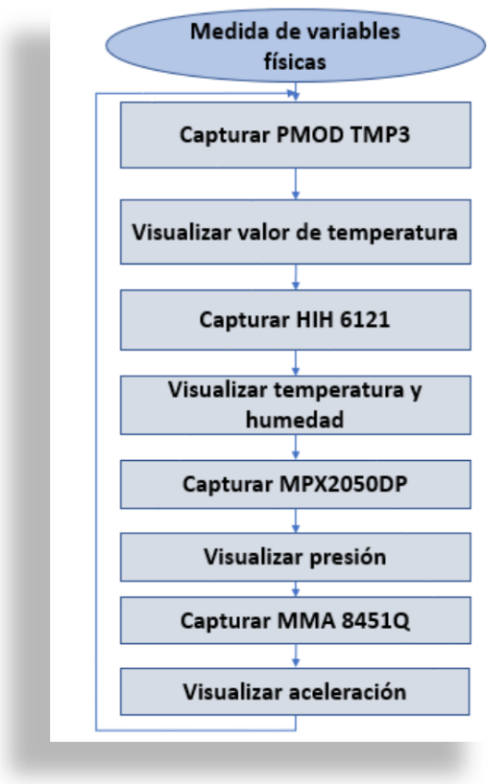


Figura 38. Diagrama de flujo todos los transductores.

La programación viene referida en el *anexo O* en el apartado Anexos.

6 DIAGRAMA DE GANTT

Tareas	Semanas															
	25/02/2019	04/03/2019	11/03/2019	18/03/2019	25/03/2019	01/04/2019	08/04/2019	15/04/2019	22/04/2019	29/04/2019	06/05/2019	13/05/2019	20/05/2019	27/05/2019	03/06/2019	10/06/2019
	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16
Estudio del programa MPLAB X IDE																
Comprobación de que el PIC18F87J11, ICD3… funcionan correctamente																
Estudio de las funciones de C en Microcontroladores																
Estudio del PIC18F87J11 y la Explorer																
Estudio del I2C																
Estudio del ADC																
Estudio del SPI																
Programación y montaje del PMOD TMP3																
Programación y montaje del HIH 6121																
Programación de la LCD																
Programación y montaje del MPX2051DP																
Programación y montaje del MMA8451Qç																
Programación y montaje del PMOD TMP3 junto al HIH6121																
Mejora de los programas y captura de resultados																

7 Resultados

hes

Name	Type	Address	Value
temperatura	float_24	0x1	24.0
<Enter new watch>			

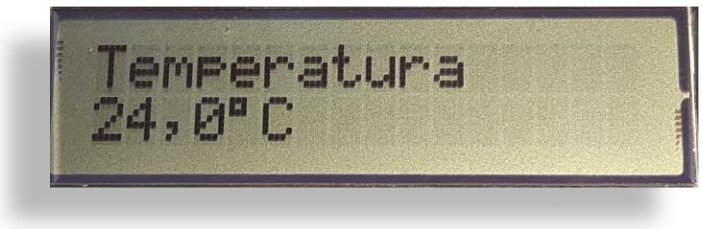


Figura 39. Resultado PMOD TMP3.

bles × Call Stack

Name	Type	Address	Value
temperatura	float_24	0x38	27.007812
humedad	float_24	0x35	89.48242

Figura 40. Resultado HIH6121.

Figura 41. Resultado MPX2050DP.

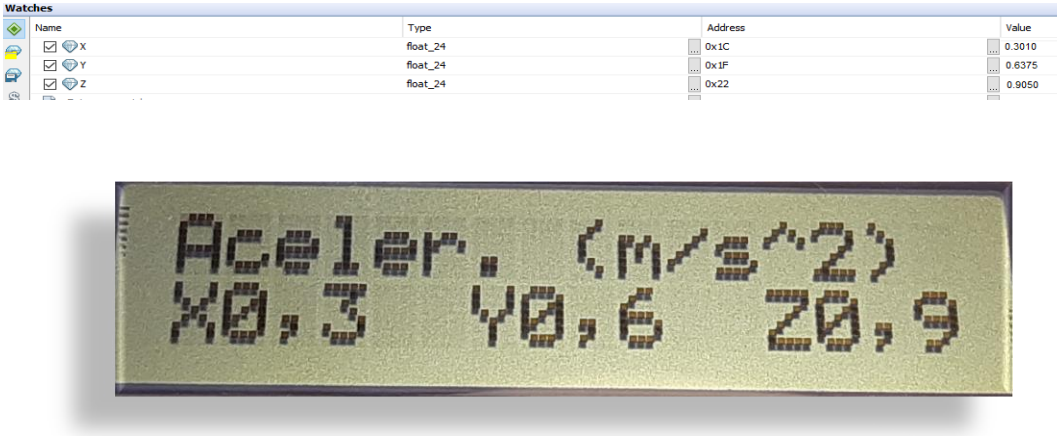


Figura 42. Resultado MMA8421Q.

8 Presupuesto

En el proyecto se ha dado uso al microcontrolador PIC18F87J11, Explorer PICDEM PIC18 e ICD3 como materiales principales de partida.

Tabla 3. Presupuesto materiales principales.

Material	Presupuesto
PIC18F87J11	3.54€
PICDEM PIC18	Actualmente fuera de producción. Precio estimado 65€
ICD3	178,94€

El presupuesto total de los materiales principales será de 247,48€.

Para realizar los montajes se ha necesitado una Protoboard, cables, resistencias, condensadores y el objeto principal del proyecto, los sensores. Los valores del presupuesto de los siguientes materiales son precios estimados, ya que, no se han comprado estos unitariamente.

Tabla 4. Presupuesto materiales secundarios.

Material	Presupuesto
Cables	2,00€
Resistencias varias	1,50€
Condensadores	3,00€
Protoboard	49,22€

El presupuesto total de los materiales secundarios será de 55,72€.

En la siguiente tabla se verá reflejado el presupuesto de los transductores utilizados.

Tabla 5. Presupuesto transductores.

Sensor	Presupuesto
PMOD TMP3	6,23€
HIH 6121	14,45€
MPX2050DP	12,95€
MMA8451Q	2,92€

El presupuesto total de los transductores será de 36,55€.

Se debe tener también en cuenta las horas trabajadas en el proyecto, teniendo en cuenta que las horas trabajadas para la realización del proyecto son alrededor de 300 horas.

Dato	Horas	€/Horas	Coste
Horas ingeniería	300	20	6.000€

Por lo tanto, el presupuesto total para la realización del proyecto será:

$$247{,}48 + 55{,}72 + 36{,}55 + 6.000 = 6339{,}75€$$

SUBTOTAL		6.339,75€
Indirectos	7%	443.78€
SUBTOTAL2		6.783,53€
Imprevistos	10%	678.35€
SUBTOTAL 3		7461,88€
Financieros	4%	298,48€
	TOTAL	7760,35€

9 Conclusiones

Una vez finalizado el proyecto se procede a detallar las conslusiones obtenidas:

- Hoy en día el conocimiento de las variables físicas del medio ambiente es de vital importancia en infinidad de casos, por lo tanto, este proyecto es algo extremadamente útil como una solución para conseguir capturar estas variables.
- Este no es el único método para conseguir conocer estas variables físicas, ya que existen múltiples aplicaciones para ello, ya sean utilizando o no sensores y sin necesidad de programar nada.
- De todos modos, el hecho de haber realizado programas y el conocimiento de su desarrollo permite realizar modificaciones en caso de ser necesario, por ejemplo, si se ha programado una alarma que se active cuando la temperatura exceda los 30°C y se desea aumentar la temperatura de dicha alarma a 35°C, con tan solo modificar un poco la programación esto será posible.
- Además, siendo posible la combinación de varios sensores, se llega a la conclusión de que con tan solo un programa que contenga todos los transductores y con su correspondiente circuito es posible conocer todas las variables físicas medibles mediante la correcta y deseada programación.
- Por último, añadir que la combinación de la instrumentación de campo y la programación es algo realmente interesante y que ofrece una gran variedad de oportunidades.

10 Bibliografía

Crespo, E. (s.f.). *Aprendiendo Arduino*. Obtenido de https://aprendiendoarduino.wordpress.com

Grusin, M. (s.f.). *Sparkfun*. Obtenido de https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Microchip. (1989). Obtenido de https://www.microchip.com/

Wikipedia. (8 de 06 de 2019). Obtenido de https://es.wikipedia.org/wiki/Conversi%C3%B3n_anal%C3%B3gica-digital

11 Anexos

Anexo A. Datasheet PIC18F87J11:

http://ww1.microchip.com/downloads/en/DeviceDoc/39778e.pdf

Anexo B. Datasheet Explorer PICDEM PIC18:

http://ww1.microchip.com/downloads/en/devicedoc/50001721c.pdf

Anexo C. Datasheet TCN75A:

http://ww1.microchip.com/downloads/en/DeviceDoc/21490D.pdf

Anexo D. PMOD TMP3:

https://reference.digilentinc.com/reference/pmod/pmodtmp3/reference-manual

Anexo E. Datasheet HIH6121:

https://sensing.honeywell.com/honeywell-sensing-humidicon-hih6100-series-product-sheet-009059-6-en.pdf

Anexo F. Datasheet MPX2050DP:

https://www.nxp.com/docs/en/data-sheet/MPX2050.pdf

Anexo G. Datasheet AD620:

https://www.analog.com/media/en/technical-documentation/data-sheets/AD620.pdf

Anexo H. Datasheet MPX2050DP:

https://www.nxp.com/docs/en/data-sheet/MMA8451Q.pdf

Programas MPLAB X

Anexo I. Programación LCD.

```
#include <xc.h>
#include "config.h"
#include "string.h"
#define _XTAL_FREQ 40000000
#include <stdlib.h>
#include "tiempo.h"
#include <spi.h>
#define cs LATAbits.LATA2
#define tris_cs TRISAbits.RA2


#define SPI_SLAVE_ID    0x40
#define SPI_SLAVE_ADDR  0x00     // A2=0,A1=0,A0=0
#define SPI_SLAVE_WRITE 0x00
#define SPI_SLAVE_READ  0x01

// Registros para BANK=0 (por defecto)
#define IODIRA 0x00
#define IODIRB 0x01
#define IOCONA 0x0A
#define GPPUA  0x0C
#define GPPUB  0x0D
#define GPIOA  0x12
#define GPIOB  0x13


void SPI_Write(unsigned char addr,unsigned char data);
void lcd_command(unsigned char data);
void lcd_data(unsigned char data);
void init_display(void);
unsigned int i;
unsigned char cnt,togbutton,inp;
unsigned int idelay;
int main (void){

 //  OSCTUNE=0x40;
 //OSCCON=0x73;
 //TRISB=0x00;  //Puerto B (informacion) como salidas
 tris_cs=0; //Pin RA2 como salida del uC, entrada al chip, encargado de activarlo
 TRISCbits.TRISC5= 0; // RC5/SDO - Salida (Entrada serie de datos) En el ejemplo es 1
```

```
TRISCbits.TRISC3= 0;   // RC3/SCK - Salida (Reloj)
TRISCbits.TRISC4=1;    //RC4/SDI -Entrada (Aunque no está conectado)
LATCbits.LATC5=0;
LATCbits.LATC3=0;
LATCbits.LATC4=1;
cs=1;


 TRISFbits.TRISF6=0; //Reset
 PORTFbits.RF6=0;
 delay_ms(5);
 PORTFbits.RF6=1;


 SSPSTAT = 0x40;       // Poner SMP=0 and CKE=1. Los 6 pines más bajos son 0
 SSPCON1 = 0x21;       //  SPI Master con Fosc/64
 cs=1;    // Desactivar Chip Select




 SPI_Write(IOCONA,0x28);   // Registros de control de E/S: BANK=0, SEQOP=1,
                              HAEN=1 (Habilita el direccionamiento)
 SPI_Write(IODIRA,0x00);   // GPIOA como salida,  están conectados a los pines RS y E
 SPI_Write(IODIRB,0x00);   // GPIOB como salida, desde el cual se envía los datos a la
                             LCD
 SPI_Write(GPIOA,0x00);    // Reset de las salidas en GPIOA
 SPI_Write(GPIOB,0x00);    // Reset de las salidas en GPIOB
 delay_ms(30);

delay_ms(50);
 idelay=100;
init_display();
delay_ms(20);
lcd_data(0x4C);
delay_ms(2);
lcd_data(0x43);
delay_ms(2);
lcd_data(0x44);
delay_ms(2);
}
```

Funciones realizadas para la facilitación del programa:

```
void SPI_Write(unsigned char addr,unsigned char data)
{
 cs = 0;  //Se activa el chip select

 SSPBUF=SPI_SLAVE_ID;
 while(!SSPSTATbits.BF);
 while( !PIR1bits.SSPIF );  // esperar hasta ciclo de bus


 SSPBUF = addr;

 // Esperar hasta completar la transmisión
 while(!SSPSTATbits.BF);
 while( !PIR1bits.SSPIF );  // esperar hasta el ciclo del bus


 SSPBUF = data;

 while(!SSPSTATbits.BF);
 while( !PIR1bits.SSPIF );  // esperar ciclo de bus

 cs = 1;


}
void lcd_command(unsigned char data){
delay_ms(5);
SPI_Write(GPIOA,0x00); //rs=0,rw=0, e=0
delay_ms(2);
SPI_Write(GPIOB,data);  //se envía el comando
delay_ms(2);
SPI_Write(GPIOA,0x40); //rs=0,rw=0,e=1
delay_ms(2);
SPI_Write(GPIOA,0x00); //rs=0,rw=0, e=0
delay_ms(2);
}
void lcd_data(unsigned char data){
delay_ms(5);
SPI_Write(GPIOA,0x80) ;  //rs=1,rw=0,e=0
delay_ms(2);
```

```
SPI_Write(GPIOB,data); //se envía el comando
delay_ms(2);
SPI_Write(GPIOA,0xC0); //rs=1,rw=0,e=1
delay_ms(2);
SPI_Write(GPIOA,0x00); //rs=0,rw=0,e=0
delay_ms(2);
}

void init_display(void){
lcd_command (0x32);
delay_ms(5);
lcd_command(0x3C);
delay_ms(1);
lcd_command(0x0C);
delay_ms(1);
lcd_command(0x01);
delay_ms(1);
lcd_command(0x06);
delay_ms(1);

}
```

Anexo J. Programación PMOD TMP3.

Se utiliza el oscilador interno, que por defecto oscila a 4 MHz, y el módulo I2C del sensor funciona a 100 kHz, por lo tanto, se debe fijar este valor como se verá en el código *(ver tabla 1).*

```
#include <xc.h>
#include "config.h" //Incluimos la cabecera donde está incluida la configuración del
                    oscilador
#include "i2c.h" //Incluimos la librería del I2C
#define _XTAL_FREQ 4000000 //Frecuencia del oscilador
#include <stdlib.h>
#include "tiempo.h" //Incluimos la librería para poder realizar las esperas (delays).
```

Declaración de las variables:

```
unsigned char sync_mode=0, slew=0, addwrite, addread;
signed int status=-1;
unsigned int i=0;
unsigned char temp[2];
float temperatura=0;
```

```
unsigned char *temp_punt;
unsigned int alarma=0x1E; //Alarma a 30°C
```

Comienzo de la ejecución del programa:

```
int main (void){

TRISB=0x00; //Puerto B como salidas
TRISD=0x00;
TRISC=0x10;//Puerto C como salida menos RC4_SDA
PORTB=0; //Borra el puerto B
addwrite=0x90; //A0,A1 y A2 son 0 y el de escritura 0, 1001 0000
addread=0x91; // El ultimo es de Lectura 1, 1001 0001

CloseI2C();
sync_mode=MASTER;
slew=SLEW_OFF;
OpenI2C(sync_mode, slew);
SSPADD=0x27; //Fijar el clock a 100khz
while(1)
  {
       PORTDbits.RD0=1; //Encender LED para conocer el comienzo del programa
       delay_ms(500);
       PORTDbits.RD0=0;

  IdleI2C();
  StartI2C();

  while(SSPCON2bits.SEN );  //Indica el fin de la condición de Start
  do{
  status= WriteI2C(addwrite);
  if (status==-1){

    SSPCON1bits.WCOL=0; //Borra el bus de colisión
  }
  }
  while (status!=0);
  status=-1;
  delay_ms(80);

  do{
  status= WriteI2C(0x00);
```

```
if (status==-1){

   SSPCON1bits.WCOL=0; //Borra el bus de colision
}
}
while (status!=0);
status=-1;

IdleI2C();

RestartI2C(); //Condición de Stop + condición de Start

while(SSPCON2bits.RSEN );   //Esperar fin de restart
do{
status=WriteI2C(addread);
if (status==-1){
   SSPCON1bits.WCOL=0;
}
}
while(status!=0);
status=-1;
delay_ms(80);


temp_punt=temp;
getsI2C(temp_punt,2);


      __delay_ms (80);
      StopI2C();
```

A continuación, se procede a plasmar toda la información de la temperatura en una variable:

```
temp[0]=temp[0]<<1;

if (((temp[1]) & (128))>0)
{
   temp[0]=(temp[0] | 1);
}
else {
    temp[0]=(temp[0] & 254);
   }
```

```
   temperatura=temp[0]/2; //Resolución de 0,5ºC
if (temperatura>alarma){
      PORTDbits.RD1=1;
      delay_ms(1000);
     PORTDbits.RD1=0;
   }
         delay_ms(2000); //retarda 2s. para nueva lectura
}
}
```

Anexo K. Programación HIH6121.

La programación Software es la siguiente:

Primero se debe elegir el oscilador a utilizar (config.h), que en este caso será oscilador interno, que por defecto funcionará a 4 MHz (4.000.000 Hz).

Las siguientes son los archivos de cabecera que se añaden para facilitar la ejecución del código:

```
#include <xc.h>
#include "config.h" //Incluimos la cabecera donde está incluida la configuración del
                    oscilador
#include "i2c.h" //Incluimos la librería del I2C
#define _XTAL_FREQ 4000000 //Frecuencia del oscilador
#include <stdlib.h>
#include "tiempo.h" //Incluimos la librería para poder realizar las esperas (delays).
```

Las siguientes son las variables definidas:

```
unsigned char sync_mode=0, slew=0, addwrite, addread;
signed int status=-1;
unsigned int i=0, temperatura1=0, temperatura2=0;
float temperaturaini=0, temperatura=0, humedad=0;
unsigned char final=0;
unsigned char temp[4]; //Array en el que se guardarán los datos leídos
unsigned char *temp_punt; //Apuntados para la recogida de datos
```

unsigned int alarmat=0x1E, alarmah=0x60 //Alarmas para 30°C y 80%RH

Comienzo de la ejecución del Programa:

```
int main (void){

TRISB=0x00; //Puerto B como salidas
TRISC=0x10;//Puerto C como salida menos RC4_SDA
PORTB=0; //Borra el puerto B
addwrite=0x4E; //0 0100 1110 La direccion 0x27 mas el bit de escritura
addread=0x4F; // 0 0100 1111 La direccion 0x27 mas el bit de lectura


CloseI2C(); //Primero se cierra el programa por si se ha utilizado anteriormente
sync_mode=MASTER;
slew=SLEW_OFF;
OpenI2C(sync_mode, slew);
SSPADD=0x09; //Fijar el clock a 400kHz, ver tabla X
while(1)
  {
        PORTBbits.RB3=1; //Bit test en RB3, para observar en un LED que comienza la
ejecución
        delay_ms(500);
        PORTBbits.RB3=0;
  IdleI2C();
  StartI2C();
  while(SSPCON2bits.SEN );  //Indica el fin de la condición de Start
  do{
  status= WriteI2C(addwrite); //Enviamos la dirección mas el bit de escritura
  if (status==-1){

    SSPCON1bits.WCOL=0; //Borra el bus de colisión
  }
  }
  while (status!=0);
  status=-1;

  delay_ms(80);

  IdleI2C();

  RestartI2C();              //Reiniciamos el programa para poder leer la temperatura
```

```
                                    y humedad
while(SSPCON2bits.RSEN );   //Esperar fin de restart

do{
status=WriteI2C(addread); //Enviamos la dirección mas el bit de lectura
if (status==-1){
   SSPCON1bits.WCOL=0;
}
}
while(status!=0);
status=-1;
delay_ms(80);


temp_punt=temp;
getsI2C(temp_punt,4); //Leemos los datos y los introducimos en un puntero

      __delay_ms (80);
NotAckI2C();
      StopI2C();
__delay_ms(80);
```

Realizamos los cálculos a continuación

```
temperatura1=temp[3]>>2; //Desplaza el LSB de la temperatura a la derecha, puesto que los dos les significant bits son 0
```

A continuación, se realizan las conversiones para poner los dos less significant bits del MSB en los dos most significant bits del LSB:

```
if ((temp[2] & 1)>0)
{
   temperatura1=(temperatura1 | 64);
}
else {
    temperatura1=(temperatura1 & 191);
   }
if ((temp[2] & 2)>0)
{
   temperatura1=(temperatura1 | 128);
}
else {
```

```
      temperatura1=(temperatura1 & 127);
     }
  temperatura2=temp[2]>>2;
  temperaturaini=temperatura1+(temperatura2*256);
  temperatura=0;
  temperatura=((temperaturaini/(16384.0-2.0))*165)-40;
  humedad=temp[1]+(temp[0]*256);
  humedad=(humedad/(16384-2))*100;
 if (temperatura>alarmat || humedad>alarmah)
{
PORTDbits.RD1=1;
delay_ms(1000);
PORTDbits.RD1=0;
}
        delay_ms(2000);
}
}
```

Anexo L. Programación MPX2050DP.

El siguiente es el software utilizado:

Librerías incluidas en el proyecto:

```
#include <xc.h>                    //Librería de funciones del microcontrolador
#include "config.h"                //Librería de configuración del oscilador
#define _XTAL_FREQ 4000000         // Frecuencia del oscilador
#include <stdlib.h>
#include <adc.h>                   //Librería del convertidor analógico-digital
#include "tiempo.h"                //Librería para los retardos.
```

Definimos las variables a utilizar:

```
unsigned int sensor;
```

```
unsigned char channel=0x00, config1=0x00, config2=0x00, config3=0x00, portconfig=0x00, i=0;
float valor;
```

Comienzo de la ejecución del programa:

```
void main (int){

TRISA=0xFF;          //Puerto A como entradas para poder capturar la tensión del
                     sensor, que es un valor analógico que después convertiremos a
                     digital.
TRISD=0x00;

CloseADC();

config1 = ADC_FOSC_2 | ADC_RIGHT_JUST | ADC_2_TAD ;
config2 = ADC_CH0 | ADC_INT_ON | ADC_REF_VDD_VSS ;
portconfig = ADC_15ANA ;
OpenADC(config1,config2,portconfig);
ADC_INT_ENABLE();
PORTDbits.RD0=1;     //Encender un LED 0,5 segundos para detectar el comienzo del
                      programa
delay_ms(500);
PORTDbits.RD0=0;
   while(1){

      for(i=0;i<16;i++)
      {
         ConvertADC();
         while(BusyADC());
         sensor += (unsigned int) ReadADC();
      }
      sensor /= 16;
      valor = (sensor*30.0);              // convertir el valor
  CloseADC();
     __delay_ms(20);
   }
}
```

Anexo M. Programación MMA8421Q.

Las librerias includas en el proyecto:

```
#include <xc.h>
#include "config.h" //Incluimos la cabecera donde está incluida la configuración del oscilador
#include "i2c.h" //Incluimos la librería del I2C
#define _XTAL_FREQ 4000000 //Frecuencia del oscilador
#include <stdlib.h>
#include "tiempo.h" //Incluimos la librería para poder realizar las esperas (delays).
```

Declaración de las variables:

```
unsigned char sync_mode=0, slew=0, addwriteposicion, addreadposicion, MSB, LSB;
signed int status=-1;
unsigned int i=0;
float posicion;
unsigned int X1=0, X2=0, Y1=0, Y2=0, Z1=0, Z2=0;
float X, Y, Z;
unsigned char pres[6];
unsigned char *pres_punt;
```

Comienzo de la ejecución del programa:

```
int main (void){

TRISB=0x00; //Puerto B como salidas
TRISC=0x10;//Puerto C como salida menos RC4_SDA
PORTB=0; //Borra el puerto B
addwriteposicion=0x3A; //0011 1000
addreadposicion=0x3B; // 0011 1001


CloseI2C();
sync_mode=MASTER;
slew=SLEW_OFF;
OpenI2C(sync_mode, slew);
SSPADD=0x09; //Fijar el clock a 400 kHz, ver hoja de características del sensor
```

```
while(1)
  {
        PORTDbits.RD0=1; //Encender LED para comienzo del programa
        delay_ms(500);
        PORTDbits.RD0=0;
  IdleI2C();
  StartI2C();
  while(SSPCON2bits.SEN );  //Indica el fin de la condicion de Start
  //MSSPbits.SSPIF = 0;   //Borrar el flag
  do{
  status= WriteI2C(addwrite);
  if (status==-1){

     SSPCON1bits.WCOL=0; //Borra el bus de colision
  }
  }
  while (status!=0);
  status=-1;
  //while(SSPSTATbits.BF); //Esperar a que el buffer este vacio
  //SSPSRbits.IF;
 // while(SSPCON2bits.ACKSTAT); //Esperar llegada ACK

  delay_ms(80);
  do{
  status= WriteI2C(0x01);
  if (status==-1){

     SSPCON1bits.WCOL=0; //Borra el bus de colision
  }
  }
  while (status!=0);
  status=-1;

  IdleI2C();

  RestartI2C();

  while(SSPCON2bits.RSEN );   //Esperar fin de restart
  //Write(0x00)????
  do{
  status=WriteI2C(addread);
```

```
if (status==-1){
   SSPCON1bits.WCOL=0;
}
}
while(status!=0);
status=-1;
delay_ms(80);


pres_punt=pres;
getsI2C(pres_punt,6);

__delay_ms (80);
NotAckI2C();
      StopI2C();
```

A continuación, se realizan los cálculos realizados para la obtención de las posiciones en los bytes correspondientes, ya que, la posición de cada uno de los ejes está separado en dos posiciones y los dos less significant bits del LSB son 0.

```
X1=pres[1]>>2;  //Eje X
if ((pres[0] & 1)>0)
{
   X1=(X1 | 64);
}
else {
    X1=(X1 & 191);
   }
if ((pres[0] & 2)>0)
{
   X1=(X1 | 128);
}
else {
    X1=(X1 & 127);
   }
X2=pres[0]>>2;
X=X1+(X2*256);


 Y1=pres[3]>>2;  //Eje Y
if ((pres[2] & 1)>0)
{
```

```
    Y1=(Y1 | 64);
  }
  else {
      Y1=(Y1 & 191);
     }
  if ((pres[2] & 2)>0)
  {
    Y1=(Y1 | 128);
  }
  else {
      Y1=(Y1 & 127);
     }
  Y2=pres[2]>>2;
  Y=Y1+(Y2*256);

   Z1=pres[5]>>2;  //Eje Z
  if ((pres[4] & 1)>0)
  {
    Z1=(Z1 | 64);
  }
  else {
      Z1=(Z1 & 191);
     }
  if ((pres[4] & 2)>0)
  {
    Z1=(Z1 | 128);
  }
  else {
      Z1=(Z1 & 127);
     }
  Z2=pres[4]>>2;
  Z=Z1+(Z2*256);

        delay_ms(2000);
}
}
```

Anexo N. Programación PMOD TMP3 y HIH 6121.

```
#include <xc.h>
#include "config.h"
```

```
#include "i2c.h"
#define _XTAL_FREQ 4000000
#include <stdlib.h>
#include "tiempo.h"

int hum1=0, hum2=0, temp1=0, temp2=0, tempohum=0; // Variables para la recogida
                                                     de temperatura

unsigned char sync_mode=0, slew=0, addwritetemp=0x90, addreadtemp=0x91,
addwritehum=0x4E, addreadhum=0x4F;
signed int status=-1;
unsigned int i=0, temperatura1=0, temperatura2=0;
float temperaturaini=0, temperatura=0, temperaturasolo=0, humedad=0;
unsigned char final=0;
unsigned char temp[4], tempsolo[2];
unsigned char *temp_punt, tempsolo_punt;


int main (void){

TRISD=0x00;
TRISB=0x00; //Puerto B como salidas
TRISC=0x10;//Puerto C como salidas menos RC4_SDA
PORTB=0; //Borra el puerto B
```

El programa primero mide la temperatura y humedad con el sensor HIH 6121:

```
while(tempohum==0) //Temperatura y Humedad
  {
  CloseI2C();
  sync_mode=MASTER;
  slew=SLEW_OFF;
  OpenI2C(sync_mode, slew);
        SSPADD=0x09; //Fijar el clock a 400kHz
  PORTDbits.RD1=1; //LED en RD1 para detectar el comienzo
        delay_ms(500);
        PORTDbits.RD3=0;
  IdleI2C();
  StartI2C();
  while(SSPCON2bits.SEN );  //Indica el fin de la condicion de Start
  do{
  status= WriteI2C(addwritehum);
```

```
if (status==-1){

   SSPCON1bits.WCOL=0; //Borra el bus de colision
}
}
while (status!=0);
status=-1;

delay_ms(80);

IdleI2C();
RestartI2C();
while(SSPCON2bits.RSEN );   //Esperar fin de restart
do{
status=WriteI2C(addreadhum);
if (status==-1){
   SSPCON1bits.WCOL=0;
}
}
while(status!=0);
status=-1;
delay_ms(80);


temp_punt=temp;
getsI2C(temp_punt,4);


      __delay_ms (80);
NotAckI2C();
      StopI2C();
__delay_ms(80);
```

Realizamos los calculos a continuación:

```
temperatura1=temp[3]>>2;
if ((temp[2] & 1)>0)
{
   temperatura1=(temperatura1 | 64);
}
else {
    temperatura1=(temperatura1 & 191);
```

```
    }
  if ((temp[2] & 2)>0)
  {
    temperatura1=(temperatura1 | 128);
  }
  else {
     temperatura1=(temperatura1 & 127);
    }
  temperatura2=temp[2]>>2;
  temperaturaini=temperatura1+(temperatura2*256);
  temperatura=0;
  temperatura=((temperaturaini/(16384.0-2.0))*165)-40;
  humedad=temp[1]+(temp[0]*256);
  humedad=(humedad/(16384-2))*100;

       delay_ms(2000);
  if (PORTAbits.RA5==0){
  tempohum=1; //De primeras mide temp y hum, si pulsas RA5 mide temp
}
  tempohum=1;


       delay_ms(2000); //retarda 2s. para nueva lectura
}
```

Ahora procedemos a calcular la temperatura con el PMOD TMP3:

```
while(tempohum==1)  //Temperatura
  {
  CloseI2C();
  sync_mode=MASTER;
  slew=SLEW_OFF;
  OpenI2C(sync_mode, slew);
       SSPADD=0x27; //Fijar el clock a 100Mhz
  PORTD=0x00;
  PORTDbits.RD0=1; //Bit test en RB3
       delay_ms(500);
       PORTDbits.RD0=0;
  IdleI2C();
  StartI2C();
  while(SSPCON2bits.SEN );  //Indica el fin de la condicion de Start
  //MSSPbits.SSPIF = 0;   //Borrar el flag
  do{
```

```
status= WriteI2C(addwritetemp);
if (status==-1){

   SSPCON1bits.WCOL=0; //Borra el bus de colision
}
}
while (status!=0);
status=-1;

delay_ms(80);
do{
status= WriteI2C(0x00);
if (status==-1){

   SSPCON1bits.WCOL=0; //Borra el bus de colision
}
}
while (status!=0);
status=-1;

IdleI2C();
RestartI2C();
while(SSPCON2bits.RSEN );   //Esperar fin de restart
//Write(0x00)????
do{
status=WriteI2C(addreadtemp);
if (status==-1){
   SSPCON1bits.WCOL=0;
}
}
while(status!=0);
status=-1;
delay_ms(80);

tempsolo_punt=tempsolo;
getsI2C(tempsolo_punt,2);
       __delay_ms (80);
       StopI2C();
tempsolo[0]=tempsolo[0]<<1;

if (((tempsolo[1]) & (128))>0)  //bit D1
{
```

```
    tempsolo[0]=(tempsolo[0] | 1);
  }
  else {
      tempsolo[0]=(tempsolo[0] & 254);
     }

  temperaturasolo=tempsolo[0]/2; //Resolucion de 0,5ºC

       delay_ms(2000); //retarda 2s. para nueva lectura

  tempohum=0;
}
}
```

Anexo O. Programación todos los transductores.

```
#include <xc.h>
#include "config.h"
#include "i2c.h"
#include "adc.h"
#define _XTAL_FREQ 4000000
#include <stdlib.h>
#include "tiempo.h"

unsigned char sync_mode=0, slew=0, addtempwrite=0x90, addtempread=0x91,
addthwrite=0x4E, addthread=0x4F, addwriteposicion=0x3A, addreadposicion=0x3B;
unsigned char channel=0x00, config1=0x00, config2=0x00, config3=0x00,
portconfig=0x00, MSB, LSB;
signed int status=-1, X1, X2, Y1, Y2, Z1, Z2;
int alarma=0x23, alarmat=0x1E, alarmah=0x60;
unsigned int i=0, sensor, temperatura1, temperatura2, temperaturaini;
unsigned char temp[2], hum[4], pos[6];
float temperatura=0, temperaturah=0, humedad=0, valor, X, Y, Z, posicion;
unsigned char *temp_punt, *hum_punt, *pos_punt;
int transductor=0;
int main (void){
  if (transductor==0){
TRISB=0x00; //Puerto B como salidas
TRISD=0x00;
TRISC=0x10;//Puerto C como salida menos RC4_SDA
```

```
PORTB=0; //Borra el puerto B



CloseI2C();
sync_mode=MASTER;
slew=SLEW_OFF;
OpenI2C(sync_mode, slew);
SSPADD=0x27; //Fijar el clock a 100Mhz
while(1)
  {
       PORTDbits.RD0=1; //Bit test en RD0
       delay_ms(500);
       PORTDbits.RD0=0;
  IdleI2C();
  StartI2C();
  while(SSPCON2bits.SEN );  //Indica el fin de la condicion de Start
  do{
  status= WriteI2C(addtempwrite);
  if (status==-1){

    SSPCON1bits.WCOL=0; //Borra el bus de colision
  }
  }
  while (status!=0);
  status=-1;


  delay_ms(80);
  do{
  status= WriteI2C(0x00);
  if (status==-1){

    SSPCON1bits.WCOL=0; //Borra el bus de colision
  }
  }
  while (status!=0);
  status=-1;

  IdleI2C();

  RestartI2C();
```

```
  while(SSPCON2bits.RSEN );   //Esperar fin de restart
  do{
  status=WriteI2C(addtempread);
  if (status==-1){
     SSPCON1bits.WCOL=0;
  }
  }
  while(status!=0);
  status=-1;
  delay_ms(80);


  temp_punt=temp;
  getsI2C(temp_punt,2);

        __delay_ms (80);
        StopI2C();
  temp[0]=temp[0]<<1;

  if (((temp[1]) & (128))>0)  //bit D1
  {
     temp[0]=(temp[0] | 1);
  }
  else {
      temp[0]=(temp[0] & 254);
     }

  temperatura=temp[0]/2; //Resolucion de 0,5ºC
  if (temperatura>alarma){
     PORTDbits.RD1=1;
     delay_ms(1000);
     PORTDbits.RD1=0;
  }
        delay_ms(2000); //retarda 2s. para nueva lectura
  transductor++;
}
}
  if (transductor==1){
     TRISB=0x00; //Puerto B como salidas
TRISC=0x10;//Puerto C como salida menos RC4_SDA
PORTB=0; //Borra el puerto B
```

```
CloseI2C();
sync_mode=MASTER;
slew=SLEW_OFF;
OpenI2C(sync_mode, slew);
SSPADD=0x09; //Fijar el clock a 400kHz
while(1)
  {
       PORTBbits.RB3=1; //Bit test en RB3
       delay_ms(500);
       PORTBbits.RB3=0;
  IdleI2C();
  StartI2C();
  while(SSPCON2bits.SEN );  //Indica el fin de la condicion de Start
  do{
  status= WriteI2C(addthwrite);
  if (status==-1){

    SSPCON1bits.WCOL=0; //Borra el bus de colision
  }
  }
  while (status!=0);
  status=-1;

  delay_ms(80);

  IdleI2C();

  RestartI2C();
  while(SSPCON2bits.RSEN );   //Esperar fin de restart

  do{
  status=WriteI2C(addthread);
  if (status==-1){
    SSPCON1bits.WCOL=0;
  }
  }
  while(status!=0);
  status=-1;
  delay_ms(80);
```

```
  hum_punt=hum;
  getsI2C(hum_punt,4);

        __delay_ms (80);
  NotAckI2C();
        StopI2C();
  __delay_ms(80);

  //Realizamos los calculos a continuacion

  temperatura1=hum[3]>>2;

  if ((hum[2] & 1)>0)
  {
    temperatura1=(temperatura1 | 64);
  }
  else {
      temperatura1=(temperatura1 & 191);
     }
  if ((hum[2] & 2)>0)
  {
    temperatura1=(temperatura1 | 128);
  }
  else {
      temperatura1=(temperatura1 & 127);
     }
  temperatura2=hum[2]>>2;
  temperaturaini=temperatura1+(temperatura2*256);
  temperatura=0;
  temperatura=((temperaturaini/(16384.0-2.0))*165)-40;
  humedad=temp[1]+(hum[0]*256);
  humedad=(humedad/(16384-2))*100;

        delay_ms(2000);
  transductor++;
}
}
  if (transductor==2){
TRISA=0xFF;  //Puerto A como entradas
TRISD=0x00;

CloseADC();
```

```
config1 = ADC_FOSC_2 | ADC_RIGHT_JUST | ADC_2_TAD ;
config2 = ADC_CH0 | ADC_INT_ON | ADC_REF_VDD_VSS ;
portconfig = ADC_15ANA ;
OpenADC(config1,config2,portconfig);
ADC_INT_ENABLE();


   while(1){

      for(i=0;i<16;i++)
      {
         ConvertADC();
         while(BusyADC());
         sensor += (unsigned int) ReadADC();
      }
      sensor /= 16;
      valor = (sensor*30.0); // convert ADC count into voltage


  CloseADC();

      __delay_ms(20);
      transductor++;
   }
}

   if (transductor==3){
      TRISB=0x00; //Puerto B como salidas
TRISC=0x10;//Puerto C como salida menos RC4_SDA
PORTB=0; //Borra el puerto B


CloseI2C();
sync_mode=MASTER;
slew=SLEW_OFF;
OpenI2C(sync_mode, slew);
SSPADD=0x09; //Fijar el clock a 400 kHz
while(1)
  {
        PORTDbits.RD0=1; //Bit test en RB3
        delay_ms(500);
        PORTDbits.RD0=0;
```

```
IdleI2C();
StartI2C();
while(SSPCON2bits.SEN );  //Indica el fin de la condicion de Start
do{
status= WriteI2C(addwriteposicion);
if (status==-1){

   SSPCON1bits.WCOL=0; //Borra el bus de colision
}
}
while (status!=0);
status=-1;


delay_ms(80);
do{
status= WriteI2C(0x01);
if (status==-1){

   SSPCON1bits.WCOL=0; //Borra el bus de colision
}
}
while (status!=0);
status=-1;

IdleI2C();

RestartI2C();

while(SSPCON2bits.RSEN );   //Esperar fin de restart
//Write(0x00)????
do{
status=WriteI2C(addreadposicion);
if (status==-1){
   SSPCON1bits.WCOL=0;
}
}
while(status!=0);
status=-1;
delay_ms(80);
```

```
pos_punt=pos;
getsI2C(pos_punt,6);

      __delay_ms (80);
NotAckI2C();
      StopI2C();


X1=pos[1]>>2;

if ((pos[0] & 1)>0)
{
  X1=(X1 | 64);
}
else {
    X1=(X1 & 191);
   }
if ((pos[0] & 2)>0)
{
  X1=(X1 | 128);
}
else {
    X1=(X1 & 127);
   }
X2=pos[0]>>2;
X=X1+(X2*256);



 Y1=pos[3]>>2;

if ((pos[2] & 1)>0)
{
  Y1=(Y1 | 64);
}
else {
    Y1=(Y1 & 191);
   }
if ((pos[2] & 2)>0)
{
  Y1=(Y1 | 128);
}
```

```
  else {
      Y1=(Y1 & 127);
     }
  Y2=pos[2]>>2;
  Y=Y1+(Y2*256);

    Z1=pos[5]>>2;

  if ((pos[4] & 1)>0)
  {
    Z1=(Z1 | 64);
  }
  else {
      Z1=(Z1 & 191);
     }
  if ((pos[4] & 2)>0)
  {
    Z1=(Z1 | 128);
  }
  else {
      Z1=(Z1 & 127);
     }
  Z2=pos[4]>>2;
  Z=Z1+(Z2*256);

        delay_ms(2000); //retarda 2s. para nueva lectura
}
}
}
```