Equipo A Proyecto PIC16F873A

Stefan Costea
Raul Mestre
Pablo G. Segarra
Pablo G. Cabo

Febrero 2022

Índice general

Capítulo 1

Organización

Para poder funcionar como equipo y realizar la tarea que nos habían encomendado era esencial coordinarnos eficientemente. Para esto decidimos utilizar diferentes he-herramientas como se explicara a continuación.
Este apartado fue el que, al principio, más nos costó, pero una vez arrancamos todo funciono de una forma fluida y correcta.

1.1. Herramientas de comunicación

La comunicación del equipo es un apartado vital para asegurar que todo el mundo sabe en todo momento en que debe trabajar, para estar informado de las últimas actualizaciones del proyecto y para estar atento a reuniones de urgencia o problemas de última hora.

Para esto decidimos crear un grupo de WhatsApp, ideal para una comunicación de carácter general y rápida, pues consideramos que esta era la forma más eficaz de asegurarnos que todos los miembros se enterasen de las últimas actualizaciones y novedades. Para realizar las reuniones utilizamos el programa Discord, se trata de una herramienta muy versátil para la comunicación de equipos y comunidades. En este programa creamos un servidor con apartados dedicados a cada subgrupo del equipo, así nos aseguramos de que, por ejemplo, ningún integrante de Hardware se perdiese ningún mensaje de este apartado.
Se puede acceder al Discord del equipo a través del siguiente enlace: https://discord.gg/wxcYY9tbN9.

1.2. Herramientas de colaboración

Para la colaboración entre los miembros del equipo decidimos utilizar la herramienta GitHub, pues es ampliamente conocida, es eficaz y fácil de usar. Utilizamos esta herramienta para subir todos los archivos del proyecto, en él esta se encuentra el software de la página web, el código del firmware y los documentos del hardware como el schematic.
Se puede acceder al GitHub del equipo a través del siguiente enlace:
https://github.com/PredaWina/Hyperloop-Equipo-A.

1.3. Documentación del proyecto

Para la documentación del equipo decidimos usar LaTeX, de nuevo una herramienta muy conocida. Esta herramienta es especialmente eficaz en este tipo de tareas y nos brinda un resultado profesional, necesario para este tipo de tareas. Para que todos los integrantes pudiesen colaborar en la documentación utilizamos una página web que permite la colaboración online y simultánea en documentos LaTeX.
Esta página web se llama OverLeaf y se puede acceder a nuestro documento mediante el siguiente enlace:
https://es.overleaf.com/9679599333dgcsbzrrjmvn

Capítulo 2

Hardware

2.1. Introducción Esquemático

El microcontrolador escogido para realizar el proyecto es el PIC16F873A, tiene un voltaje de funcionamiento de 5V, posee un oscilador interno y se puede conectar a uno externo que tenga como máximo 20MHz. Dispone de varios protocolos de comunicación tanto en serie como en paralelo, como por ejemplo I^2C, USART o SPI. Para este proyecto usaremos la comunicación SPI. El microcontrolador dispone también de un Conversor Analógico-Digital (ADC) de 10 bits.

El objetivo del proyecto es diseñar un esquemático que incorpore una resistencia NTC (Negative Temperature Coefficient) para implementar un sensor de temperatura, la NTC es un tipo de resistencia la cual disminuye su resistividad progresivamente al aumentar la temperatura a la que se expone, haciéndola muy práctica para diseñar este tipo de circuitos.

El circuito con la NTC se alimentará a 5V y relación entre este voltaje y la salida variará según el valor de la resistividad de la misma gracias a la disposición de divisor de tensión. Esa señal de salida será transmitida al ADC del microcontrolador el cual la convertirá en una señal digital que será almacenada en un registro y más adelante transmitida a través del protocolo SPI por los canales correspondientes del microcontrolador.

La señal SPI será recibida por una Raspberry Pi la cual se encargará de mandar los valores de la temperatura a un servidor web y mostrarla por un diseño web realizado por el equipo de Software.

En este capítulo se explicará más a fondo el funcionamiento de las diferentes partes del esquemático así como elaborarse un presupuesto del coste de elaboración del mismo.

El circuito completo esquematizado tiene la siguiente apariencia:

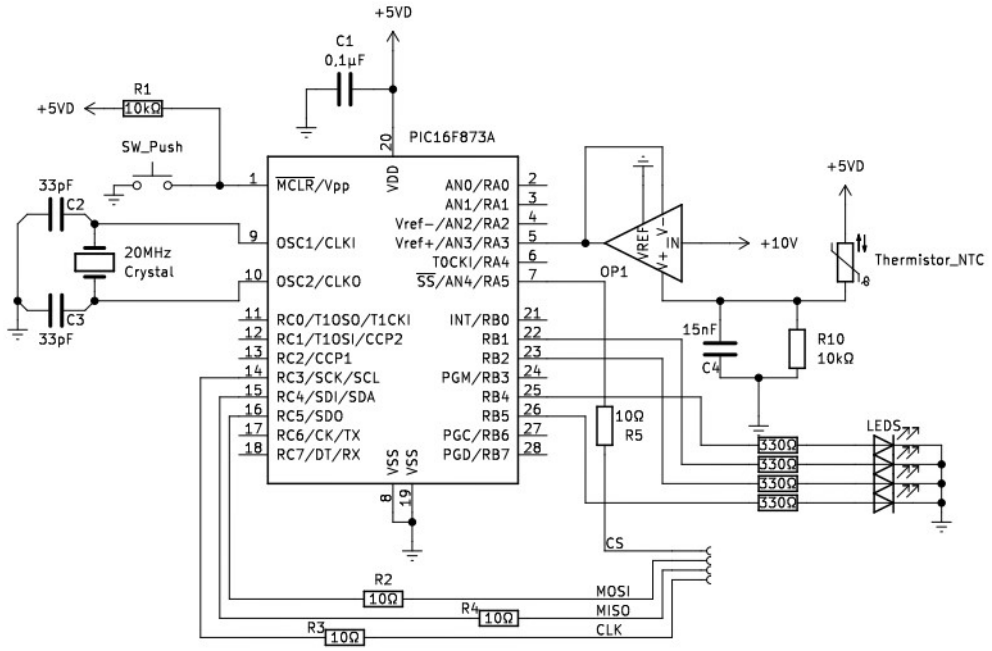


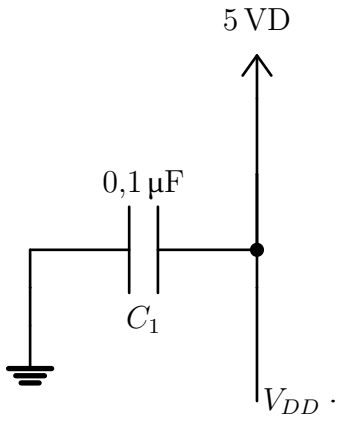
Ilustración 2.1: Esquemático del circuito elaborado

Explicación Esquemático

2.2. Alimentación/VDD

Para la alimentación del circuito necesitaremos dos fuentes de alimentación consistentes de 5V y de 10V en corriente continua. La mayoría del circuito estará alimentado a 5V excepto el amplificador operacional usado en la entrada del ADC del microcontrolador que se alimentará entre 10V y masa.

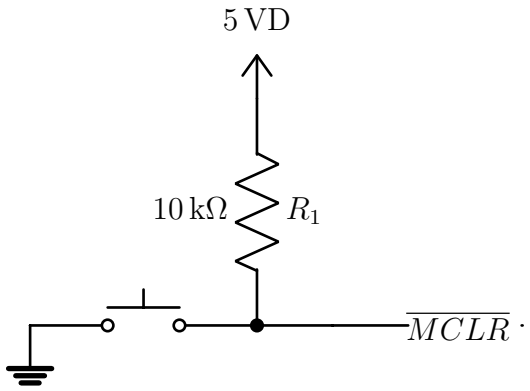
En las entradas de alimentación se han colocado unos condensadores de desacoplo de 0,1µF para prevenir posibles picos o caídas de tensión repentinas en la entrada del microcontrolador por culpa de la fuente. La capacidad del condensador venía dada por el fabricante del microcontrolador en su hoja de características, y para la alimentación del amplificador operacional se ha optado por el mismo valor por facilidad de compra. Cuando hay una subida de tensión el condensador se carga mientras que cuando hay una bajada el condensador se descarga, manteniendo el nivel de alimentación prácticamente constante ante perturbaciones puntuales.



Condensador de desacoplo en VDD

2.3. Circuito de reset/ $\overline{MCLR}$

El Pin 1 o $\overline{MCLR}$ se emplea para reiniciar el microcontrolador. Este se reinicia a nivel bajo, por lo tanto, para que el microcontrolador opere debe estar normalmente a 5V y cuando quiera accionarse un reset debe colocarse a 0V. Para ello se ha empleado un circuito con una resistencia pull-up y un pulsador.
El botón mantiene el circuito normalmente abierto, es estos casos, como la entrada al pin tiene impedancia infinita, la corriente por el circuito sería despreciable y en la resistencia no caería tensión, eso querrá decir que habrá 5V en la entrada al microcontrolador. Mientras que si el botón se accionase, toda la tensión caería en la resistencia pull-up y el pin $\overline{MCLR}$ bajaría a 0V, ya que estaría conectado a masa, reiniciando así el microcontrolador.
La impedancia de entrada de los pines es muy grande, pero no infinita en realidad, esto limita los valores de las resistencias pull-up para que se considere despreciable la caída de tensión cuando el pulsador esté abierto, a la vez que no ha de ser demasiado pequeña para que la corriente dañe el circuito cuando se cierra el pulsador por lo que hay un rango de valores de resistencias para esta aplicación, pero el fabricante nos recomienda, a 5V que se utilicen resistencias de pull-up de $10\,\mathrm{k}\Omega$.

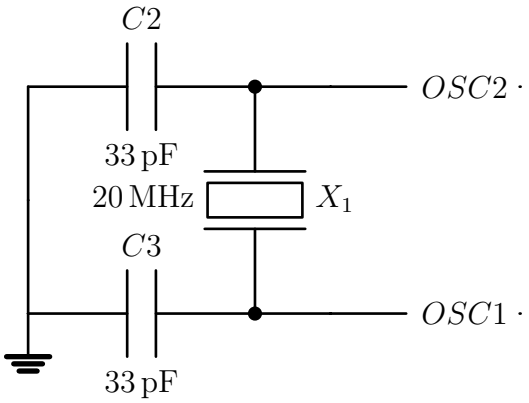


Circuito de reset con resistencia pull-up

2.4. Oscilador Externo

A pesar de que el microcontrolador posee un oscilador interno se ha optado por la opción de implementar uno externo para reducir la carga de trabajo del microcontrolador, además que se obtiene más precisión en los pulsos de reloj con este método. Se ha implementado el circuito re-

comendado en la hoja de características para osciladores externos, que requiere de un cristal y dos condensadores. La frecuencia inicial escogida para el cristal era de $32\,\mathrm{kHz}$, pero dado que no era suficiente para que el ADC del microcontrolador operase correctamente se cambió a uno de $20\,\mathrm{MHz}$, que es la frecuencia máxima que permite el microcontrolador. Los condensadores son ambos de $33\,\mathrm{pF}$ que según el fabricante se habían probado y funcionaban correctamente con cristales de dicha frecuencia. El circuito implementado es el siguiente:



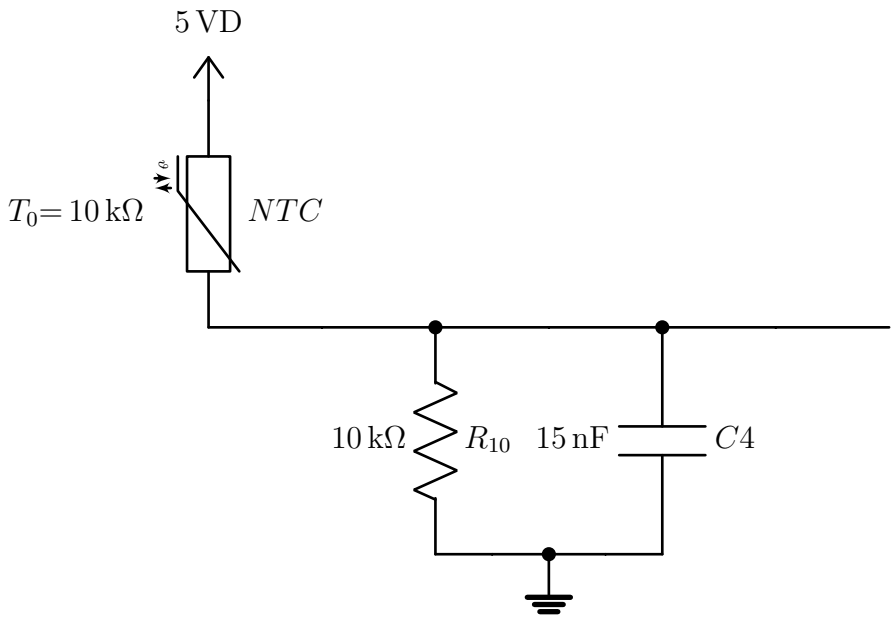
Ciruicto de oscilador externo

2.5. Filtro RC Paso Bajo

A través de la resistencia NTC circula una corriente de $5\,\mathrm{V}$. Dependiendo de la temperatura del ambiente en el que se encuentre tendrá una resistencia mayor o menor, inversamente proporcional a la temperatura. La NTC conforma un divisor de tensión con la resistencia R10 repartiéndose la tensión proporcionalmente. El filtro tiene la capacidad de dejar pasar las frecuencias bajas, pero elimina las altas frecuencias que consideramos ruido proveniente del sensor. A frecuencias altas la impedancia del condensador C4 aumenta restringiendo el paso de la señal mientras que a frecuencias bajas es más pequeña, permitiendo su paso. El valor de la resistencia R10 se escogió para que coincidiera con la resistencia base de la NTC y el condensador fue calculado con la ecuación para un filtro de paso bajo imponiendo que nuestra frecuencia de corte fuese de $1\,\mathrm{kHz}$. El resultado venía dado por la siguiente ecuación:

$$f_c = \frac{1}{2\pi RC}$$

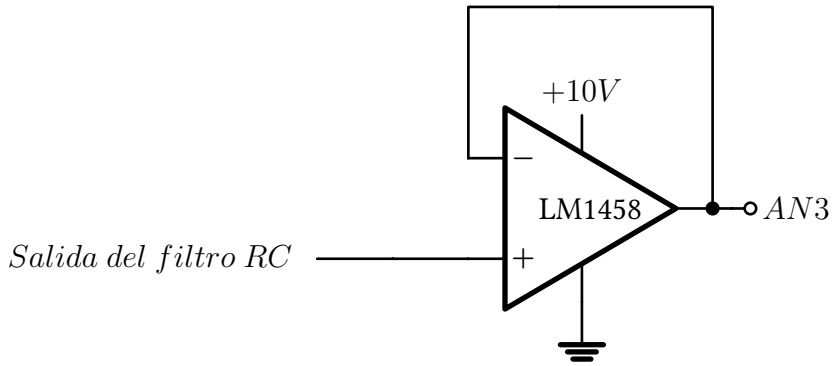
$$C = \frac{1}{2\pi R f_c}$$



Circuito de filtro RC de paso bajo

2.6. Amplificador Operacional Seguidor de tensión

Tras el filtro RC se coloca un amplificador operacional puesto en la configuración de seguidor de tensión. En esta configuración se retroalimenta la entrada inversora. Es un caso especial del operacional en configuración no inversora en el que la ganancia total es 1, por lo tanto, la señal original no se ve ni amplificada ni reducida. La impedancia de entrada es muy alta, casi infinita, mientras que la impedancia de salida es muy baja, aproximadamente $0\,\Omega$. Esto es muy útil, ya que la entrada al ADC del microcontrolador solo acepta impedancias hasta $2{,}5\,\mathrm{k}\Omega$ según consta en el datasheet. La entrada de voltaje de referencia positiva del operacional se coloca a $10\,\mathrm{V}$ para que la bajada de tensión que hay en el operacional no afecte a la señal. Experimentando en un simulador se ha comprobado que a V_{REF+} = $5\,\mathrm{V}$ cuando la bajada de tensión en la NTC es muy pequeña el operacional no saca más de $3\,\mathrm{V}$, con el V_{REF+} = $10\,\mathrm{V}$ el problema se soluciona.



Amplificador operacional en modo seguidor

2.7. Presupuesto

A continuación se detalla el precio de todos los componentes utilizados y el coste total de todos estos.

Detalles	Cantidad	Precio/Ud.	Precio
PIC16F873A 28 pines Microchip Technology *	1	4.82€	4.82€
Crystal 20MHz ECS-200-10-37B2-JTN-TR ECS*	1	0.39€	0.39€
NTC 10kΩ Vishay/BC Components*	1	1.23€	1.23€
Botón interruptor Schurter*	1	0.43€	0.43€
LED rojo Cree LED CLM2D-RCC-CZ0C0BB3*	4	0.14€	0.56€
Amplificador Operacional LM1458MX/NOPB*	1	1.08€	1.08€
Condensador 33 pF GRM21A5C2J330FWA1D (Cristal)*	2	0.35€	0.70€
Condensador 15nF C0805C153K5GECTU (Filtro RC)*	1	0.31€	0.31€
Condensador 0,1µF C0805X104M5RACTU (Alimentación)*	2	0.42€	0.84€
Resistencia 10kΩ ERA-6VRW1002V (Reset y filtro RC)*	2	0.71€	1.42€
Resistencia 330Ω ERJ-U6RD3300V (LEDs)*	4	0.29€	1.16€
Resistencia 10Ω CRGCQ0805F10R*	3	0.24€	0.72€
Coste total:			13.66€

*Los asteriscos muestran hipervínculos a la página de venta de los componentes

Capítulo 3

Software

3.1. Primeras versiones de la página.

El objetivo inicial era disponer de un sitio web para mostrar los datos obtenidos por el sensor de temperatura. Por eso se optó al principio por un diseño simple y conciso como se puede ver en la siguiente figura.

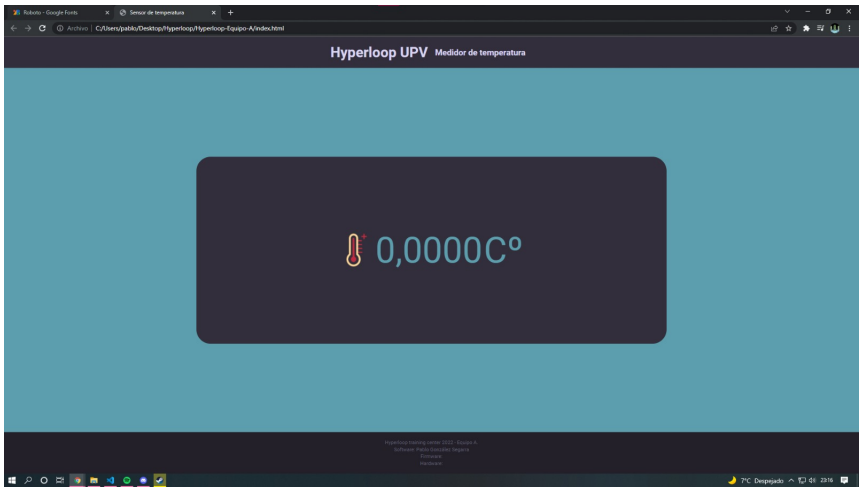


Ilustración 3.1: Primera versión de la web.

Más adelante se hizo un cambio en los colores de la página para intentar que fuese más estética. Se eligió usar un contraste entre claros y oscuros como se puede observar en la siguiente figura.

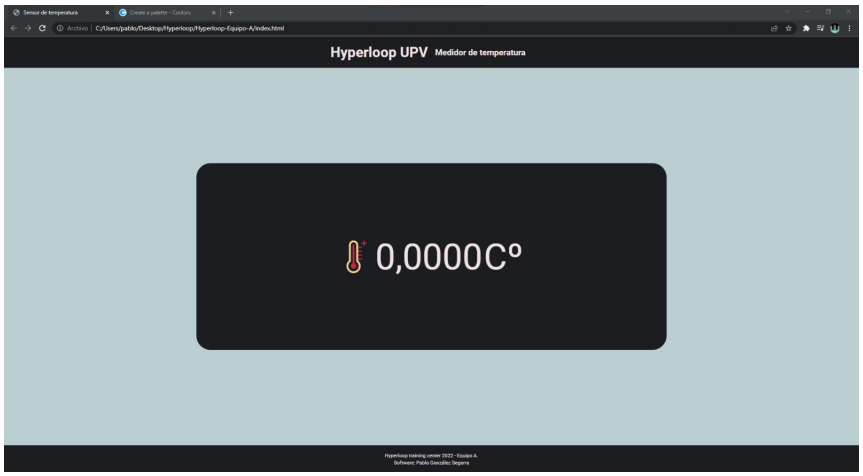


Ilustración 3.2: Segunda versión de la web.

Con el objetivo de poder mostrar más información, es decir, la temperatura máxima y mínima registrada, se añadieron dos contadores nuevos. Además, también pensamos en hacerle un hueco al hardware, así que decimos incluir una imagen del schematic diseñado por los miembros de Hardware.

Mientras diseñábamos esta página pensamos que sería útil disponer de un botón que nos permitiese pausar o reanudar la recepción de datos. De esta forma decidimos incluirlo debajo de los contadores. El resultado se puede observar en la siguiente figura.

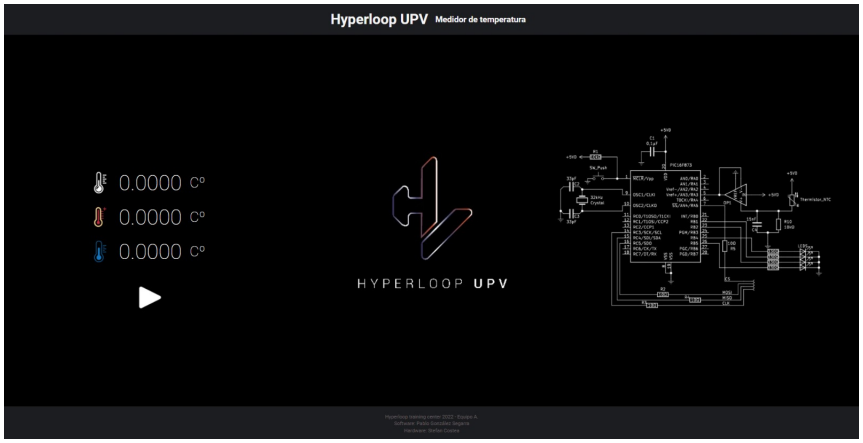


Ilustración 3.3: Tercera versión de la web.

3.2. Versión actual de la página.

Finalmente, nos decantamos por hacer que la página fuese algo más que un simple archivo html para mostrar los datos. Así que tomamos la decisión de hacer dos apartados, el primero, dedicado únicamente para mostrar los datos obtenidos, el segundo, dividido en otros tres, dedicado a mostrar los integrantes del equipo y sus datos de contacto.

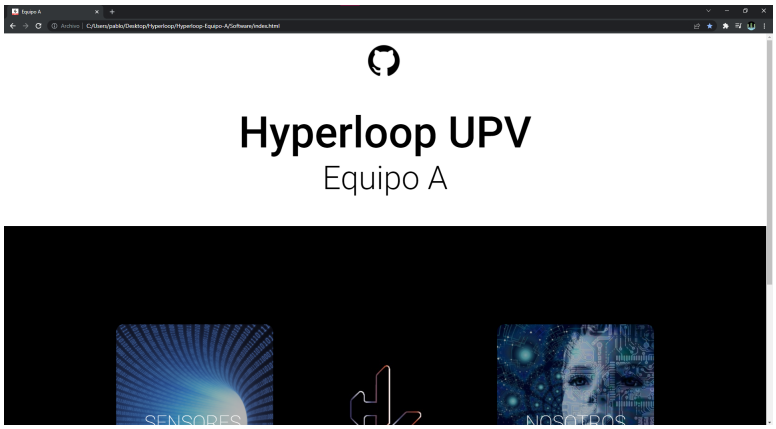


Ilustración 3.4: Index de la página actual.

Para asegurarnos que la página fuese fácil de usar y fácil de programar, primero nos reunimos para decidir a grandes rasgos que queríamos que se pudiese hacer en la página y como queríamos que fuese. Después hicimos diseños preliminares de la estructura de cada sub página, como por ejemplo el que se puede observar en la siguiente figura.

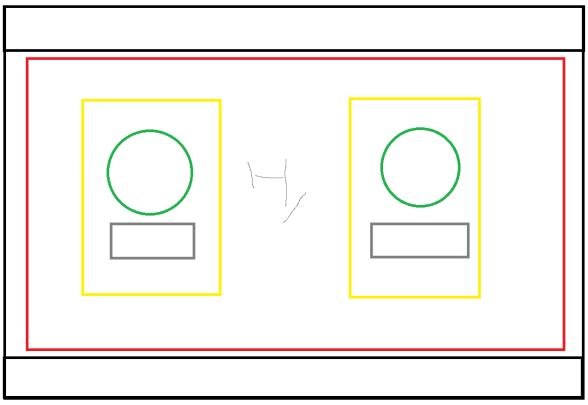


Ilustración 3.5: Diseño preliminar del apartado de información.

Una vez claro como iba a ser cada página nos dispusimos a programar-

la de la forma más estructurada posible. Una vez terminada se solicitó feedback al equipo para detectar fallos en el diseño y poder corregirlos. Nuestro objetivo con la página era que fuese minimalista y simple, asegurándonos en todo momento que fuese fácil navegar por ella. Tampoco quisimos dejar que quedase demasiado estática por eso añadimos alguna animación, ayudando a hacer más visual la página y haciendo que navegar por ella sea más intuitivo.

Finalmente, programamos un pequeño script en JavaScript que es capaz de recibir datos de un servidor cuando pulsamos el botón de *play* y parar de recibirlos cuando lo volvemos a pulsar. A modo de prueba ahora mismo solo se muestra un contador descendente, pero con un pequeño cambio en vez de mostrarse el contador se mostrarían los datos recibidos.

El resultado final de la página y el código de esta se puede consultar en él GitHub del equipo.

Capítulo 4

Firmware

El objetivo del Firmware elaborado era coordinarse con el Hardware para que todo el apartado de operaciones que tenía que realizar el microcontrolador funcionase. El Firmware se ocupa del funcionamiento de la conversión del ADC así como la transmisión SPI y realizar internamente los cálculos necesarios para averiguar la temperatura en la NTC. El firmware fue realizado en el IDE oficial de MicroChip, MPLAB, en concreto la versión 6.0.0, el compilador empleado fue el XC8. Para testear el correcto funcionamiento del firmware se recreó el esquemático en Proteus. Un programa que permite la simulación de microcontroladores cargándoles programas. En este capítulo se explicará el funcionamiento de diversas partes del Firmware.

ADC (Analog-to-Digital Converter)

El ADC del microcontrolador puede convertir señales analógicas en señales digitales de como máximo 10 bits. El módulo ADC dispone de 4 registros: ADRESL, ADRESH, ADCON0 y ADCON1. Los dos primeros están al cargo del almacenamiento del resultado de la conversión digital. ADCON0 controla lo relativo al funcionamiento del ADC mientras que ADCON1 controla el posicionamiento del resultado en los registros de almacenamiento así como la configuración de los puertos de entrada analógicos. Para el programa se han creado dos funciones para utilizar el ADC, ADC_init() y ADC_read().

4.1. ADC_init()

```
void ADC_init(void) // Funcion que se encarga de la
↪  configuracion inicial del ADC
{
    ADCON1bits.ADFM = 1;         // Posiciona los MSB a
    ↪  la derecha
    ADCON1bits.PCFG = 0b0100;    // Configuracion que
    ↪  establece
                                 // → el pin AN3 como
                                 ↪  analogico

    ADCON1bits.ADCS2 = 0;        // Seleciona la
    ↪  operacion 32 TOSC
    ADCON0bits.ADCS1 = 1;        //  → para superal el
    ↪  TAD
    ADCON0bits.ADCS0 = 0;        //  → minimo de 1.6uS

    ADCON0bits.ADON = 1;         //Enciende el ADC
}
```

La función se encarga de la configuración inicial del ADC, ha de ser ejecutada antes de comenzar a realizar conversiones en el ADC.
La primera sentencia estable el bit ADFM del ADCON1 a 1. En esta posicion los primeros 8 bits del resultado de la conversión se almacenan en ADRESL y los ultimos 2 se almacenan en los 2 LSB (Least Significant Bits) de ADRESH.

Se establece el bit PCFG del ADCON1 a 0100, este bit sirve para elegir la distribución de puertos analógicos y digitales que se va a usar. En este caso se escoge una distribución que establezca el puerto AN3 como analógico y deje el mayor número de puertos restantes como digitales por si se van a usar más tarde, esto coincide bastante bien con el caso 0100.

La conversión está formada por dos periodos: Adquisición (TACQ) y Conversión (TAD) que deben coordinarse para no introducir ningún error en la conversión. En el interior del microcontrolador, previo a la entrada al ADC se encuentra un condensador el cual se carga con el voltaje que se encuentra en la entrada analógica, este es el tiempo de

adquisición. Mientras que el tiempo que tarda el ADC en convertir el voltaje del condensador en una señal digital sumado al tiempo de descarga del condensador es el tiempo de conversión.
Para seleccionar el tiempo de conversión, el fabricante nos indica en el datasheet que el TAD mínimo para una conversión correcta es $1{,}6\,\mu s$.
Para establecer el tiempo de conversión hay que multiplicar el periodo de oscilación del reloj que estamos empleando por una constante que se establece con los bits del ADCS y este debe ser mayor al TAD mínimo de $1{,}6\,\mu s$.:

$$TAD_{min} \leq \frac{1}{f_{osc}} * ADCS$$

Se observa que la constante por la que hay que multiplicar el periodo de oscilación es por lo menos 32. Según se muestra en el datasheet para escoger el 32 hay que colocar el bit ADC2 del ADCON1 a 0, el ADCS1 del ADCON0 a 1 y el ADCS0 del ADCON0 a 0.

Finalmente se coloca el bit ADCON del ADCON1 a 1 el cual comienza la alimentación el conversor ADC.

4.2. ADC_read()

```
uint16_t ADC_read(uint8_t canal) // Funcion que se
 ↪  encarga de leer el ADC
{
    ADCON0bits.ADCS = canal;    //Selecciona el canal
     ↪  de entrada
                                // →analogica
    __delay_us(25);             //Tiempo de
     ↪  adquisicion TACQ,
                                // →el minimo es
                                 ↪  19,72uS
    ADCON0bits.GO = 1;          //Inicia el ADC
    while(ADCON0bits.GO_DONE);  //Espera que termine
                                // →la conversion.
    return ((uint16_t)((ADRESH<<8)+ADRESL));//Devuelve
     ↪  el valor del ADC
}
```

La función ADC_read() es llamada cada vez que se quiera hacer una lectura del puerto analógico, recibe como parámetro el puerto que se

quiere leer.
Comienza seleccionando el canal que se quiere leer en base al parámetro recibido. Para ello se emplean los bits Channel Select del ADCON0.
A continuación se esperan $25\,\mu s$ hasta que se cargue el condensador previo al ADC, ya que segun consta en el datasheet el TACQ mínimo son $19{,}72\,\mu s$.
Se coloca el bit GO del ADCON0 a 1, esto comienza el proceso de conversión en el ADC, este bit siempre hay que activarlo por firmware cada vez que se quiera comenzar la conversión de ahí que se incluya en la función. Sin embargo, se coloca a 0 automáticamente cuando la conversión finaliza. Aprovechándose de ello se introduce un bucle while vació que tiene como condición el estado del bit GO, de manera que frena la ejecución del programa hasta que finalice la conversión.
Finalmente teniendo en cuenta como se almacenaba el resultado en los registros ADRESL y ADRESH. Se desplazan los dos LSB de ADRESH y se añaden a los bits de ADRESL y se realiza una refundición a entero sin signo de 16 bits y se devuelve como resultado de la función, el cual será un resultado de 0 a 1023.

SPI(Serial Peripheral Interface)

La transmision serial es realizada por el módulo MSSP (Master Synchronous Serial Port), puede realizar tanto comunicación SPI como I^2C.
Para nuestro proyecto empleamos la comunicación SPI que requiere de 4 conexiones entre el emisor y el receptor y se caracteriza por poder conectarse múltiples receptores al emisor principal. El emisor principal se suele denominar Master (Maestro) y los receptores reciben el nombre de Slave (Esclavo). Las 4 conexiones principales son el cable MOSI(Master-Output Slave-Input), MISO(Master-Input Slave-Output), SCK (Synchronous Clock) y el SS(Slave Select). A traves del MOSI el Master manda datos al esclavo, el MISO se emplea para recibir datos del esclavo. El Slave Select sirve para establecer con que esclavo se va a realizar la comunicación y el SCK es la señal de reloj que establece los intervalos de lectura de los datos tanto para el maestro como para el esclavo.
En el caso del PIC16F873A la comunicación SPI se ve limitada a 8 bits lo cual introdujo problemas a la hora de mandar los datos leídos a través del ADC, ya que estos eran de 10 bits. Para introducir la menor imprecisión posible en la lectura de los datos, se ha decidido mandar los datos bajo un convenio que hemos elegido y que más adelante se explicará.
El módulo MSSP dispone de 5 registros: SSPCON1, SSPCON2, SSPSTAT, SSPBUF y SSPSR. Los SSPCON sirven para controlar y configurar

el módulo MSSP, el SSPCON2 está reservado para la comunicación I^2C por lo que solo emplearemos el SSPCON1. El registro SSPSTAT sirve para revisar el estatus del módulo MSSP por lo que es de solo lectura en su mayoría. El registro SSPBUF hace de buffer para el dato que se va a enviar o el que se va a leer. Dado que ambos datos de lectura y de escritura se almacenan en el SSPBUF existe otro registro, el SSPSR, que no es directamente accesible, tiene la función de mover datos dentro o fuera del SSPBUF en base a si se está leyendo o si se está escribiendo. Cuando se recibe un byte, lo almacena primero el SSPSR y luego lo mueve al SSPBUF mientras que cuando se transfiere un byte no pasa por el SSPSR y se almaecna directamente en el SSPBUF.

4.3. SPI_init()

```
void SPI_init(void) // Funcion que se encarga de la
↪  configuracion inicial del SPI
{
    PORTAbits.RA5 = 1; // Desactiva el esclavo
    TRISC = 0b01000000; // Configura todos los puertos
    ↪  C a entrada y el RC4(SCK) a salida
    SSPCONbits.SSPEN = 0; // DESACTIVA MSSP
    SSPSTAT = 0b01000000; // bit 7 muestreo en la
    ↪  mitad del dato bit 6 activo por flanco de
    ↪  bajada
    SSPCON = 0b00010010; // bit 0-3:Fosc/64 = 312kHz
    ↪  bit4:idle status high
    SSPCONbits.SSPEN = 1; // ACTIVA MSSP
}
```

La función se encarga de la configuración inicial de la comunicación SPI, ha de ser ejecutada antes de comenzar a recibir o transmitir datos a traves de SPI.
La primera sentencia pone un 1 en el pin RA5 que es el correspondiente al puerto SS que hemos elegido. El Slave Select es activo a nivel bajo entonces ponerlo a 1 estaría desactivando la comunicación con el esclavo para poder modificar los puertos SSPCON y SSPSTAT sin causar ningún problema.
A continuación se configuran los pines del puerto C como entradas y salidas. Ya que el MISO, MOSI y SCK se encuentran en este puerto. Se colocan todos como entrada digital, es decir 0, menos el RC4 que se coloca como salida digital, es decir 1, ya que es el correspondiente al pin de SCK. EL MOSI lo configura el microcontrolador automáticamente a la hora de entablar comunicación justo después de la activación del bit SSSPEN del SSPCON1, por lo tanto, no hace falta colocarlo como salida.

Se desactiva el bit SSPEN del registro SSPCON1. Este bit se encarga de la activación del módulo MSSP y de altenar los pines entre I/O digital o la configuración para la comunicación SPI, como es el caso del MISO y del MOSI mencionado anteriormente.
Se realizan una serie configuraciones en el registro SSPSTAT. Se coloca a 0 el bit de SMP haciendo que las lecturas de datos ocurran en la mitad de un ciclo de reloj evitando que se lean al final de un ciclo, ya que hay más posibilidades de que se lea el dato incorrecto. Se coloca a 1 el bit CKE para que la transmisión de datos ocurra en el flanco de bajada en el reloj.
Se configuran los bits del SSPCON1. Los bits del 0 al 3 corresponden al SSPM que sirve para seleccionar la frecuencia del reloj SCK de la transmision en base a la frecuencia del reloj externo. En la configuracion elegida la frecuencia del reloj es igual a:

$$f_{SCK} = \frac{f_{OSC}}{64} = 312\,\text{kHz}$$

El bit 4 se pone a 1, ya que corresponde al CKP ocupado de fijar el estado por defecto del reloj a nivel alto.

Finalmente se vuelve a activar el bit SSPEN del SSPCON0 para activar el MSSP y configurar los puertos de la comunicación SPI.

4.4. SPI_read()

```
uint8_t SPI_read(uint8_t dato) // Funcion que se
↪  encarga de leer/escribir el SPI
{
    uint8_t y;
    SSPBUF = dato; // guarda el dato en el
                   //registro buffer
    while(SSPSTATbits.BF == 0); //Mientras el buffer
                                //no este lleno nada
    y = SSPBUF; //guarda en y el buffer
    return y;
}
```

La función SPI_read() es llamada cada vez que se quiera transmitir o recibir un dato a través de comunicación SPI, recibe como parámetro un entero de 8 bits sin signo y devuelve el mismo tipo de dato, ya que como se mencionaba antes es el máximo que puede manejar el microcontrolador. El dato en el parámetro es el dato a transmitir, si este parámetro estuviera vació no se transmitiría nada y se realizaría solo lectura porque siempre devuelve el dato recibido el cual se almacena internamente

en la variable y.
Para comenzar se almacena en el SSPBUF el dato a mandar el cual comenzará a ser transmitido.
A continuación se emplea el bit BF del SSPSTAT. Este bit se coloca automáticamente a 1 cuando el SSPBUF está lleno, indicando que la recepción de un dato se ha completado, mientras no este lleno se encuentra a 0, de manera que se ejecuta un bucle vació que espera a que el SSPBUF este lleno para almacenarlo en la variable y, así devolviendo el dato recibido.

4.5. main()

```
void main(void) {
    ADC_init();
    SPI_init();
    while(1)
    {
        PORTAbits.RA5 = 0; // Activa el esclavo
        lectura = ADC_read(0b011); //Lee el canal 011
        ↪  que corresponde al AN3
        NTCvoltage = ((5.0/1023) * lectura);
        R=1/((5/(5-NTCvoltage))-1) ;
        ↪  // CALCULO TEMPERATURA
        temp=2*(1/((log(R)/3380)+(1/298.15))-273.15);
        spireturn = SPI_read(temp);
        ↪  // Se transmite SPI 2*temp
        __delay_ms(100);
    }
}
```

Se han suprimido de la ilustración las declaraciones de las variables. El método main se ejecuta cuando el microcontrolador recibe alimentación. Comienza inicializando la lectura del ADC como la comunicación SPI con las funciones previamente explicadas. Entra en el bucle while(1) el cual se va a estar ejecutando mientras el microcontrolador esté encendido, a grandes rasgos, se realiza la lectura del ADC, se despeja la temperatura en la NTC usando una serie de ecuaciones y se transmite a través de SPI.
Primero se coloca el pin RA5 a 0, ya que este corresponde al SS y es activo por nivel bajo, indicando que vamos a comenzar la comunicación con el esclavo.
Se le el valor del voltaje en el pin del ADC el cual se almacenara como

un valor entero sin signo de 0 a 1023. En este caso se está leyendo el pin AN3 el cual corresponde al 011, pero si se desea leer otro pin analógico solo habría que consultar los Analog Channel Select Bits en el datasheet.

Dado que únicamente podemos mandar datos de 8 bits al esclavo se ha decidido calcular internamente en el microcontrolador el valor de la temperatura, este multiplicarlo por 2 y transmitirlo y que luego el esclavo vuelva a dividirlo entre 2. De esta manera obtenemos un rango de representación de $0\,^{\circ}\mathrm{C}$ a $127{,}5\,^{\circ}\mathrm{C}$ con un error máximo de $\pm 0{,}5\,^{\circ}\mathrm{C}$.

Se prosigue con el cálculo de la temperatura en la NTC, las fórmulas utilizadas están simplificadas aprovechándose valores de la NTC, de tal manera que el microprocesador reciba menor carga. Las ecuaciones utilizadas para hallar estas fórmulas son las siguientes:

$$V_{ADC} = \frac{ADC\_read()}{1023} * 5$$

$$V_{NTC} = 5 - V_{ADC}$$

$$V_{NTC} = (1 - \frac{R_{10}}{R_{10} + R_{NTC}}) * 5^{**}$$

$$R_{NTC} = \frac{R_{10} V_{NTC}}{5 - V_{NTC}}$$

$$T_K = \left(\frac{ln(\frac{R_{NTC}}{R_0})}{\beta} + \frac{1}{T_0} \right)^{-1}$$

$$T_C = T_K - 273{,}15$$

Y por último, tras calcular la temperatura, se transmite el doble de la temperatura usando la función SPI_read().

**Proviene del divisor de tensión que conforma R_{10} con la NTC