ESCUELA SUPERIOR DE TECNOLOGÍA Y CIENCIAS EXPERIMENTALES

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Diseño y realización de un controlador de retrolavado para su uso en instalaciones de riego

Trabajo de final de Máster

Autor: Benjamín Carratalá Font

Director: Ignacio Peñarrocha Alós

Castellón, 11 de julio de 2018

ÍNDICE

1 MEMORIA

1.1 Objeto

El presente proyecto tiene como objetivo la automatización del sistema de lavado en los equipos de filtrado de agua utilizados para el riego de zonas agrícolas con el fin de mejorar la eficiencia y productividad del campo de cultivo donde se aplica.

El proyecto ha sido desarrollado en la empresa "Electronobo s.l.", especializada en la automatización de sistemas en entornos agrícolas, y tiene su origen en la necesidad del cliente de automatizar el equipo de filtrado de aguas en la instalación de riego.

Imagen 1 – Sede principal de Electronobo s.l.

1.2 Alcance

El sistema objeto del presente proyecto ha sido concebido para su uso en equipos de filtrado de agua con filtros de disco que permitan la limpieza por retrolavado.

El dispositivo está diseñado para equipos de filtrado compuestos por un máximo de 5 filtros.

El sistema está preparado para funcionar en emplazamientos sin acceso a la red eléctrica. Además, gracias a su resistencia e impermeabilidad, puede instalarse a la intemperie sin que esto altere su funcionamiento.

1.3 Antecedentes

El proyecto ha sido desarrollado en "Electronobo s.l.", una empresa especializada en la automatización de sistemas empleados en campos de cultivo y entornos rurales. Para la realización del presente proyecto, la experiencia de esta empresa en este tipo de

emplazamientos tan particulares ha hecho preferible utilizar los mismos métodos y materiales con los que ellos trabajan habitualmente.

El sistema realizado debe además ser compatible con los equipos de filtrado con filtros de disco y de limpieza por retrolavado para los cuales se diseña.

Dicho esto, debe saberse principalmente que el controlador de lavado de filtros creado, tiene su referencia más importante en otro controlador similar. Así pues, si bien no se ha buscado recrear el mismo dispositivo, el objetivo sí ha sido reproducir las funcionalidades que se han considerado útiles para nuestra aplicación.

1.4 Normas y referencias

1.4.1 Disposiciones legales y normas aplicadas

De la misma manera que se hace en "Electronobo s.l." con los demás dispositivos con propiedades similares, para la realización del sistema objeto del presente proyecto se han tenido en cuenta distintas normas referentes a los dispositivos electrónicos, así como a sus propiedades relacionadas con la compatibilidad electromagnética.

Las normas principales que se hayan involucradas en el proceso son las siguientes:

- EN 300 113-2 V1.2.1 y EN 301 489-05 V1.3.1 – Compatibilidad electromagnética y propiedades del espectro de frecuencias.
- UNE-EN 60950-1:2007 – Referente a la seguridad de los equipos electrónicos empleando tecnología de la información y/o tecnología de la comunicación.

1.4.2 Programas de cálculo

1.4.2.1 IAR Embedded Workbench IDE

En términos de tiempo de utilización, la principal herramienta software empleada para la realización del proyecto ha sido el IDE (*Integrated Development Environment*, o en español: Entorno de desarrollo integrado) que permite programar el microcontrolador para así definir su comportamiento.

El IDE empleado ha sido el IAR Embedded Workbench IDE, mediante el cual ha sido posible, tras definir el modelo de microcontrolador sobre el que actuaría, desarrollar el código en lenguaje C, que será automáticamente traducido a lenguaje máquina para programar el microcontrolador.

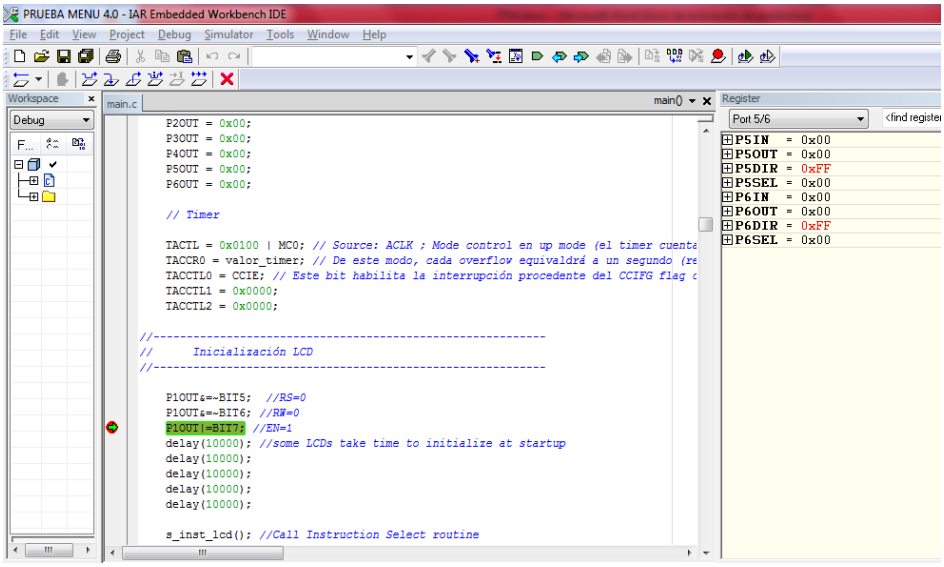


Imagen 2 – Captura de pantalla del IAR Embedded Workbench IDE en modo depurador

Cabe destacar la función de depurador que ofrece el IDE. Como vemos en la imagen 2, podemos hacer un seguimiento del código ejecutando el programa línea a línea y viendo cómo responde el microcontrolador al mismo tiempo. Como vemos también en la parte derecha de la imagen 2, podemos ver también en directo el contenido de los distintos registros de los que dispone nuestro microcontrolador.

1.4.2.2 AutoCAD

Entre las múltiples posibilidades de diseño que ofrece AutoCAD, aquella que más ha sido utilizada para el desarrollo del presente proyecto ha sido el diseño de circuitos eléctricos.

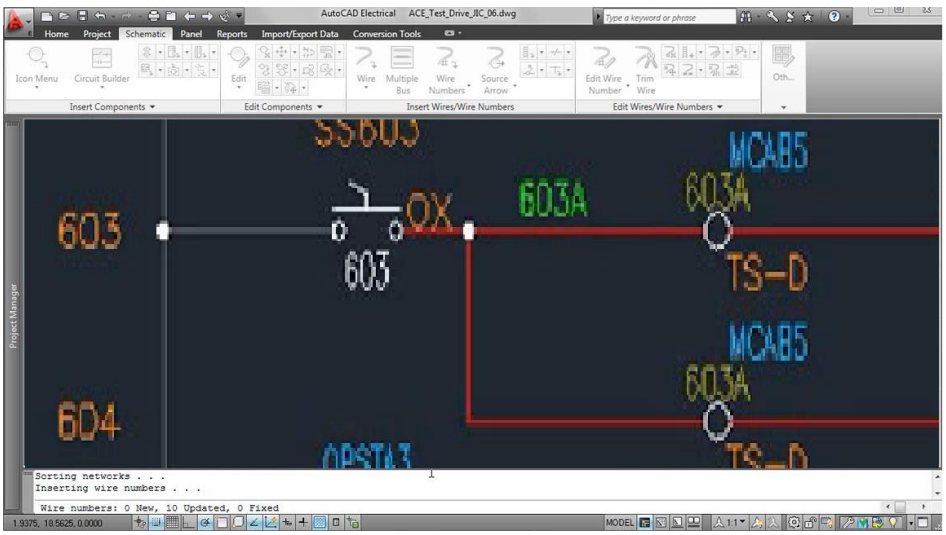


Imagen 3 – Captura de pantalla de AutoCAD Electrical

Así pues, haciendo uso de la gran variedad de herramientas de que dispone AutoCAD, se ha dibujado el circuito eléctrico de nuestro sistema de modo a tener plasmados los

detalles de las conexiones y así facilitar la instalación y su posterior mantenimiento si fuese necesario.

1.4.2.3 OrCAD

Una vez tengamos el circuito del dispositivo definido y el microcontrolador programado, estaremos en disposición de diseñar la PCB (*Printed Circuit Board,* o en español: Placa de circuito impreso). Es para este fin para el que ha sido utilizada la herramienta software OrCAD.

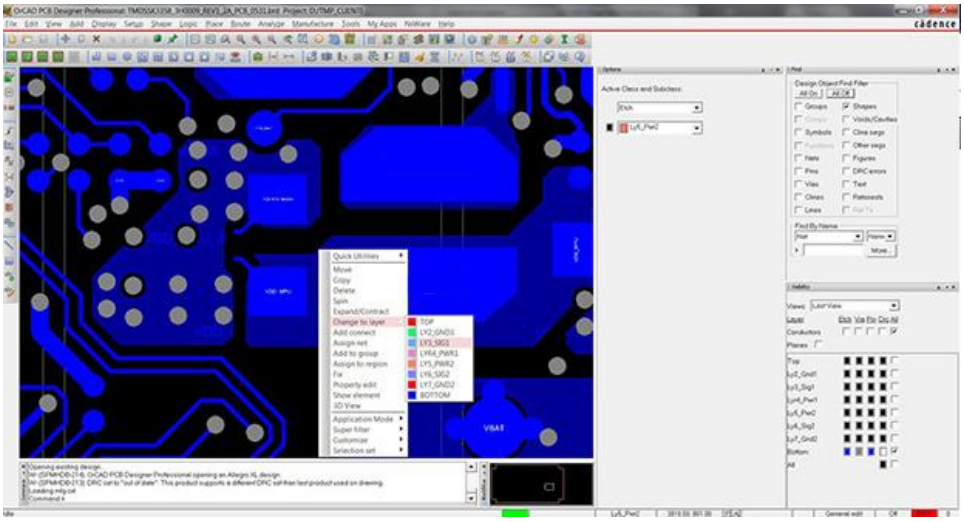


Imagen 4 – Captura de pantalla de OrCAD

Mediante este software, los distintos elementos que constituyen la placa son dispuestos de forma optimizada para ocupar el menor espacio posible, y una vez terminado el diseño, se puede guardar como un fichero informático que será enviado a las fábricas especializadas para la materialización de nuestra PCB tal y como la habíamos diseñado.

1.4.3 Bibliografía

[1] Texas Instruments - MSP430x1xx Family User's guide. Disponible en: <http://www.ti.com/lit/ug/slau049f/slau049f.pdf>

[2] Wikipedia - Hitachi HD44780 LCD controller. Disponible en: <https://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller#Instruction_set/ >

[3] Xanthium enterprises - ADC10 Tutorial for MSP430 Launchpad. Disponible en: <http://xanthium.in/msp430-launchpad-adc10-configuration-tutorial>

[4] Coder-Tronics - MSP430 Programming Tutorial Pt/1. Disponible en: <http://coder-tronics.com/msp430-programming-tutorial-pt1/>

[5] Embedded related - MSP430 Launchpad Tutorial - Part 2 - Interrupts and timers. Disponible en: <https://www.embeddedrelated.com/showarticle/182.php>

[6] Universitat Jaume I - SJA003 - Electrónica Industrial e Instrumentación

[7] Universitat Jaume I - EE1009 - Expresión Gráfica

1.5 Definiciones y abreviaturas

1.5.1 Definiciones

- Electroválvula: Válvula accionada por medio de una bobina solenoide permitiendo así controlar el paso de un fluido por un conducto o tubería mediante una señal eléctrica.
- Estructura condicional: Tipo de estructura de control que permite ejecutar, en función del valor de una variable, unas u otras acciones.
- Estructura de bucle: Tipo de estructura de control que permite ejecutar repetidamente un conjunto de acciones hasta cumplir con una condición definida.
- Datasheet: Documento en el que se encuentran las especificaciones necesarias para que el dispositivo objeto del documento sea empleado correctamente y de forma segura.
- Hardware: En oposición al software, el hardware se compone de las partes físicas tangibles de un sistema informático; sus componentes eléctricos, electrónicos, electromecánicos y mecánicos.
- High: En español: alto. Se utiliza para indicar el nivel de una señal digital.
- Interrupción: Evento que hace que el microcontrolador deje de ejecutar la tarea que está realizando para atender dicho acontecimiento y luego regrese y continúe la tarea previa.
- Low: En español: bajo. Se utiliza para indicar el nivel de una señal digital.
- Microcontrolador: Circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Generalmente son utilizados para controlar dispositivos con una función bien definida.
- Polling: Método de comunicación en el que el procesador se encarga de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tiene pendiente alguna comunicación para él.
- Registro: Espacios de memoria del microcontrolador, generalmente reducidos, y que sirven tanto para la configuración de su funcionamiento, como para actuar sobre las salidas o leer entradas o estados del sistema.
- Sensor de presión diferencial: Dispositivo capaz de emitir una señal eléctrica variable en función de la diferencia de presión entre dos puntos.
- Software: En oposición al hardware, el software consiste en el conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

- Timer: Módulo que permite el contaje del tiempo a partir de un oscilador en un sistema informático.

1.5.2 Abreviaturas

- C-LiFi: Controlador de Limpieza de Filtros.
- DC: Direct Current. En español: Corriente continua.
- IDE: Un entorno de desarrollo integrado. Del inglés: Integrated Development Environment.
- I/O: Input/Output.
- LPM: Low Power Mode. En español: Modo de baja potencia.

1.6 Requisitos de diseño

Para la realización del presente proyecto se han tenido en cuenta unos requisitos de diseño impuestos ya sea por el cliente, por las características del emplazamiento donde se instalará el dispositivo, etc...

Como se ha explicado con anterioridad, nuestro dispositivo está destinado a su uso en el sector agrícola. En este tipo de entornos, como puede ser un campo de cultivo se deben prever una serie de particularidades que condicionan el diseño de nuestro sistema y que "Electronobo s.l." por haberse especializado en dicho sector, conoce de primera mano.

Así pues, para encontrar una buena solución, acorde con los intereses del cliente y la empresa responsable, se considerarán también aspectos como el coste económico, la robustez del sistema frente a las condiciones a que será expuesto, su durabilidad, su facilidad y comodidad de uso, entre otros.

1.6.1 Requisitos derivados del emplazamiento

Habitualmente los campos de cultivo se encuentran a la intemperie y tampoco se tiene acceso a la red eléctrica, es por ello que nuestro dispositivo debe estar provisto de características que le permitan funcionar de forma autónoma. Para ello nuestro sistema deberá:

- **Disponer de una fuente de energía para alimentar el dispositivo**. De este modo el mismo sistema presentaría su propia fuente de energía y no sería necesario conectarlo a la red eléctrica, algo que supondría modificar la instalación eléctrica, causando que el coste del proyecto sea demasiado elevado.
- **Presentar un bajo consumo**. Debido al requisito del funcionamiento autónomo, el bajo consumo del sistema se convierte en un punto a tener siempre en cuenta. Cuanto menos consuma nuestro sistema, menos potente deberá ser

nuestra fuente de alimentación, algo que también repercute en los costes del proyecto.

- **Ser robusto frente a las condiciones meteorológicas.** Al estar expuesto a las condiciones climáticas, el dispositivo deberá presentar una buena impermeabilidad, además de no sufrir alteraciones en su funcionamiento debido a la temperatura. Para ello, se deberán utilizar componentes poco sensibles a estos cambios. Un ejemplo sería la utilización de un piezo eléctrico externo como sistema para contar el tiempo en lugar del oscilador interior del microcontrolador, que presenta un comportamiento demasiado sensible a la temperatura.

Imagen 5 – Imagen de diversas condiciones posibles a la intemperie

1.6.2 Requisitos del cliente

Por su parte, el cliente requiere de unas prestaciones que permitan que el dispositivo se adapte de la mejor forma posible a su actividad. Dichos requisitos deben pues tenerse en cuenta y convivir con los demás. En el presente caso se han especificado los siguientes requisitos:

- **Posibilidad de configurar prácticamente la totalidad del funcionamiento:** Se requiere tener acceso a la configuración de prácticamente todo tipo de parámetros. De este modo será posible ajustar mejor el funcionamiento a sus necesidades en función de sus intereses en cada momento.
- **Uso simple e intuitivo:** Puesto que el cliente no prevé manipular el dispositivo con frecuencia (dado que funciona automáticamente), se desea que su uso sea

simple e intuitivo. Lo que permitirá su configuración sin necesidad de leer el manual, además de ayudar a prevenir posibles malentendidos derivados de posibles ambigüedades.

- **Posibilidad de introducir el número de filtros:** Dado que el cliente dispone de equipos de filtrado con diferente número de filtros, es interesante que los dispositivos puedan ser configurados para adaptarse al número de filtros necesario. El requisito en nuestro caso ha sido que pueda adaptarse a equipos de filtrado de desde 1 filtro hasta 5 filtros.
- **Activación de la limpieza mediante 3 modos posibles:** A la hora de empezar a limpiar, la especificación consiste en qué dicha limpieza se pueda activar porque ha transcurrido un determinado tiempo (modo temporizado), porque se ha superado una determinada presión (modo por diferencia de presión) o bien porque se ha activado la limpieza manual mediante la interfaz del dispositivo (modo manual). Tanto la diferencia de presión como el tiempo a partir de los cuales se empieza a limpiar podrán ser configurados por el usuario.
- **Detección de funcionamientos no deseados:** Se pide que en caso de producirse algún fallo en el sistema que el dispositivo pueda detectar, que dicho fallo se identifique y se registre como tal en el dispositivo.
- **Registro de limpiezas:** También se desea que cada vez que se efectúe una limpieza, ésta quede registrada en el historial del dispositivo especificando mediante qué modo se ha activado dicha limpieza: manual, por diferencia de presión o por tiempo.

1.6.3 Otros requisitos importantes

A parte de los requisitos ya mencionados que definen la mayor parte del proyecto, cabe nombrar algunos más, que también han debido ser considerados para el buen resultado de nuestro proyecto.

- **Dispositivo lo más compacto posible:** En la medida de lo posible, utilizar el menor número de pines del microcontrolador para así facilitar la adición de periféricos y nuevas funciones al dispositivo en el futuro.
- **Software flexible:** Puesto que se trata del primer dispositivo de una serie de ventas potenciales de dispositivos similares al mismo cliente, se ha recomendado también que el software, que constituye el coste más elevado en la elaboración del dispositivo, pueda ser mejorado o reutilizado de forma más simple y eficiente para efectuar ciertas modificaciones o ser adaptado para unas funcionalidades distintas en otro dispositivo.

1.7 Análisis de soluciones

A la hora de buscar las soluciones para nuestro proyecto, se han tenido en cuenta principalmente los requisitos anteriormente citados, así como otros condicionantes derivados de la empresa, como puede ser, la disponibilidad de ciertos materiales en stock, el *know-how* que posee sobre ciertos dispositivos etc...

Si bien las distintas partes del dispositivo objeto del proyecto conviven trabajando juntas, podemos hacer una distinción bastante ilustrativa entre aspectos del software y aspectos del hardware.

1.7.1 Software

Ha sido en lo referente al software en lo que más se ha trabajado en este proyecto. A continuación se describirán las funciones que debía realizar así como las soluciones por las que se ha optado.

1.7.1.1 Lectura de las señales por el microcontrolador (interrupciones o "polling")

Para la lectura de las distintas señales que debe procesar el microcontrolador, como son la señal procedente del sensor de presión, la del timer y la enviada por los pulsadores, sehan comparado las dos métodologías existentes:

- **Polling**: Sondeo cíclico del estado de las señales. El microcontrolador está siempre en modo activo comprobando si hay algún cambio al que deba responder.
- **Interrupciones**: Activación por eventos. Sólo cuando ocurre un evento el microcontrolador ejecuta las acciones asociadas. Mientras no ocurre ningún evento, se encuentra en reposo.

Teniendo en cuenta los aspectos a valorar para nuestra aplicación, se pueden comparar los dos métodos según:

- **Simplicidad del diseño:** Al contrario que con las interrupciones, el código ejecutado en el método "polling" es siempre el mismo y en el mismo orden. Esto simplifica el diseño del programa ya que no se debe tener en cuenta la posibilidad de ejecución de acciones en órdenes variables cada vez, como ocurre con las interrupciones, dada su naturaleza impredecible.
- **Funcionalidad:** Tanto con un método como con el otro, el funcionamiento sería correcto. No obstante, si se desea una mayor precisión en la medición del tiempo, es preferible la utilización de interrupciones para responder sin retrasos.
- **Consumo energético:** En cuanto al consumo, el método de las interrupciones es generalmente más eficiente, y la diferencia es mayor cuanto menos

frecuentes sean los eventos a los que se debe responder. Así pues la valoración de este aspecto dependerá de la previsión de funcionamiento.

Puesto que uno de los principales requisitos del proyecto es su buena autonomía, se debe priorizar la optimización del consumo.

Además, puesto que el sistema está diseñado para permanecer la mayor parte del tiempo en reposo, la diferencia de consumo se ve aumentada en gran medida. Es por esto que se ha decidido utilizar el método de las interrupciones para nuestro sistema.

1.7.1.2 Método de navegación del menú de la interfaz del dispositivo

Para poder modificar parámetros del dispositivo con el fin de configurarlo y programarlo, así como para poder revisar los registros u otros parámetros mediante la interfaz del dispositivo es necesario poder navegar por las distintas pantallas que indican las opciones disponibles.

Para ello, y partiendo del conocimiento de que la interfaz se compondrá de una pantalla LCD de 2x16 caracteres y de unos pulsadores, se han comparado distintas opciones para posibilitar la navegación y la interacción por las distintas pantallas.

Para el diseño del método a implementar se ha partido de la hipótesis de que el acceso a cualquier pantalla viene definido por dos variables:

- La pantalla anterior.
- El botón pulsado.

No obstante, el acceso a una pantalla no es único, y se puede llegar desde múltiples pantallas pulsando distintos botones.

1.7.1.2.1 Método función - pantalla

Investigando acerca de cuál sería el mejor método y consultando diversas fuentes bibliográficas donde se resolvían problemáticas similares, el principal resultado encontrado fue la navegación mediante funciones.

El método consiste en que cada pantalla tenga asignada una función, que será llamada cuando se deba acceder a la misma, y que tendrá como deber esperar a que el usuario introduzca un dato para actuar en consecuencia.

Así pues, esta función constaría principalmente de:

- Una estructura de bucle en la que se espera la introducción de un dato por parte del usuario.
- Una estructura condicional donde el dato recibido se compara con las distintas opciones disponibles para ejecutar la acción correspondiente según su valor.

Tal y como se muestra en el código de ejemplo siguiente:

```
int choice;
do {
   printf("Please make a choice: \n"
          "1) Option One \n"
          "2) Option Two \n"
          "3) Option Three \n"
          "4) Option Four \n"
          "5) Exit \n");
   scanf("%d", &choice);
   if (choice == 1) {funcion1();}
   else if (choice == 2) {funcion2();}
   else if (choice == 3) {funcion3();}
   else if (choice == 4) {funcion4();}
   else if (choice == 5) {funcion5();}
   else {
        printf("Invalid Choice");
        }
} while (choice != 5);
```

Imagen 6 – Código de ejemplo método de navegación "función – pantalla"

En nuestro caso, puesto que se trabaja por interrupciones, este método se adaptaría sin necesitar el bucle mencionado. Este se sustituiría por una variable que nos indicara la pantalla en la que nos ubicamos y sería utilizada para dirigirnos directamente a ella cuando se pulse un botón. De este modo, se requeriría una estructura condicional para cada pantalla donde se identificaría el botón pulsado y posteriormente realizar la acción correspondiente.

Así pues, considerando que, como se ha indicado antes, a cada pantalla se podría acceder desde otras X, se deduce que para cada pantalla se tendrían X líneas condicionales para llamar a su función (una en cada pantalla desde la que se puede acceder).

1.7.1.2.2 Método matriz de cadenas de caracteres

Estudiando las necesidades de nuestro sistema, se estudiaron otras alternativas distintas a las encontradas en las fuentes bibliográficas. De ellas, la que se postuló como más atractiva para nuestro sistema fue la de utilizar una matriz donde se almacenaran las cadenas de caracteres correspondientes a los textos de las pantallas de que debía estar compuesto el menú. Dicha matriz se recorrerá para encontrar el índice de la pantalla a la que queramos acceder.

A continuación se muestra un ejemplo de matriz con sus elementos (índices y cadenas de caracteres):

```
struct menu menu_principal [] = {
    {100000, "Pantalla 1 nivel menú 1"},
        {110000, "Pantalla 1 nivel menú 2"},
            {111000, "Pantalla 1 nivel menú 3"},
                {111100, "Pantalla 1 nivel menú 4"},
        {120000, "Pantalla 2 nivel menú 2"},
            {121000, "Pantalla 1 nivel menú 3"},
                {121100, "Pantalla 1 nivel menú 4"},
```

Imagen 7 – Ejemplo matriz de menú con el método de navegación mediante matriz

El principio de funcionamiento de este método es el siguiente:

- A cada pantalla le corresponde un índice propio y exclusivo.
- Cada botón tiene asignada una operación matemática que altera el valor del índice de pantalla de forma que se obtiene el correspondiente a la pantalla que se desea acceder.

1.7.1.2.3 Comparación de los métodos de navegación

Comparando los dos métodos presentados se pueden destacar los siguientes aspectos:

- **Simplicidad:** Si se trata de un menú relativamente pequeño, la opción función-pantalla presenta un funcionamiento más fácil de interpretar y por tanto de manipular y depurar. También es más simple, por lo que requiere un menor tiempo de desarrollo.
  Por su parte, en el método de la matriz se requiere desarrollar la programación de la acción asociada a cada botón antes de poder navegar por el menú, por pequeño que sea éste. Lo cual supone un trabajo excesivo con respecto al método anterior.
- **Líneas de código:** Como se ha visto, en el método función-pantalla, para añadir una pantalla se debe introducir la función correspondiente en las pantallas desde las que se debe tener acceso, y evidentemente escribir las acciones a realizar en la pantalla añadida.
  Por su parte, si se utiliza el método de la matriz, para cada pantalla que se añada únicamente se deberá introducir su índice y el texto en la matriz del menú.
  Así pues, mediante el método de la matriz se ahorrarán muchas líneas de código para realizar un menú amplio como el que se necesita en nuestro caso.
- **Facilidad para realizar modificaciones:** Utilizando el método función-pantalla, para añadir una pantalla, o simplemente para modificar el orden o la disposición de las pantallas en el menú se deberá actualizar el código correspondiente a cada una de las pantallas desde las que se podrá acceder a la pantalla que incorporemos o que cambiemos de lugar. Esto es sin duda farragoso, a la vez que una potencial fuente de errores si no se tiene un extremo cuidado a la hora de hacer las modificaciones.

Este problema no existe si utilizamos el método de navegación mediante matriz, ya que sólo con añadir el índice correspondiente a la ubicación donde queramos que esté la pantalla añadida o que hayamos cambiado de sitio, ésta será accesible utilizando los botones de navegación.

Debido al requisito de "Software flexible" presentado en el capítulo anterior, ha cobrado importancia la posibilidad y flexibilidad del método para realizar modificaciones.

Dado que se requiere para nuestra aplicación un menú con una gran cantidad de opciones y pantallas, así como que pueda ser modificado de forma simple, se ha optado para nuestro sistema por el método de navegación mediante matriz.

1.7.1.3 Respuesta ante limpiezas en bucle

Se sabe, por la experiencia con el tipo de equipos de filtrado para el que se crea nuestro dispositivo, que en ocasiones los filtros reciben tanta suciedad que ésta no puede ser eliminada con una secuencia de limpieza.

Cuando esto pasa, si nos encontramos en el modo de activación de limpieza por diferencia de presión, automáticamente al finalizar la limpieza, se vuelve a iniciar otra. Esto puede en ocasiones provocar limpiezas en bucle durante un considerable espacio de tiempo.

Puesto que dichas limpiezas en bucle suponen ya no sólo la activación de limpiezas por parte de nuestro dispositivo, sino también el desecho de una considerable cantidad de agua, se estudió la posibilidad de reducir las consecuencias de este problema.

1.7.1.3.1 Solución estudiada

La solución propuesta consiste en la detección de cuándo se producen limpiezas consecutivas, y cuando se dé el caso, dejar de limpiar según la diferencia de presión y empezar a funcionar de forma temporizada, es decir, efectuando una limpieza cada cierto tiempo especificado por el usuario. Una vez la diferencia de presión disminuyera del umbral definido para la limpieza, el dispositivo volvería a su funcionamiento normal según la diferencia de presión. Este funcionamiento tiene la capacidad de limpiar los filtros con un menor número de limpiezas debido al espaciamiento temporal entre las limpiezas.

La solución propuesta presenta algunas ventajas pero también ciertas desventajas:

- **Ahorro de recursos:** Como se ha descrito, al disminuir el número de limpiezas necesarias se tendría tanto un menor consumo en el controlador como un menor desecho de agua.

- **Rapidez del sistema:** Aunque con más número de limpiezas, mediante la ejecución de limpiezas en bucle el equipo de filtrado queda limpio en menos tiempo.

Aunque ambos modos son eficaces para limpiar los filtros, la solución propuesta es más eficiente en términos de consumo de energía y agua. Dado que esta ventaja es considerada más importante que la diferencia de tiempo que requieren para la limpieza del equipo, se ha decidido implementar dicha solución para nuestro sistema.

1.7.2 Hardware

1.7.2.1 Alimentación del sistema

Para alimentar el sistema, a partir del requisito de deber ser un sistema independiente de la red eléctrica, se propusieron dos soluciones.

La primera consistía en obtener energía de la luz solar mediante el uso de una placa fotovoltaica. Dicha placa iría acompañada de una batería que permitiera administrar la energía y de un regulador de carga que controlara la carga de la misma. De este modo, la energía obtenida de la luz solar podría ser utilizada por nuestro sistema.

Por otro lado se propuso la utilización de pilas para proporcionar la energía al sistema, eligiendo tanto la tensión como la carga disponible en el interior de estas pilas, se podrían utilizar para que el dispositivo funcionase de forma autónoma.

Si comparamos ambos sistemas, vemos que:

- En el aspecto económico, la inversión inicial a realizar es mucho menor en el caso de las pilas.
- La autonomía de las pilas tiene un límite que, una vez superado, obligaría a cambiarlas para que el sistema siga funcionando. Por su lado las baterías se recargan automáticamente permitiendo usarlas durante más tiempo.
- La meteorología así como otros factores que puedan influir en la recepción de luz por parte del panel alterará la energía obtenida, mientras que en el caso de las pilas no existe este problema.

Teniendo en cuenta estos factores, y añadiendo que los paneles fotovoltaicos requieren de un mantenimiento para su buen funcionamiento, y que las pilas se estima a partir de datos históricos que no deberán cambiarse hasta los dos años, se ha decidido finalmente utilizar pilas para el suministro energético al sistema.

Además, si en algún caso el equipo de filtrado se encontrara en un lugar techado, el sistema utilizando pilas nos sería útil sin necesidad de realizar modificaciones.

1.7.2.2 Medición de la presión

Para medir la presión se han estudiado dos opciones, la de utilizar a cada lado del equipo de filtrado un sensor de presión relativa (que mida la presión dentro de la instalación con respecto a la atmosférica) o la de utilizar un único sensor diferencial que mida la diferencia de presiones entre la entrada y la salida del equipo de filtrado.

Este sensor es el encargado no sólo de decirnos cuándo los filtros están demasiado sucios, sino también de informarnos de si existe alguna anomalía en el sistema, como pueden ser fugas aguas arriba o aguas abajo del equipo, mediante la medición de un valor que se encuentre fuera del rango habitual de trabajo.

Dado que las averías que se desea poder detectar producen presiones que pueden ser identificadas tanto mediante el sensor de presión relativa como mediante el diferencial, se ha decidido emplear el sensor diferencial por simplicidad.

En los sensores analógicos se puede hacer una distinción principal dependiendo de si a señal se envía en tensión o en corriente.

Para nuestro caso, dado que se desea optimizar el consumo, se ha decidido emplear un sensor de salida en tensión. Ya que en caso de ser en corriente, puesto que la interrupción de la lectura analógica debe estar siempre habilitada para poder detectar averías, se tendría continuamente un paso de corriente de mínimo 4 mA y hasta 20 mA dependiendo de la presión medida. Lo cual supondría un mayor consumo energético.

1.7.2.3 Actuadores

También se han estudiado distintas opciones para la activación de las electroválvulas que causarán la inversión del sentido del flujo del agua en el equipo de filtrado para desprender las partículas e impurezas.

Tras haber observado las opciones disponibles en el mercado, se ha optado por utilizar válvula de tipo “latch”. Estas válvulas tienen la característica principal de poderse abrir y cerrar mediante el envío de un pulso de tensión, lo que presenta la ventaja de no deber estar alimentando la electroválvula ininterrumpidamente para mantenerla abierta. Así pues, este funcionamiento es más eficiente en términos de energía, permitiendo disminuir el consumo del sistema.

Estas válvulas se encuentran en dos tipos de formatos para su apertura/cierre. Estos formatos son las válvulas de 2 hilos y las de 3. En el primer caso se abre enviando un pulso de tensión + Ve , y se debe invertir la tensión (-Ve) para efectuar el cierre. Las de 3 hilos, en cambio, funcionan sin modificar la tensión, y es enviando el pulso entre el hilo denominado “común” y el de apertura o el de cierre para abrir o cerrar la válvula respectivamente.

Tras estudiar las dos posibilidades vemos que la electroválvula de 2 hilos requiere la utilización de un circuito que permita invertir la polaridad de la tensión del pulso (por ejemplo mediante un circuito de puente en H), lo cual se traduce en un mayor número de componentes que el caso de la electroválvula de 2 hilos. Es por ello que finalmente se ha optado por emplear el formato de 3 hilos, tal y como se describe en el siguiente capítulo de este documento.

1.8 Resultados finales

Con las soluciones para las problemáticas anteriores definidas, es momento de abordar el conjunto de características finales del sistema.

Para describir de una forma más clara las características y el funcionamiento del sistema, se presentarán en primer lugar los elementos del hardware de forma a adquirir una visión del formato del sistema, para así después poder describir los del software de una forma más justificada.

1.8.1 Hardware

En este apartado se presentará el aspecto hardware del sistema. Como se puede observar en la imagen 8, el sistema está compuesto por distintos elementos que interactúan entre sí y que son coordinados por el microcontrolador y su software.

Entre estos elementos distinguimos una interfaz compuesta por unos pulsadores que envían su señal al microcontrolador, y una pantalla mediante la que el microcontrolador mostrará mensajes al usuario.

Los otros dos elementos, el sensor de presión diferencial y las electroválvulas constituyen los elementos de conexión entre el microcontrolador y el equipo de filtrado, y permitirán conocer mediante el sensor de presión la suciedad presente así como actuar sobre él utilizando las electroválvulas para efectuar una limpieza.

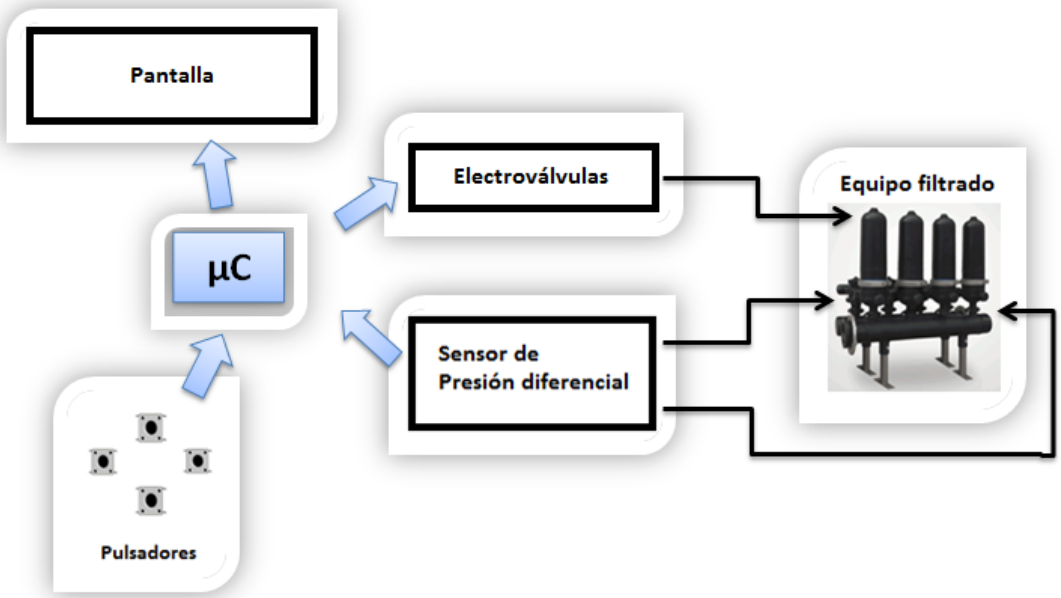


Imagen 8 – Diagrama de bloques de los elementos hardware del sistema

A continuación se describirán con detalle tanto los distintos elementos hardware del sistema como los circuitos diseñados necesarios para la interacción entre dichos elementos.

No obstante, es importante señalar que para el desarrollo del software de nuestro dispositivo se ha diseñado un sistema que permite imitar el sistema final desde el punto de vista del microcontrolador, y que ha servido para depurar el programa y comprobar su buen funcionamiento antes de instalarlo de forma definitiva en el dispositivo. Por lo que este sistema para el desarrollo del software también será presentado más tarde en este apartado.

1.8.1.1 Sistema de aplicación real

El sistema de aplicación real realizado se encuentra en el documento de planos bajo el nombre: *Conexionado del circuito de aplicación*.

En los siguientes subapartados se detallarán los componentes empleados para posteriormente describir el funcionamiento del circuito completo.

1.8.1.1.1 Componentes utilizados

1.8.1.1.1.1 Microcontrolador

El modelo de microcontrolador utilizado es el MSP430F149 de Texas Instruments. Cuya principal característica es su bajo consumo.

Además, puede ser programado utilizando múltiples IDE en lenguaje C, lo que permite elaborar programas más extensos de forma más rápida y clara.

El detalle de los pines de que dispone dicho microcontrolador se encuentra en el documento de planos.

1.8.1.1.1.2 Pantalla LCD

La pantalla elegida es de tecnología LCD, que presenta un bajo consumo así como un precio también asequible.

Esta provista además del controlador HITACHI HD44780, que es el más utilizado actualmente y que se controla de forma simple mediante los pines de control y de datos, que pueden verse en el siguiente diagrama de bloques:

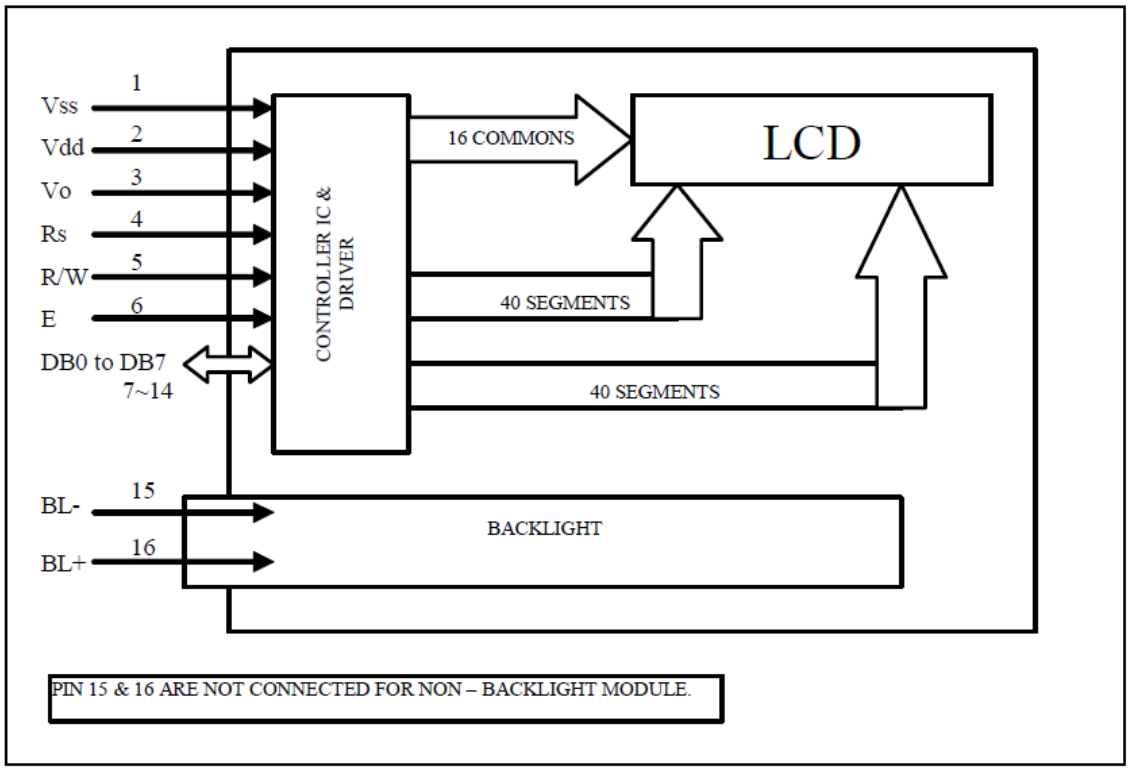


Imagen 9 – Diagrama de bloques de la pantalla LCD con controlador HITACHI HD44780

El juego de instrucciones mediante el que podemos configurar su funcionamiento se encuentra en anexos.

1.8.1.1.1.3 Sensor de presión diferencial

Siguiendo el razonamiento descrito en el capítulo previo, el sensor de presión ha sido elegido con las siguientes características:

- Presión diferencial
- Salida en tensión: 0 – 10V
- Rango de trabajo: 0 – 2bar
- Tensión de alimentación: 15 V DC

Más precisamente el modelo es el 4311B de la serie 4000 de “Bailey & Mackey”.

Imagen 10 – Ilustración del sensor de presión diferencial seleccionado

Entre las características eléctricas del dispositivo cabe destacar que exige una resistencia de entrada del circuito al que vaya a conectarse de mínimo 10K para que se cumplan las prestaciones indicadas en la datasheet.

1.8.1.1.1.4 Solenoides - electroválvulas

Los solenoides seleccionados son solenoides de tecnología latch, lo que significa que son abiertos y/o cerrados mediante el envío de un pulso de tensión. Una vez enviado el pulso, la electroválvula queda en la posición definida sin necesidad de aplicarle tensión, lo que supone un ahorro energético.

También se ha tomado como requisito el tener una conexión a 3 hilos (o doble bobina), por las razones descritas en el capítulo anterior.

Así pues el solenoide seleccionado ha sido el 018634S de la marca “burkert”. Que además presenta las siguientes características:

- Conexión a 3 hilos con común positivo.
- Tensión de alimentación: 12 V DC
- Potencia bobina: 5 W
- Duración impulso: > 20 ms

Imagen 11 – Ilustración del solenoide seleccionado

1.8.1.1.1.5 Alimentación del sistema

La alimentación se ha realizado mediante pilas. Dado que se deben alimentar dispositivos hasta con una tensión de 15 V DC, se han debido emplear 5 celdas de 3,5 V cada una para conseguir obtener una tensión superior que luego pueda ser adaptada mediante el uso de los reguladores de tensión.

Así pues la tensión de alimentación exacta es de 17,5 V DC.

1.8.1.1.1.6 Reguladores de tensión

Para alimentar los distintos componentes que requieren una tensión menor a la de alimentación, también en DC, se han empleado diversos reguladores de tensión, que además permiten ofrecer una señal de tensión más estable.

1.8.1.1.1.6.1 Reguladores 15 V, 12V y 5V

El modelo utilizado para estas 3 tensiones es el mismo: el modelo L78 de la marca "ST".

Así pues para la regulación de las tensiones a 15, 12 y 5 voltios los modelos utilizados serán L7815CV, L7812CV y L785CV respectivamente.

La conexión de los pines se ilustra en la siguiente imagen:

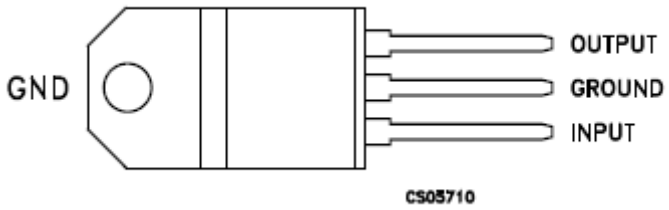


Imagen 12 – Conexión de pines del regulador de tensión L78 de la marca ST

1.8.1.1.1.6.2 Regulador 3.3 V

El regulador utilizado para obtener la tensión de 3.3 V para la alimentación del microcontrolador es el modelo LD1117AV33 también de la marca ST.

La conexión de los pines se ilustra en la siguiente imagen:

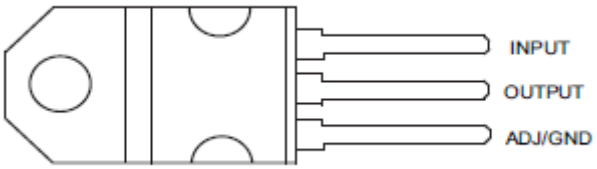


Imagen 13 – Conexión de pines del regulador de tensión LD1117AV33 de la marca ST

1.8.1.1.1.7 Driver para el control de la alimentación de las electroválvulas

El driver utilizado para permitir el paso de corriente o no hacia las bobinas que controlan las electroválvulas ha sido el modelo TD62783AP de la marca TOSHIBA.

Dicho modelo de driver presenta la siguiente asignación de pines:

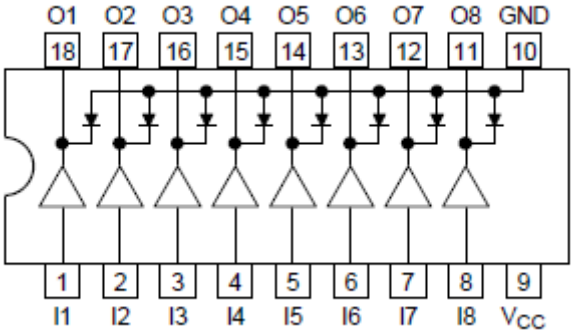


Imagen 14 – Asignación de pines del driver TD62783AP de la marca TOSHIBA

1.8.1.1.2 Funcionamiento y descripción del conexionado del sistema de aplicación

En los siguientes subapartados se describirá el conexionado del sistema de aplicación cuya representación se encuentra en el documento de planos.

1.8.1.1.2.1 Conexión microcontrolador – solenoides

El primer conexionado propuesto para accionar las electroválvulas consistía en emplear dos pines del microcontrolador para cada electroválvula: uno para la apertura y el otro para el cierre.

No obstante, finalmente se propuso el conexionado actual que permite emplear menos pines siempre que el número de filtros a controlar sea al menos 3.

Este conexionado consiste en asignar un pin a la apertura, otro al cierre de electroválvulas, y luego un pin para cada electroválvula a controlar. De este modo, si se

quiere abrir la electroválvula 3 por ejemplo, activaríamos la salida de la electroválvula 3 y la de apertura. De este modo el número de pines a utilizar es:

$$n = número\ electroválvulas + 2$$

1.8.1.1.2.2 Conexión sensor de presión – microcontrolador

La salida del sensor de presión es de 0 – 10 V, mientras que la entrada analógica de nuestro microcontrolador presenta un rango de 0 – 3.3 V. Por lo tanto se ha debido implementar un circuito de adaptación de la señal.

El circuito elegido ha sido un divisor de tensión utilizando resistencias de precisión (para una conversión más exacta de la tensión).

La proporción necesaria en las resistencias utilizadas es:

$$\frac{R2}{R1 + R2} = 0.33$$

Así pues queda definir el valor exacto de las resistencias, para lo que se han tenido en cuentas los siguientes aspectos:

- Requisito del sensor de presión de resistencia mínima de entrada: 10 KΩ

  Este valor es en nuestro caso la suma de las resistencias del divisor resistivo:

  $$Re = R1 + R2$$

- Hipótesis de intensidad de entrada del microcontrolador (leakage current en anexos) despreciable frente a intensidad en el divisor resistivo. Esta hipótesis nos permite despreciar la corriente de entrada del microcontrolador para que la fórmula del divisor resistivo pueda ser considerada como válida.

leakage current (see Note 1)

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
$I_{lkg(P1.x)}$	Leakage current (see Note 1)	Port P1	$V_{(P1.x)}$ (see Note 2)	V_{CC} = 2.2 V/3 V			±50	nA
$I_{lkg(P2.x)}$		Port P2	$V_{(P2.3)}$ $V_{(P2.4)}$ (see Note 2)				±50	
$I_{lkg(P6.x)}$		Port P6	$V_{(P6.x)}$ (see Note 2)				±50	

NOTES: 1. The leakage current is measured with V_{SS} or V_{CC} applied to the corresponding pin(s), unless otherwise noted.
2. The port pin must be selected as input and there must be no optional pullup or pulldown resistor.

Imagen 15 – Extracto del datasheet con detalle del leakage current

La corriente de entrada al microcontrolador es de 50 nA, que es un valor extremadamente pequeño por lo que será fácil tener una corriente en el divisor de resistencias que permita despreciar su efecto.

- Bajo consumo: Como es sabido, un divisor resistivo presenta un continuo consumo, inversamente proporcional a la suma del valor de las resistencias que lo componen, según la fórmula:

$$P = \frac{V^2}{(R1 + R2)}$$

Así pues se debe buscar una resistencia alta para que el consumo sea bajo, pero sin que la corriente sea comparable a la de entrada del microcontrolador.

Finalmente los valores seleccionados han sido: R1 = 670K y R2 = 330K.

Estos valores permiten despreciar sin problemas el "leakage current" del microcontrolador (200 veces inferior a la corriente del divisor) y tienen valores altos para que la corriente del divisor no sea muy importante (10 µA).

1.8.1.1.2.3 Conexión botones – microcontrolador

La conexión entre los botones y el microcontrolador está realizada en configuración pull-up. De forma que cuando el botón esté pulsado el microcontrolador reciba una tensión de 0V (Low) y cuando no esté presionado, la tensión recibida sea de 3.3V (High), con lo que no tienen lugar falsos estados de tensiones en el pin del microcontrolador derivados del ruido en el ambiente.

1.8.1.1.2.4 Conexión microcontrolador – LCD

El conexionado entre el microcontrolador y el LCD es directo y se hace respetando la función de pines según sean de control o de datos.

1.8.1.2 Sistema para el desarrollo del software

También en el documento de planos se presenta el conexionado realizado para el desarrollo y depuración del software creado.

Como se verá a continuación, existen componentes en común con el esquema de aplicación real, por lo que nos centraremos únicamente en las variantes.

1.8.1.2.1 LEDs en lugar de solenoides para las salidas digitales

Para poder observar visualmente la activación de las salidas digitales del microcontrolador se han conectado 5 LEDs con una resistencia para limitar la corriente, que simulan las 5 electroválvulas que como máximo puede controlar nuestro sistema.

1.8.1.2.2 Potenciómetro en lugar del sensor de presión

Para simular la tensión de 0 a 3.3V que se tendrá en el sistema real gracias al sensor de presión, se ha empleado un potenciómetro que, conectado a la misma tensión de 3.3V, permite variar su tensión de salida entre 0 y 3.3V.

1.8.1.2.3 LED indicador de la habilitación de la interrupción del timer

Para facilitar la depuración del software, se ha conectado además un LED que está activo siempre que la interrupción del timer esté habilitada. De este modo, se puede observar de forma visual el funcionamiento interno del microcontrolador en lo que a la habilitación del timer se refiere.

1.8.2 Software

En este apartado se describen las características del software desarrollado para el dispositivo. Esta descripción se dividirá en distintos subapartados y se complementará con los anexos para facilitar su comprensión. El código completo en lenguaje C se encuentra en el anexo.

1.8.2.1 Configuración de dispositivos

Antes del diseño del código para nuestro dispositivo, se deben realizar las configuraciones para definir el comportamiento de algunos componentes.

1.8.2.1.1 Microcontrolador

En cuanto al microcontrolador, se deben configurar los siguientes aspectos:

- **Funciones de los pines:** Puesto que cada pin del microcontrolador está preparado para poder desempeñar distintas funciones, debemos indicar cuál de ellas queremos para nuestra aplicación. En nuestro caso se requieren las siguientes (Los valores correspondientes en cada registro están especificados en anexos)
- **Interrupciones:** Como se ha indicado anteriormente, se requerirá el uso de 3 interrupciones, que se configurarán como sigue:
    - **Interrupción puerto P1:** Se utilizará para recibir las señales de los botones. la configuración deberá ser la siguiente (los valores correspondientes se pueden consultar en anexos):
    - **Interrupción TIMER A:** El Timer ha sido configurado para entrar en la interrupción asociada una vez por segundo utilizando un reloj externo de 32768 Hz. Los registros a manipular para la configuración del Timer son los siguientes (los valores correspondientes se pueden consultar en anexos):
    - **Interrupción ADC12:** Mediante esta interrupción, el valor de tensión recibido de 0 a 3.3V es convertido a un valor digital de 12 bits. La interrupción se produce cada vez que se completa una conversión, y los registros a configurar son los siguientes(los valores correspondientes se pueden consultar en anexos):

1.8.2.1.2 Pantalla LCD

La pantalla utilizada es una pantalla LCD con un controlador HITACHI HD44780.

Para nuestra aplicación se requiere que se puedan comunicar los datos mediante el uso de 4 pines (multiplexado). Así pues, se deberá especificar tanto este parámetro como los demás que definen el funcionamiento del LCD, como son el número de líneas utilizadas, que son 2 en nuestro caso, los puntos utilizados para representar cada carácter (7x5), la representación del cursor en la pantalla (apagado excepto en la pantalla de introducción de valor).

Los valores a introducir para la configuración de cada aspecto del controlador LCD 44780 se muestran en anexos.

1.8.2.2 Modo de funcionamiento del microcontrolador

El microcontrolador presenta distintos modos de funcionamiento que le permiten estar inactivo y a la espera de recibir interrupciones. Dichos modos difieren en el consumo energético según la cantidad de funcionalidades habilitadas.

Así pues, el modo elegido es el "LPM3" (Low Power Mode 3) por ser el de menor consumo que permite la utilización de la fuente de reloj ACLK, ligado al reloj piezoeléctrico externo.

1.8.2.3 Interrupciones utilizadas

En nuestro caso se han debido utilizar 3 tipos de interrupciones:

- **Interrupción de puerto I/O digital:** Para responder a la pulsación del botón, se debe implementar esta interrupción. De este modo, sólo cuando se pulse un botón se ejecutará el código asignado a esta interrupción.
  Mediante los botones podremos tanto navegar por el menú del dispositivo como introducir valores para los parámetros que se desee, así como también activar o desactivar ciertas funciones, entre otros.
- **Interrupción TIMER:** Mediante esta interrupción, que se activa cada vez que el registro de contaje llega al valor definido, se pueden realizar acciones periódicas. En nuestro caso se ha diseñado para que el ciclo sea de un segundo, y la interrupción se utiliza para contar tanto tiempos de limpieza, como tiempos de espera, de estabilización y otros como se detallará más adelante en este documento.
  El reloj utilizado consiste en un piezoeléctrico externo, ya que presenta mejores prestaciones en cuanto a estabilidad frente a cambios de temperatura.
  Como se verá en más detalle más adelante, esta interrupción puede ser deshabilitada durante ciertos espacios de tiempo sin afectar al funcionamiento de nuestro sistema para así reducir el consumo del microcontrolador.
- **Interrupción ADC12:** Esta interrupción se activará cada vez que se efectúe una conversión de un valor analógico en tensión a uno digital que quedará almacenado en un registro del microcontrolador para su posterior lectura. Dicha conversión es comandada por un reloj interno, cuya frecuencia no es tan

estable frente a la temperatura como el piezoeléctrico externo, pero es útil para realizar lecturas de la entrada analógica de forma continua.

Mediante esta interrupción se podrá leer el valor enviado por el sensor de presión y actuar según corresponda. Al contrario de la interrupción del timer, no podremos deshabilitar esta interrupción si queremos ser capaces de detectar determinadas averías en la instalación que tienen un efecto visible en la presión, tal y como veremos más adelante.

1.8.2.4 Descripción del funcionamiento del software

Una vez descrito el aspecto hardware, procede detallar lo relacionado con el software. Para ello se presentarán y explicarán las distintas partes que lo componen.

Imagen 16 – Diagrama de los elementos software del sistema

Como vemos en la imagen 16 se pueden distinguir distintas partes que interaccionan entre ellas.

Todas ellas están conectadas con el sistema de navegación de pantallas, pues dichco sistema es el intermediario entre el usuario y todas las funciones que presenta el dispositivo.

Además, como relaciones importantes entre los demás elementos, se pueden: citar las que relacionan la lectura de presiones, que puede activar un lavado cuando la presión es suficientemente alta, o activar una alarma si se lee una presión fuera de los rangos

de seguridad; Estas alarmas, al activarse, detienen el lavado si estaba en proceso y bloquean futuros lavados hasta que la alarma sea borrada; El contaje del tiempo, por su lado, interviene principalmente en la temporización de la secuencia de lavado así como en la lectura de las presiones para determinar si la lectura es estable o no.

A continuación se analizará el aspecto software del sistema presentando y describiendo todas sus partes.

1.8.2.4.1 Filtrado de ruido en los pulsadores

Empezaremos explicando que, al presionar un pulsador, lejos de producirse una señal rectangular limpia, se producen los denominados rebotes que consisten en variaciones de la tensión que se asemejan a rápidas pulsaciones. Estos rebotes deben ser distinguidos de las pulsaciones para poder ignorarlos.

Así pues, en el programa se comprueba que la señal se encuentre en estado "Low" durante un espacio de tiempo mínimo antes de proceder con las siguientes acciones.

1.8.2.4.2 Menús y tipos de pantallas

El método de navegación para el menú es el descrito en el capítulo anterior, utilizando una matriz donde en cada línea se tiene una cadena de caracteres y su correspondiente índice (de tipo "long int").

Estas cadenas de caracteres son en ocasiones el texto a mostrar en la pantalla (denominaremos entonces a dicha pantalla como de tipo 0), y en otras ocasiones son una referencia a pantallas especiales (que se distinguen por empezar con una secuencia de caracteres exclusiva).

Estas pantallas especiales pueden ser de dos tipos, la que denominaremos de tipo 1 y que solicita una respuesta de SI/NO por parte del usuario. Y la que denominaremos de tipo 2 y que permitirá al usuario introducir un valor numérico (como se ilustra en la imagen 17). Así pues estas pantallas son las que permitirán que el usuario configure el funcionamiento del controlador. Activando las funcionalidades a su gusto y definiendo los parámetros con los valores que desee.

Imagen 17 – Pantalla LCD mostrando una pantalla de tipo 2

A parte de los dos descritos, existe un tercer tipo de cadena de caracteres que dirige al usuario a un menú distinto, basado en una matriz distinta. Estos menús anidados han sido utilizados para la consulta del registro de alarmas e históricos, y para la definición de las temporizaciones de lavado para cada filtro. Cada uno de los menús tiene asociada una variable con el valor del número de pantallas que presentan, y modificando este valor se aumenta el tamaño del menú cuando se registran nuevas alarmas, o se disminuye al borrarlas. En el caso de las temporizaciones de lavado, dado que habrá más o menos dependiendo del número de filtros especificado, el menú también se ajustará al número de pantallas necesarias.

1.8.2.4.2.1 Operaciones asignadas a los botones

Dependiendo del tipo de pantalla en el que nos encontremos (0 1 ó 2), los botones deberán realizar unas acciones distintas. Mientras que en las tipo 0 realizará la operación necesaria sobre el índice para llegar a la siguiente pantalla, en las de tipo 1 y 2, servirá para modificar un valor (el de la variable asignada a la pantalla).

Las acciones realizadas según el tipo de pantalla en el que nos encontremos y el botón pulsado se muestran gráficamente en anexos.

1.8.2.4.2.2 Búsqueda de la pantalla objetivo y representación en el LCD

El nuevo índice resultante al pulsar un botón (principalmente si estábamos en pantalla tipo 0), es buscado en la matriz, si se encuentra, se identifica a qué tipo de pantalla corresponde, se representa en el LCD, y se actualiza la variable que definirá el tipo de comportamiento que deben tener los botones en la siguiente pulsación, como se muestra en el diagrama presente en anexos.

1.8.2.4.3 Medida de la presión y operaciones correspondientes

La función principal de nuestro dispositivo es la medición de la presión y la actuación en consecuencia. Los rangos de presión considerados son los de la siguiente representación:

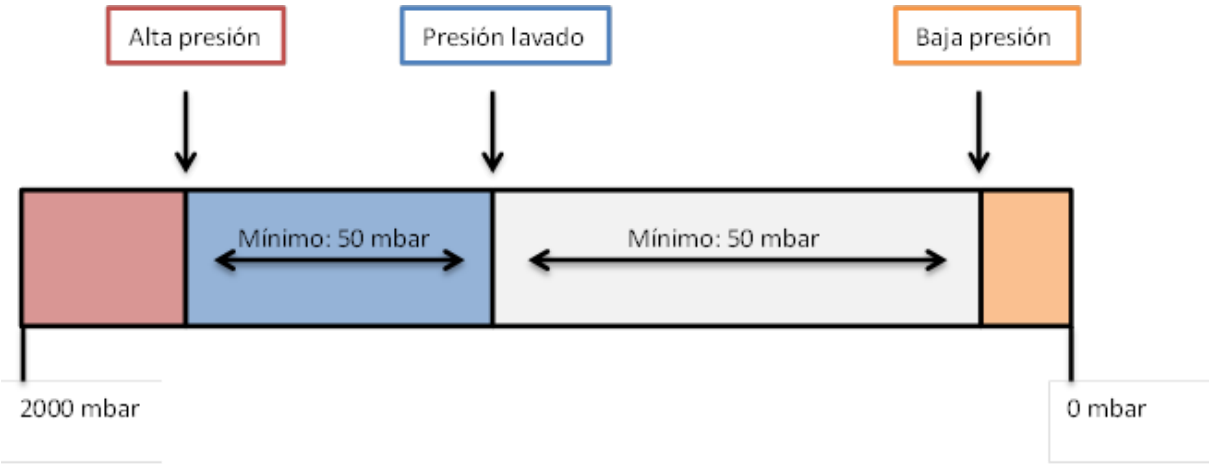


Imagen 18 – Representación de los rangos de presión considerados

Así pues, tras cada lectura de presión se comprueba siempre que la presión no esté por encima de una cierta presión máxima ni por debajo de una mínima, ambas definidas por el usuario. Si la presión supera uno de estos límites durante al menos el tiempo establecido por el usuario como tiempo de estabilización de la señal, se activará una alarma que impedirá la ejecución de nuevas limpiezas.

Si la presión está dentro del rango de presión de lavado (color azul de la imagen 18) y está activado el modo de limpieza por diferencia de presión, tras la espera de la estabilización, se ejecutará una secuencia de limpieza.

Si no está activado el modo de limpieza por diferencia de presión o la presión se encuentra en el rango gris de la imagen 18, no se realizará ninguna operación.

Esto es siempre y cuando no haya alarma por bucle infinito. En caso de haberla, las limpiezas serían temporizadas y la presión deberá situarse en el rango de color gris para que se desactive el modo de bucle infinito.

Este funcionamiento se encuentra representado con más detalle en anexos.

1.8.2.4.4 Habilitación/Deshabilitación del Timer para el contaje del tiempo

En el diagrama del apartado anterior, se observa que las distintas variables para contar el tiempo, como son las de estabilización, se incrementan en uno dependiendo de las circunstancias.

Para entender este incremento en dichas variables se debe recordar que se dispone de la interrupción del timer, que funciona de forma paralela a la de lecturas analógicas.

Además, puesto que existen momentos en los que el contaje del tiempo no es de utilidad en el funcionamiento del sistema, la interrupción del timer es deshabilitada mientras no se necesita para así ahorrar energía.

Concretamente la habilitación y deshabilitación de las interrupciones del timer se efectuarán según el diagrama en lenguaje de contactos (ya que se ha creído más adaptado al caso) representado en anexos.

1.8.2.4.5 Activación/Paro del lavado de filtros

Puesto que ya se han detallado las condiciones que causan la activación del lavado cuando el modo limpieza por diferencia de presión está activado, es momento de presentar las demás condiciones que pueden desencadenar el lavado.

Como ya se ha dicho previamente, las otras causas posibles para la ejecución del lavado son la activación manual y el modo de funcionamiento temporizado. Para ilustrar de mejor manera como estas condiciones están relacionadas entre ellas y con la activación del lavado, se presenta un diagrama (también en lenguaje de contactos) en el documento de anexos.

1.8.2.4.6 Operaciones realizadas durante la ejecución de la secuencia de lavado

Una vez vistas las causas que pueden iniciar un lavado, veremos en este subapartado cómo éste se desarrolla y qué operaciones van teniendo lugar.

La primera operación a realizar será la copia de los valores de las temporizaciones definidas por el usuario, en una matriz auxiliar para que así, aunque se modifiquen los valores durante el lavado, estos ya no se tengan en cuenta hasta el siguiente.

Se cuentan segundos para respetar las temporizaciones dedicadas a cada electroválvula, y en el momento que se ha terminado de operar con todos los filtros, se cierran las electroválvulas y se vuelve a desactivar el lavado.

Estas operaciones descritas con más detalle y considerando otros factores se encuentran representadas en forma de diagrama en anexos.

1.8.2.4.7 Registro de históricos y alarmas

Como se ha dicho previamente, los registros se almacenan en un menús anidados, que utilizan una matriz distinta a la del menú principal. De este modo, tanto al introducir nuevos registros como al borrar las alarmas, se posibilita la variación del tamaño del menú mediante el uso de una variable que contiene el número de pantallas del menú. Así pues este límite en el tamaño del menú se tiene en cuenta en el bucle utilizado para encontrar la pantalla a la que se debe acceder. Cuando se han revisado las pantallas correspondientes, se considera que la pantalla no se encuentra en el menú y se ignora.

Para ambos tipos de registros, se ha utilizado el mismo sistema, por lo que sólo se presentará un único estándar. Su principio de funcionamiento es el siguiente:

Se recibe un nuevo dato a almacenar. Si el registro se encontraba vacío, dicho dato se almacena en la primera línea sin más. Si por el contrario ya había algún registro, primero desplazaremos todos los antiguos una línea hacia abajo antes de almacenar el nuevo en la primera línea. En el caso de estar lleno el registro, se borraría el dato más antiguo.

El funcionamiento descrito se encuentra representado en forma de diagrama con más nivel de detalle en anexos.

1.9 Conclusiones

En el ámbito del presente proyecto ha sido diseñado tanto el aspecto hardware como el software del sistema controlador de lavado de filtros.

Las principales tareas en las que ha consistido el proyecto son:

- Estudio de los condicionantes derivados del entorno agrícola y las necesidades del cliente.
- Estudio de las posibles soluciones y valoración según los requerimientos del entorno y el cliente.
- Diseño de la parte software en línea con los requerimientos, como la facilidad de uso y el bajo consumo.
- Diseño de la parte hardware en línea con los requerimientos, como el bajo coste, la robustez y el bajo consumo.

Gracias a la realización del proyecto, no sólo se ha conseguido ofrecer al cliente la solución requerida, sino que también cabe valorar la experiencia adquirida con los condicionantes del entorno del proyecto así como con las herramientas con las que se ha llevado a cabo. Lo cual permitirá desarrollar futuros proyectos de una forma más eficiente.

1.9.1 Continuidad del proyecto

El presente proyecto abre determinadas tareas por resolver para su posible instalación como son:

- El diseño de la PCB del sistema y su fabricación.
- El diseño de la carcasa protectora.
- El ensamblaje de los distintos componentes y su posterior instalación y pruebas de funcionamiento.

1.9.2 Posibles mejoras futuras del sistema

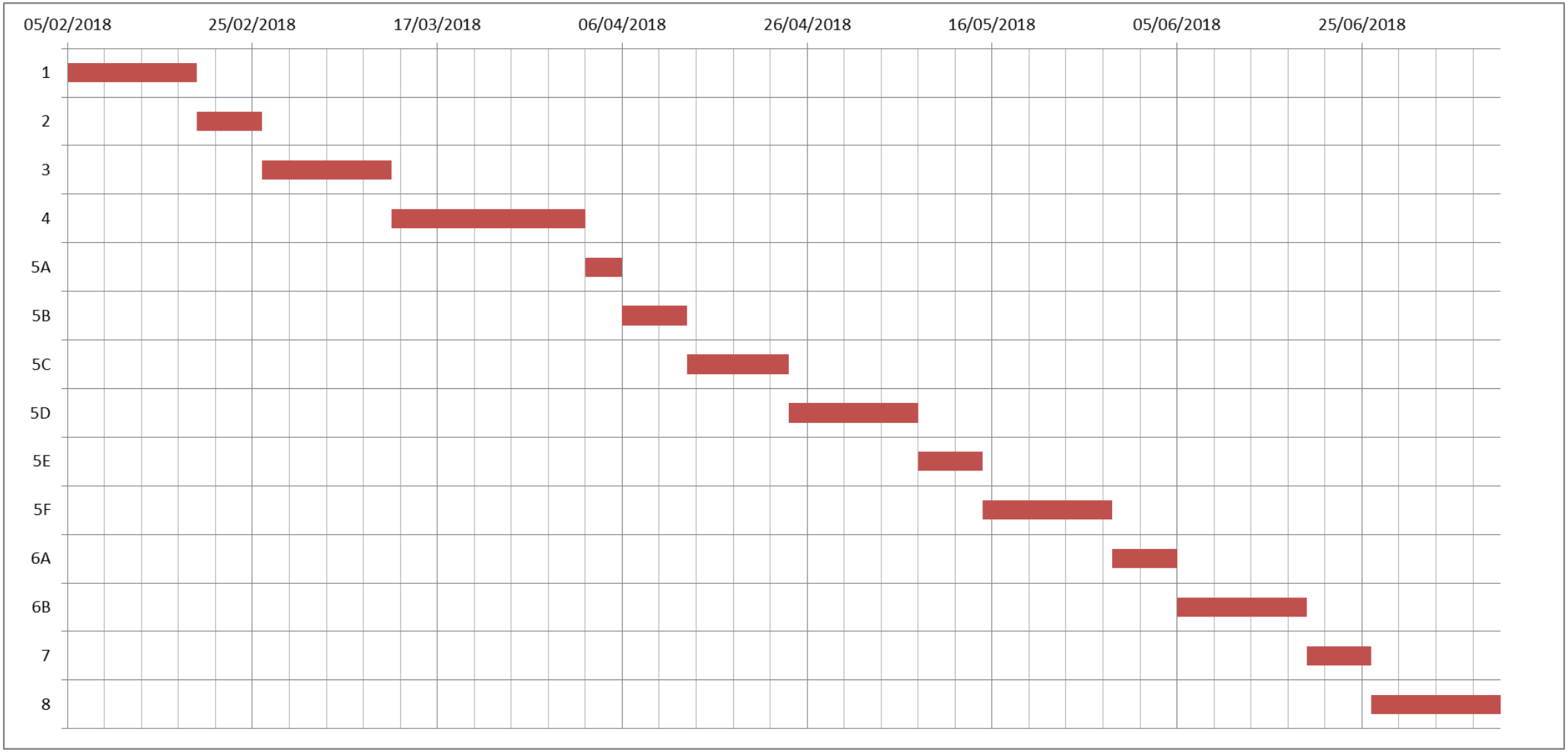
Al realizar el proyecto se han propuesto posibles mejoras para futuros dispositivos similares:

- **Fecha y hora en el dispositivo:** El disponer de la hora en el dispositivo permitiría, no sólo consultarla, sino poder relacionar los registros de alarmas e históricos con el momento en que han ocurrido, lo que sería muy interesante para el análisis del funcionamiento del sistema mediante el análisis de esta información.
- **Utilización de memoria externa:** En nuestro caso, un corte de alimentación causará la pérdida de los valores guardados por el usuario mediante la interfaz del dispositivo. Para evitar este problema, se propone el uso de una memoria externa tipo EEPROM (Electrically Erasable Programmable Read-Only Memory) para el almacenamiento de los datos de forma segura.

1.10 Planificación

A continuación se presenta la planificación seguida durante el desarrollo del proyecto. En primer lugar en forma de tabla y posteriormente de forma más gráfica en un diagrama de Gantt.

ACTIVIDAD	FECHA INICIO	DURACION	FECHA FIN
1. Definición de las funcionalidades requeridas para el dispositivo	05/02/2018	14	19/02/2018
2. Aprendizaje de uso de las herramientas software utilizadas	19/02/2018	7	26/02/2018
3. Aprendizaje del funcionamiento del microcontrolador y la pantalla LCD	26/02/2018	14	12/03/2018
4. Realización del montaje destinado al desarrollo del software	12/03/2018	21	02/04/2018
5. Desarrollo software			
5A. Configuración e inicialización del microcontrolador y la pantalla LCD	02/04/2018	4	06/04/2018
5B. Definición de las pantallas y opciones que compondrán el menú	06/04/2018	7	13/04/2018
5C. Diseño del método de navegación del menú	13/04/2018	11	24/04/2018
5D. Desarrollo de las acciones realizadas con la interrupción de los pulsadores	24/04/2018	14	08/05/2018
5E. Desarrollo de las acciones realizadas con la interrupción del timer	08/05/2018	7	15/05/2018
5F. Desarrollo de las acciones realizadas con la interrupción del ADC12	15/05/2018	14	29/05/2018
6. Diseño del sistema final de aplicación real			
6A. Selección de componentes	29/05/2018	7	05/06/2018
6B. Diseño de los esquemas y planos	05/06/2018	14	19/06/2018
7. Documentación técnica para la empresa	19/06/2018	7	26/06/2018
8. Redacción de los documentos del proyecto	26/06/2018	14	10/07/2018


05/02/2018
25/02/2018
17/03/2018
06/04/2018
26/04/2018
16/05/2018
05/06/2018
25/06/2018
1
2
3
4
5A
5B
5C
5D
5E
5F
6A
6B
7
8

1.11 Orden de prioridad entre los documentos

Para el presente proyecto, se define el orden de prioridad de los documentos como el establecido en la norma UNR 157001:2014.

2 ANEXOS

2.1 Registros del microcontrolador

2.1.1 Definición de función de los pines del microcontrolador

- **Pines utilizados como entrada o salida digital:** Tanto para los botones, como para la pantalla y para la activación de las electroválvulas se requerirá el uso de estos pines. Los registros a manipular serán:
  - Registro PxSEL: Función I/O
  - Registro PxDIR: Dirección Input
- **Pin utilizado como entrada analógica:** Este pin posibilitará la lectura del sensor de presión. Puesto que el pin presenta la función I/O y la de lectura analógica de tensión, se deberá especificar la función configurándose como sigue:
    - Registro P6SEL: Función de módulo periférico

Tal y como recomienda el fabricante, los pines no utilizados se configurarán, para un mayor ahorro de energía según:

- Registro PxSEL: Función I/O
- Registro PxDIR: Dirección Output
- Registro PxOUT: Output Low

Function Select Registers PxSEL

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function – I/O port or peripheral module function.

Bit = 0: I/O Function is selected for the pin

Bit = 1: Peripheral module function is selected for the pin

Direction Registers PxDIR

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other module functions must be set as required by the other function.

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

Output Registers PxOUT

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function and output direction.

Bit = 0: The output is low

Bit = 1: The output is high

2.1.2 Registros de configuración de la interrupción de los puertos P1 y P2

- Registro P1IES: Interrupción con flanco descendente. Ya que los pulsadores están conectados en modo pull-up (cuando se pulsa el botón, la señal es "Low").
- Registro P1IE: Habilitar interrupción. Se deberá habilitar para los pines a los que tenemos conectados los botones. Para los demás, lo deshabilitamos para evitar posibles señales de ruido.
- Registro P1IFG: Flag de las interrupciones. En este registro, se activará el bit correspondiente al pin del puerto 1 en el que se ha producido el flanco descendente. Para empezar lo ponemos a 0x00 para que no haya interrupciones pendientes.

Interrupt Flag Registers P1IFG, P2IFG

Each PxIFGx bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge occurs at the pin. All PxIFGx interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Each PxIFG flag must be reset with software. Software can also set each PxIFG flag, providing a way to generate a software initiated interrupt.

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

Interrupt Edge Select Registers P1IES, P2IES

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

Bit = 0: The PxIFGx flag is set with a low-to-high transition

Bit = 1: The PxIFGx flag is set with a high-to-low transition

Interrupt Enable P1IE, P2IE

Each PxIE bit enables the associated PxIFG interrupt flag.

Bit = 0: The interrupt is disabled

Bit = 1: The interrupt is enabled

2.1.3 Registros de configuración del Timer A

- Registo TACTL: Fuente de reloj: ACLK, que es obtenida de la señal del reloj externo. Modo de contaje: Modo arriba; el timer cuenta hasta el valor almacenado en el registro TACCR0.
- Registro TACCR0: En cada ciclo se cuentan TACCR0 + 1 oscilaciones, por lo que el valor de TACCR0 para provocar interrupciones a una frecuencia de 1 Hz deberá ser TACCR0 = 32767.
- Registro TACCTL0: Habilitación de la interrupción producida al contar hasta TACCR0.

TACTL, Timer_A Control Register

TASSELx	Bits 9-8	Timer_A clock source select 00 TACLK 01 ACLK 10 SMCLK 11 INCLK
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: the timer is halted 01 Up mode: the timer counts up to TACCR0 10 Continuous mode: the timer counts up to 0FFFFh 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled

TACCTLx, Capture/Compare Control Register

CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled

2.1.4 Registros de configuración del ADC12

- Registro ADC12CTL0: Mediante este registro se activa y se habilitan las conversiones.
- Registro ADC12CTL1: En este registro se debe seleccionar el reloj utilizado para el lanzamiento cíclico de las conversiones.
- Registro ADC12MCTL0: Se especificará en este registro el canal utilizado para recibir la señal analógica.
- Registro ADC12IE : Este registro permite la habilitación de la interrupción de cada canal analógico.

ADC12CTL0, ADC12 Control Register 0

ADC12ON	Bit 4	ADC12 on 0 ADC12 off 1 ADC12 on

ENC	Bit 1	Enable conversion 0 ADC12 disabled 1 ADC12 enabled
ADC12SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC12SC and ENC may be set together with one instruction. ADC12SC is reset automatically. 0 No sample-and-conversion-start 1 Start sample-and-conversion

ADC12CTL1, ADC12 Control Register 1

ADC12 SSELx	Bits 4-3	ADC12 clock source select 00 ADC12OSC 01 ACLK 10 MCLK 11 SMCLK
CONSEQx	Bits 2-1	Conversion sequence mode select 00 Single-channel, single-conversion 01 Sequence-of-channels 10 Repeat-single-channel 11 Repeat-sequence-of-channels

ADC12MCTLx, ADC12 Conversion Memory Control Registers

INCHx	Bits 3-0	Input channel select 0000 A0 0001 A1 0010 A2 0011 A3 0100 A4 0101 A5 0110 A6 0111 A7 1000 Ve_{REF+} 1001 V_{REF-}/Ve_{REF-} 1010 Temperature sensor 1011 $(AV_{CC} - AV_{SS}) / 2$ 1100 $(AV_{CC} - AV_{SS}) / 2$ 1101 $(AV_{CC} - AV_{SS}) / 2$ 1110 $(AV_{CC} - AV_{SS}) / 2$ 1111 $(AV_{CC} - AV_{SS}) / 2$

ADC12IE, ADC12 Interrupt Enable Register

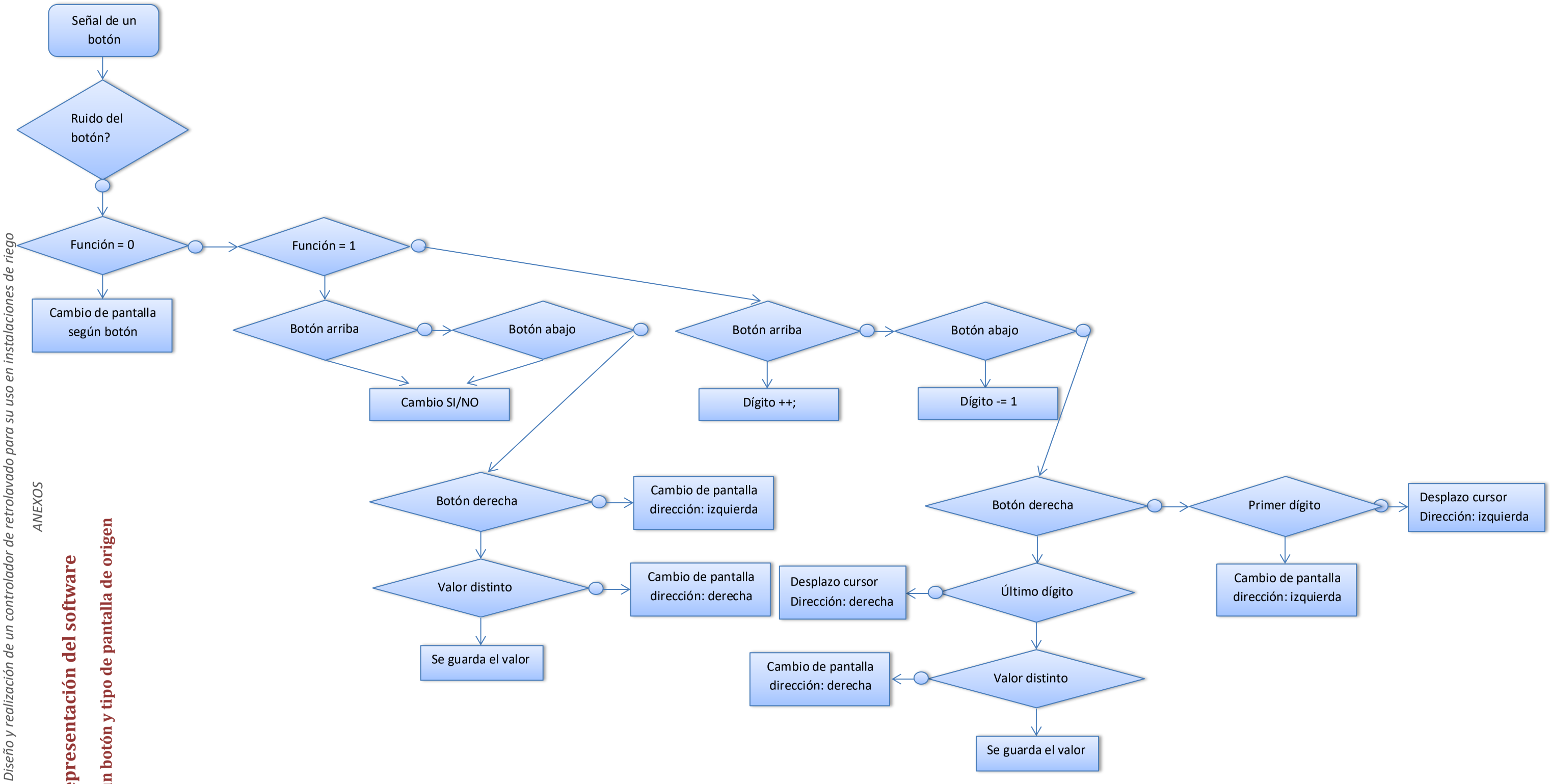
ADC12IEx	Bits 15-0	Interrupt enable. These bits enable or disable the interrupt request for the ADC12IFGx bits. 0 Interrupt disabled 1 Interrupt enabled

2.2 Juego de instrucciones de la pantalla LCD (HD44780U)

COMMAND	**RS**	**R/W**	**DB 7**	**DB 6**	**DB 5**	**DB 4**	**DB 3**	**DB 2**	**DB 1**	**DB 0**	**DESCRIPTION**	**Executing time** fosc=250khz
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears Display & Returns to Address 0.	1.64ms
Cursor at Home	0	0	0	0	0	0	0	0	1	x	Returns Cursor to Address 0. Also returns the display being shifted to the original position. DDRAM contents remain unchanged.	1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	I/D: Set Cursor Moving Direction I/D=1: Increment I/D=0: Decrement S: Specify Shift of Display S=1: The display is shifted S=0: The display is not shifted	40µs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Display D=1: Display on D=0: Display off Cursor C=1: Cursor on C=0: Cursor off Brink B=1: Brink on B=0: Brink off	40µs
Cursor / Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Moves cursor or shifts the display w/o changing DD RAM contents S/C=0: Cursor Shift (RAM unchanged) S/C=1: Display Shift (RAM unchanged) R/L=1: Shift to the Right R/L=0: Shift to the Left	40µs
Function Set	0	0	0	0	1	DL	N	F	x	x	Sets data bus length (DL), # of display lines (N), and character fonts (F). DL=1: 8 bits F=0: 5x7 dots DL=0: 4 bits F=1: 5x10 dots N=0: 1 line display N=1: 2 lines display	40µs
Set CG RAM Address	0	0	0	1	Character Generator (CG) RAM Address						Sets CG RAM address. CG RAM data is sent and received after this instruction.	40µs
Set DD RAM Address	0	0	1	Display Data (DD) RAM Address / Cursor Address							Sets DD RAM address. DD Ram data is sent and received after this instruction.	40µs
Busy Flag / Address Read	0	1	BF	Address counter used for both DD & CG RAM address							Reads Busy Flag (BF) and address counter contents.	40µs
Write Data	1	0	Write Data								Writes data into DDRAM or CGRAM.	46µs
Read Data	1	1	Read Data								Reads data from DDRAM or CGRAM.	46µs

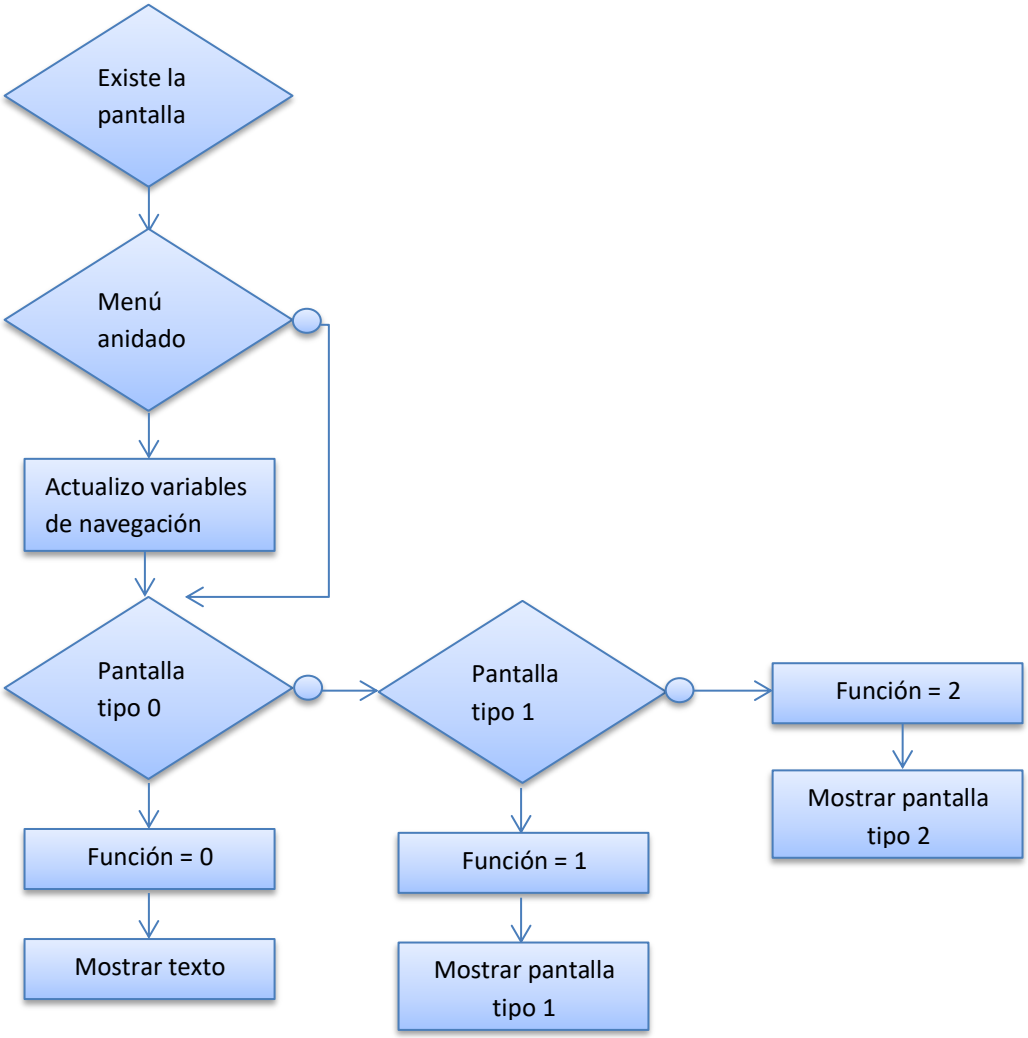
2.3 Diagramas de representación del software

2.3.1 Operaciones según botón y tipo de pantalla de origen



ANEXOS

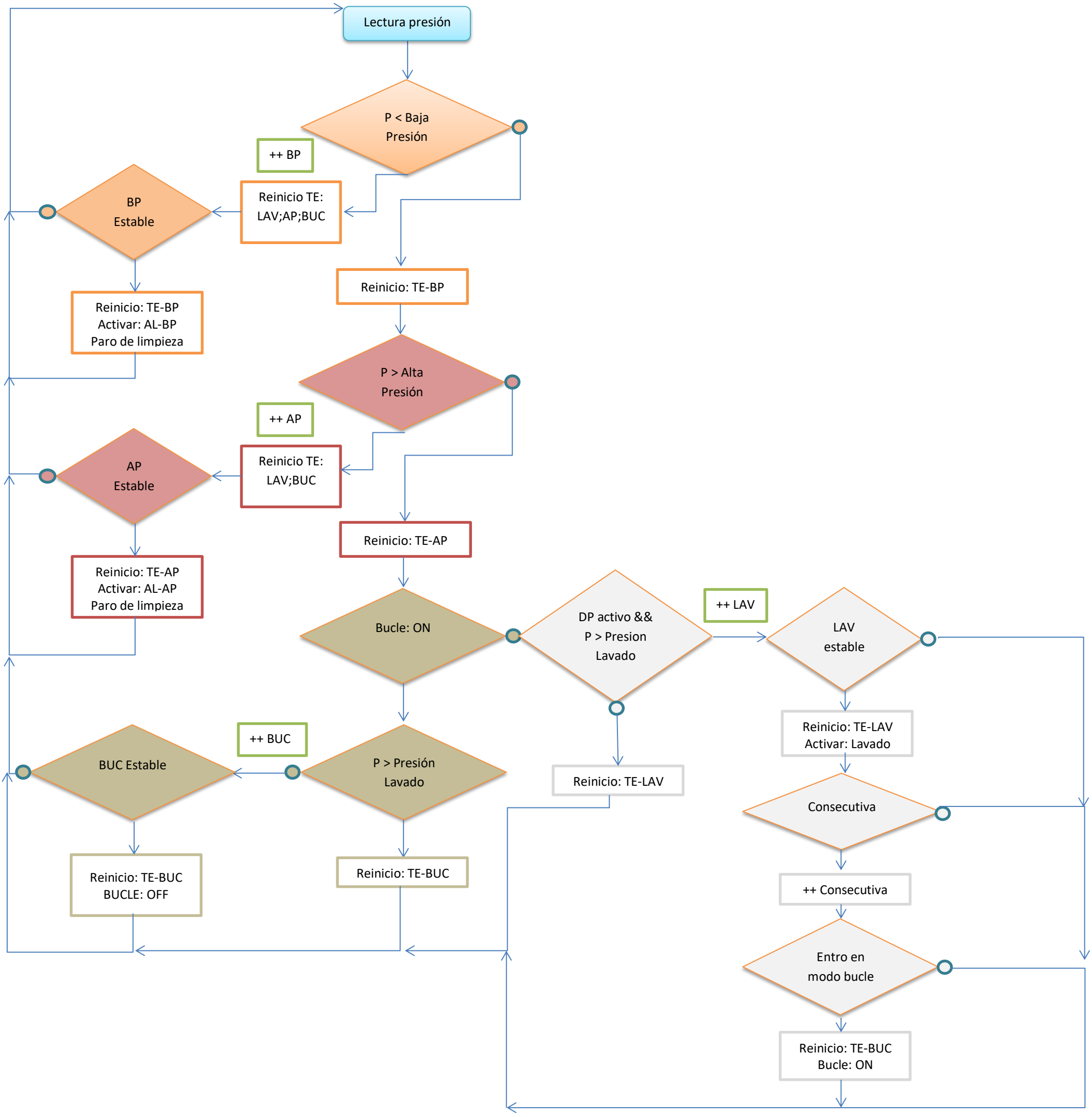
2.3.2 Operaciones según la pantalla de destino



2.3.3 Lectura de la presión y acciones correspondientes

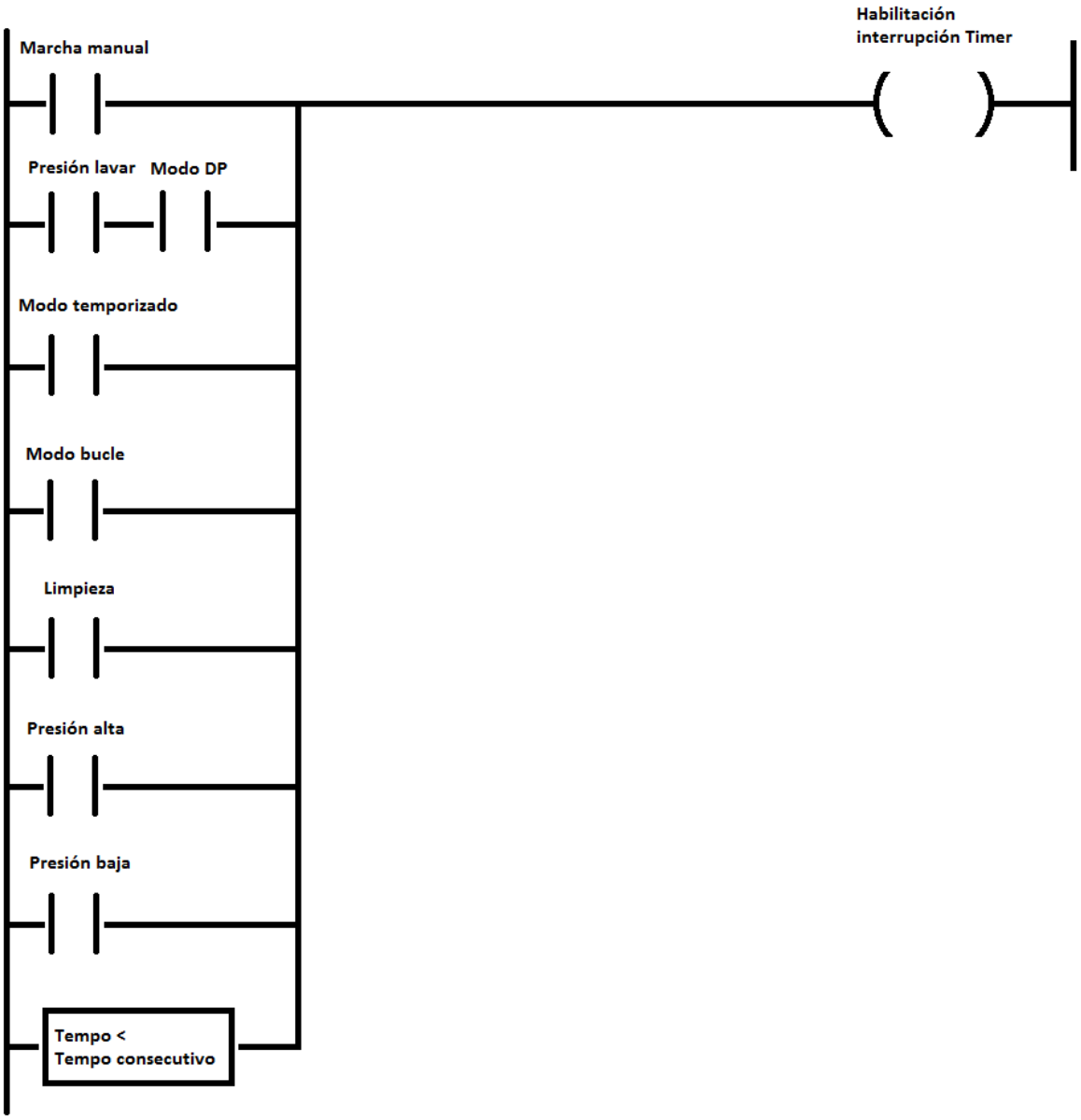
2.3.3.1 Leyenda

AP: Alta presión; BP: Baja presión; TE: Tiempo Estable; LAV: Lavado; BUC: Bucle; AL: Alarma



ANEXOS

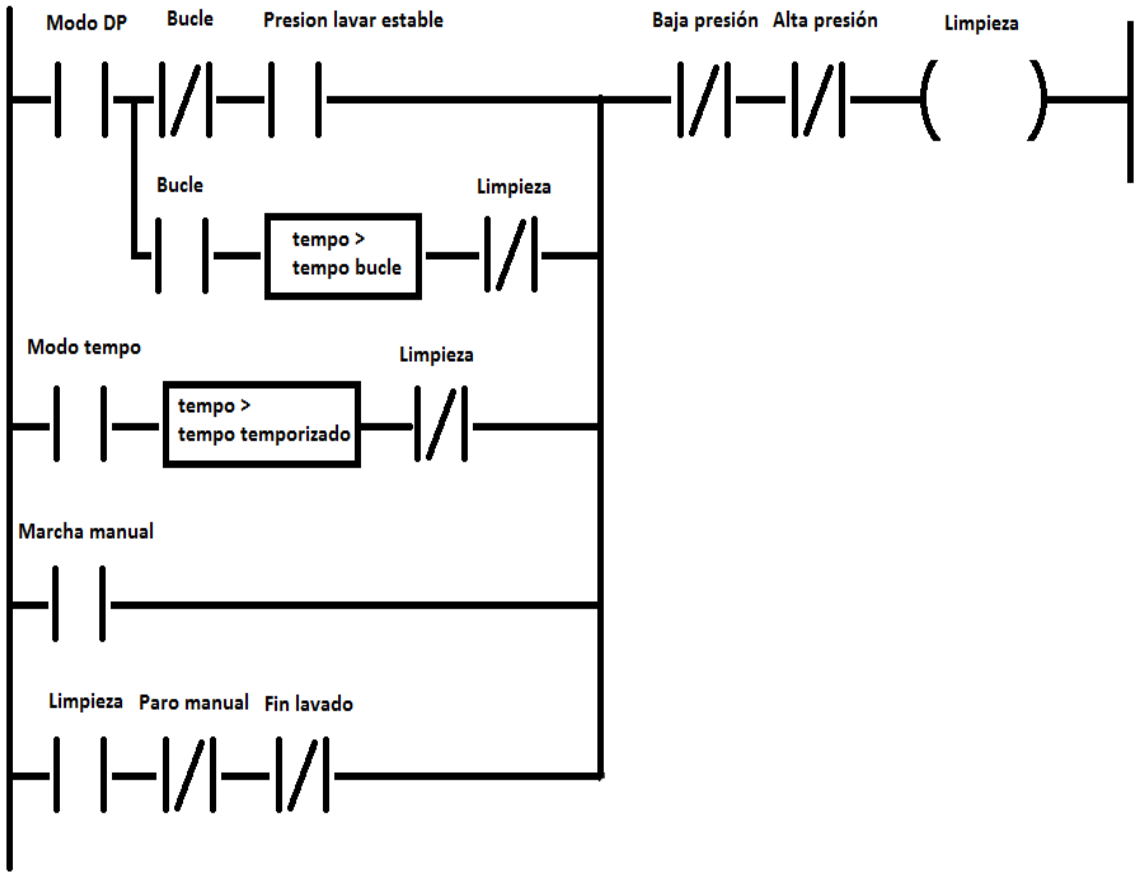
2.3.4 Condiciones para la Habilitación/Deshabilitación del Timer



Para una mejor comprensión del diagrama se debe saber que la variable “Tempo” es utilizada tanto para el contaje de las temporizaciones de la secuencia de lavado, como para contar el tiempo transcurrido desde el último lavado por si se produce otro lavado pronto y se debiera considerar consecutivo. Una vez superado el tiempo definido para considerarse lavado consecutivo, se deshabilitarían las interrupciones del timer.

Así pues la variable “Tempo” reinicia su valor cuando se empieza la limpieza y cuando se termina.

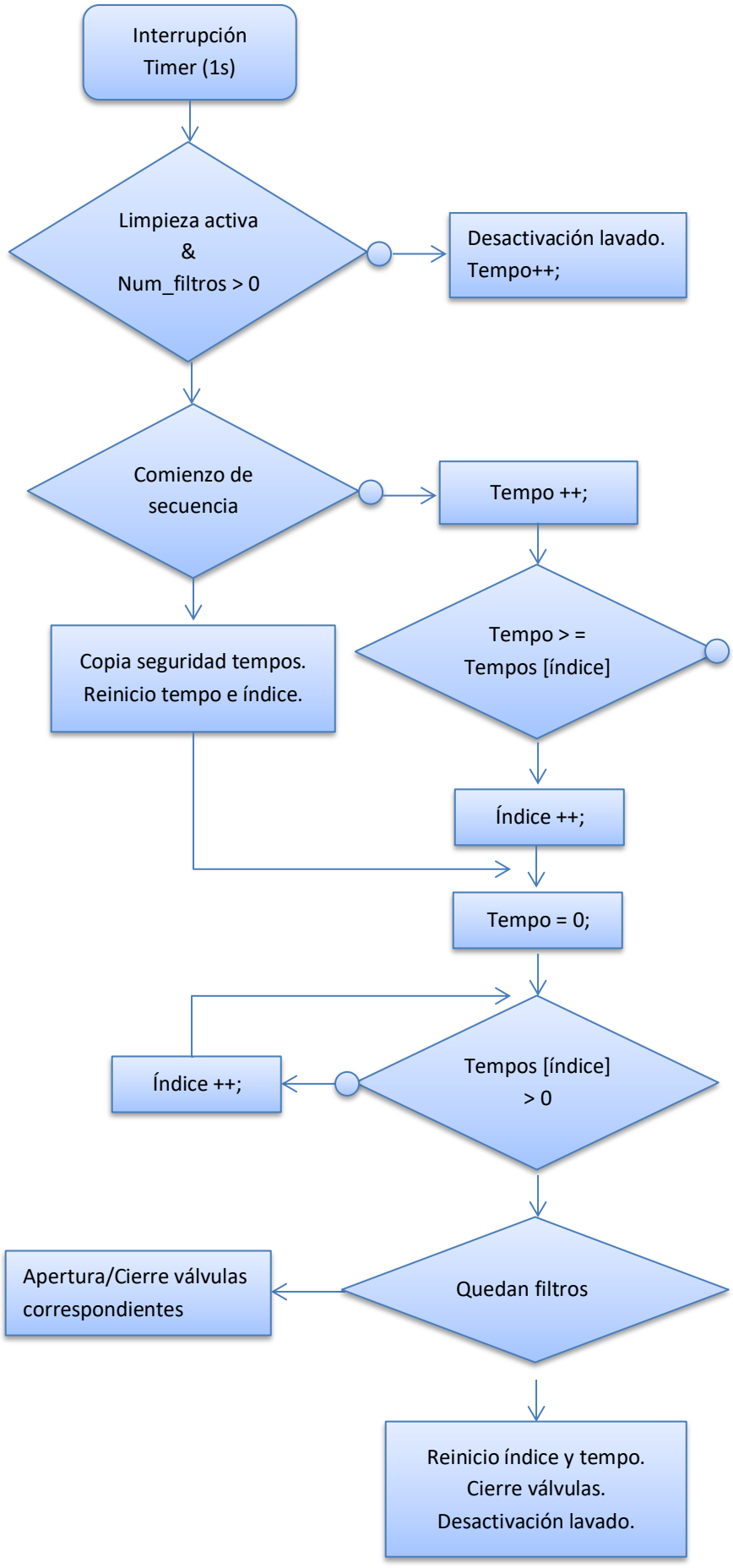
2.3.5 Activación/Paro del lavado



Como aclaraciones cabe decir que la variable “tempo bucle” contiene el valor del espacio de tiempo entre lavados cuando se está en modo bucle infinito (funcionamiento temporizado).

Por su lado, la variable “tempo temporizado” contiene el valor del espacio de tiempo entre lavados cuando se está en modo temporizado.

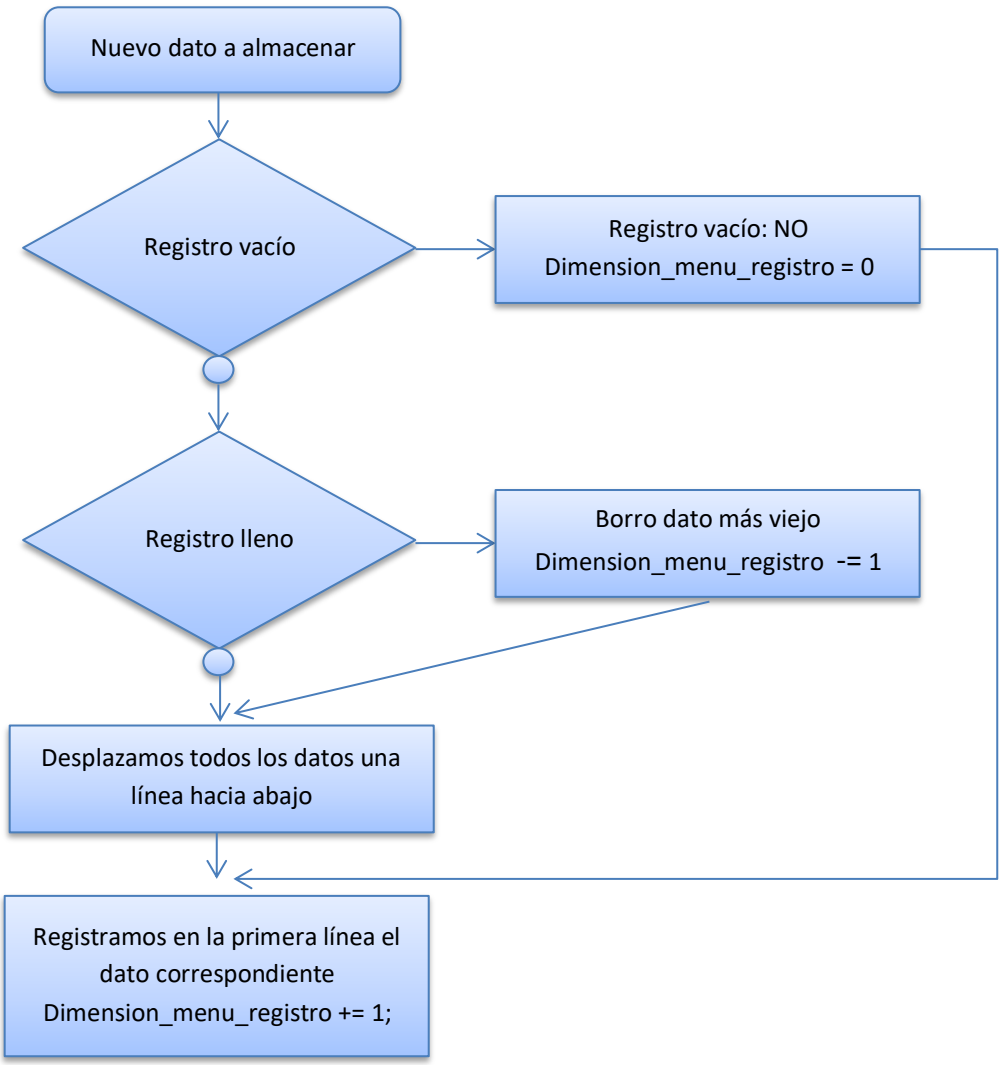
2.3.6 Operaciones a realizar en la secuencia de lavado



Como vemos, este diagrama se ejecuta una vez por segundo. Y mientras no se alcance el valor de tiempo definido para el filtro, la única acción a realizar es incrementar la variable tempo. Cuando se llega a dicho valor, se incrementa el índice para apuntar al siguiente valor de temporización.

Finalmente, si el valor del índice al que se llega todavía corresponde a algún filtro, se cierra la válvula actual y se abre la siguiente. En caso de ser la última, se cerrará la válvula actual y se desactivará el lavado, para que se deje de entrar en la ejecución del lavado.

2.3.7 Funcionamiento de los registros de históricos de lavado y alarmas



2.4 Código software del sistema (lenguaje de programación: C)

```
#include "io430.h"
#include "math.h"


//-------------------------------------------------------------
//      Funciones
//-------------------------------------------------------------

void delay(int n); //universal Delay routine
void s_inst_lcd(void); //Send Instruction Select routine on LCD
void s_data_lcd(void); //Send Data Select routine on LCD
void s_latch_lcd(void); // Latch Data/Instruction on LCD
void lcd_escribe1(char texto[32]); // Escribe en la primera linea hasta llegar a los 16 caracteres, luego pasa a la segunda
void lcd_escribe2(int linea, char texto[16]); // Se escribe el texto introducido en la linea especificada
int compara_strings (char a[], char b[]);

//-------------------------------------------------------------
//      Variables
//-------------------------------------------------------------

// ::: Variables para limpieza

int limpieza = 0; // Cuando vale 1 se debe limpiar, cuando vale 0 no (si se estaba limpiando se detiene)
int limpieza_anterior = 0; // Sirve para detectar los canvios de flanco de la variable "limpieza"
int indice_limpieza = 0; // Para saber qué salida toca activar en cada momento

int tempo_limpieza = 50; // Variable usada para contar los tiempos de lavado y reposo de cada estación en la secuencia de lavado así como los tiempos de espera entre lavados en el modo temporizado
int tempo_estable_DPlavar = 0; // Variable usada para contar el tiempo de estabilizacion de la presion para lavar
int tempo_estable_alta_presion = 0; // Variable usada para contar el tiempo de estabilizacion de la presion para saltar alarma por baja presion
int tempo_estable_salir_bucle = 0; // Variable usada para contar el tiempo de estabilizacion de la presion para considerar que la presion es normal y se puede salir del bucle infinito
int tempo_estable_baja_presion = 0; // Variable usada para contar el tiempo de estabilizacion de la presion para saltar alarma por poca DP

long int temposdef [9] = {0,0,0,0,0,0,0,0,0}; // Tempos usados para la secuencia de limpieza, copiados de temposbas o temposint y preservados para que no puedan modificarse mientras se está limpiando
                                              // Su valor corresponde al numero de veces que se deberá entrar en la interrupcion del timer para alcanzar los segundos definidos por el usuario.
                                              // Este valor depende por tanto del asignado a TACCR0 mediante la variable "valor_timer"
long int num_filtros_def = 0; // Número de filtros considerado para la secuencia de limpieza, copiado del valor editable por el usuario de modo que no se pueda modificar mientras se está limpiando

// ::: Variables editables por el usuario
long int *variable_funcion; // Puntero que apunta a la variable a modificar en la función. Mismo puntero para todas las funciones
long int variable_funcion_copia; // Sirve para no modificar la original hasta que el usuario confirme la selección. Misma variable para todas las funciones

long int crear_alarma = 0;
long int variable_conf_borrar_alarmas = 0;

long int f_conf_borrar_alarmas = 0;
long int f_conf_borrar_historico = 0;

long int f_intval_num_filtros = 5;
long int temposbas [9] = {4,2,3,2,1,0,2,0,1}; // Array con los valores de temporizaciones basicas introducidas por el usuario
long int f_conf_temposbas = 1; // Confirmacion de uso de temporizaciones basicas
```

```
51. long int temposint [9] = {0,0,0,0,0,0,0,0,0}; // Array con los valores de
    temporizaciones inteligentes introducidas por el ussuario
52. long int f_conf_temposint = 0; // Confirmacion de uso de temporizaciones
    inteligentes
53.
54. long int f_intval_DPlavar = 1000; // Valor de presion en milibares que
    desencadena el lavado
55. long int f_intval_estabilizado_DPlavar = 2; // Valor en segundos durante los que
    la presion debe no ser inferior al máximo de presion definido para que se ejecute
    el lavado (para desactivarlo ponerlo a 0)
56. long int f_conf_DPlavar = 0; // Activacion del modo por DP
57. long int f_intval_tempolavar = 0; // Valor temporización para activar lavado
58. long int f_conf_tempolavar = 0; // Activacion del modo temporizado
59. long int f_conf_marchalavar = 0; // Activacion manual del lavado
60. long int f_conf_parolavar = 0; // Paro manual del lavado
61.
62. long int f_intval_alta_presion = 1500; // DP en milibares mínima para que salte
    la alarma por baja presión
63. long int f_intval_estabilizado_alta_presion = 3; // Valor en seguntos durante los
    que la presion debe no ser inferior al minimo de presion definido para que se
    active la alarma
64. long int f_conf_alta_presion = 0; // Habilitacion de la alarma por baja presion
65.
66. long int f_intval_baja_presion = 350; // DP en milibares máxima para que salte la
    alarma por no haber presión
67. long int f_intval_estabilizado_baja_presion = 3; // Valor en seguntos durante los
    que la presion debe no ser superior al maximo de presion definido para que se
    active la alarma
68. long int f_conf_baja_presion = 0; // Habilitacion de la alarma por poca presion
69.
70. long int f_intval_limpiezas_bucle_infinito = 4; // Numero de lavados consecutivos
    que deben realizarse en modo DP para que salte la alarma
71. long int f_intval_tempo_bucle_infinito_entrar = 10; // Segundos que deben pasar
    para que la limpieza se considere consecutiva con la anterior. (Debe estsar
    activado el modo DP)
72. long int f_intval_bucle_infinito_salir_estable = 4; // Segundos que deben pasar
    con la presion por debajo del umbral para lavar para que se pueda salir del bucle
    infinito
73. long int f_conf_bucle_infinito = 0; // Habilitación de la alarma por bucle
    infinito
74.
75. // ::: Variables para la programación del menú
76. long int posicion_cifra_f_intval = 1000; // Utilizaremos números de 4 cifras,
    empezamos por las unidades de millar. Misma variable para todas las funciones
    tipo "introducir valor"
77. char valores_f_intval[16] = "     Actual:    ";
78.
79. int valor_modificado = 0; // Variable que sólo se activará cuando se guarde un
    valor en una de las dos funciones (pulsando el boton derecho)
80.
81. long int nivel_menu_principal = 0;
82. long int indice_pantalla_principal = 0;
83. long int nivel_menu_anidado = 100000; // nivel de menú utilizado para navegar por
    la struct de un menú anidado
84. long int indice_pantalla_anidado = 100000; // indice de pantalla utilizado para
    navegar por la struct de un menú anidado
85.
86. long int *nivel_menu_actual = &nivel_menu_principal;
87. long int *indice_pantalla_actual = &indice_pantalla_principal;
88. long int nivel_menu_nuevo;
89. long int indice_pantalla_nuevo;
90.
91. int pantalla_inicio = 1; // 1: Estamos en la pantalla de inicio ; 0: No estamos
    en la pantalla de inicio
92.
93. int tipo_funcion = 0; // Sirve para saber si estamos en una funcion_confirmar,
    funcion introducirvalor, etc...
94.
95. // ::: La dimensión es el número de pantallas que tiene la struct correspondiente
96. long int dimension_menu_principal = 52;
97. long int dimension_menu_alarmas = 1;
98. long int dimension_menu_historico = 1;
99. long int dimension_menu_temposbas = 2; // Le pongo por defecto como si tuviese un
    solo filtro
100.        long int dimension_menu_temposint = 2;
101.        long int *dimension_menu = &dimension_menu_principal;
```

```
102.
103.        // ::: Variables de historicos y alarmas
104.        int bucle_infinito = 0; // 0: No estamos en modo bucle infinito; 1: Sí lo
    estamos.
105.        int bucle_infinito_anterior = 0; // Para detección de flanco
106.        int tempo_bucle_infinito = 15; // segundos de separación entre un lavado
    y el siguiente cuando estamos en modo bucle infinito
107.        int alta_presion = 0; // Hay problema de baja presion
108.        int alta presion anterior = 0; // Para detección de flanco
109.        int baja_presion = 0; // No hay presion en el filtro (puede ser que haya
    petado la tubería aguas arriba, o que no esté pasando agua)
110.        int baja presion anterior = 0; // Para detección de flanco
111.        int alarmas_vacio = 1;
112.        int tipo_alarma = 0;
113.
114.        int historico_vacio = 1;
115.
116.        // ::: Variables varias
117.
118.        long int valor_timer = 32767;
119.
120.        long int lectura analogica = 0; // Variable donde guardamos la lectura de
    la analógica
121.        long int lectura presion = 0; // Lectura analogica convertida a bares
122.
123.        int limpiezas consecutivas = 0; // Aquí se cuentan las limpiezas
    consecutivas
124.
125.        // PRUEBAS
126.        long int crear_alarma_anterior = 0;
127.        //------------------------------------------------------------
128.        //      Menús
129.        //------------------------------------------------------------
130.
131.        struct menu {
132.            long int indice pantalla struct;
133.            char *texto_pantalla_struct;
134.        };
135.
136.        struct menu menu_principal [] = {
137.            {100000, "Configuracion"},
138.                {110000, "Sistema"},
139.                    {111000, "Num. de filtros del equipo"},
140.                        {111100, "f intval num filtros"},
141.                {120000, "Modo de lavado"},
142.                    {121000, "Basico"},
143.                        {121100, "Definir temporizaciones"},
144.                            {121110, "menu anidado temposbas"},
145.                        {121200, "Activar modo"},
146.                            {121210, "f_conf_temposbas"},
147.                    {122000, "Inteligente"},
148.                        {122100, "Definir coeficientes"},
149.                            {122110, "menu anidado temposint"},
150.                        {122200, "Activar modo"},
151.                            {122210, "f_conf_temposint"},
152.                {130000, "Modo activacion lavado"},
153.                    {131000, "Limite DP"},
154.                        {131100, "f intval DPlavar"},
155.                            {131110, "f_intval_estabilizado_DPlavar"},
156.                                {131111, "f conf DPlavar"},
157.                    {132000, "Temporizado"},
158.                        {132100, "f intval tempolavar"},
159.                            {132110, "f conf tempolavar"},
160.                    {133000, "Manual"},
161.                        {133100, "Marcha manual"},
162.                            {133110, "f conf marchalavar"},
163.                        {133200, "Paro manual"},
164.                            {133210, "f conf parolavar"},
165.            {200000, "Seguridad"},
166.                {210000, "Alarma alta     presion"},
167.                    {211000, "f_intval_alta_presion"},
168.                        {211100, "f intval estabilizado alta presion"},
169.                            {211110, "f conf alta presion"},
170.                {220000, "Alarma baja     presion"},
171.                    {221000, "f intval baja presion"},
172.                        {221100, "f_intval_estabilizado_baja_presion"},
```

```
                            {221110, "f_conf_baja_presion"},
                {230000, "Alarma bucle    infinito"},
                    {231000, "f intval limpiezas bucle infinito"},
                        {231100, "f_intval_tempo_bucle_infinito_entrar"},
                            {231110, "f_intval_bucle_infinito_salir_estable"},
                                {231111, "f_conf_bucle_infinito"},
                {240000, "Registro de     alarmas"},
                    {241000, "Mostrar alarmas"},
                        {241100, "menu_anidado_alarmas"},
                    {242000, "Borrar alarmas"},
                        {242100, "f_conf_borrar_alarmas"},
                {250000, "Registro de     lavados"},
                    {251000, "Mostrar         historico"},
                        {251100, "menu_anidado_lavados"},
                    {252000, "Borrar historico"},
                        {252100, "f_conf_borrar_historico"}
        };

        struct menu menu alarmas [] = {
            {100000, "Ninguna Alarma"},
            {200000, "A2"},
            {300000, "A3"},
            {400000, "A4"},
            {500000, "A1"},
            {600000, "A2"},
            {700000, "A3"},
            {800000, "A4"},
            {900000, "A1"}
        };

        struct menu menu historico [] = {
            {100000, "Ningun registro de lavado"},
            {200000, "R2"},
            {300000, "R3"},
            {400000, "R4"},
            {500000, "R1"},
            {600000, "R2"},
            {700000, "R3"},
            {800000, "R4"},
            {900000, "R1"}
        };

        struct menu menu temporizaciones_basico [] = {
            {100000, "tpo(s) lavar E1"},
            {110000, "f_intval_baslav1"},
            {200000, "tpo(s) reposo E1"},
            {210000, "f_intval_basrep1"},
            {300000, "tpo(s) lavar E2"},
            {310000, "f_intval_baslav2"},
            {400000, "tpo(s) reposo E2"},
            {410000, "f intval basrep2"},
            {500000, "tpo(s) lavar E3"},
            {510000, "f intval baslav3"},
            {600000, "tpo(s) reposo E3"},
            {610000, "f_intval_basrep3"},
            {700000, "tpo(s) lavar E4"},
            {710000, "f intval baslav4"},
            {800000, "tpo(s) reposo E4"},
            {810000, "f_intval_basrep4"},
            {900000, "tpo(s) lavar E5"},
            {910000, "f intval baslav5"},
        };

        struct menu menu_temporizaciones_inteligente [] = {
            {100000, "seg/bar lavar E1"},
            {110000, "f intval intlav1"},
            {200000, "tpo(s) reposo E1"},
            {210000, "f intval intrep1"},
            {300000, "seg/bar lavar E2"},
            {310000, "f intval intlav2"},
            {400000, "tpo(s) reposo E2"},
            {410000, "f intval intrep2"},
            {500000, "seg/bar lavar E3"},
            {510000, "f_intval_intlav3"},
            {600000, "tpo(s) reposo E3"},
            {610000, "f_intval_intrep3"},
```

```
249.            {700000, "seg/bar lavar E4"},
250.            {710000, "f intval intlav4"},
251.            {800000, "tpo(s) reposo E4"},
252.            {810000, "f_intval_intrep4"},
253.            {900000, "seg/bar lavar E5"},
254.            {910000, "f_intval_intlav5"},
255.        };
256.
257.        struct menu *menu_actual = menu_principal;
258.        struct menu *menu_viejo = menu_principal; // Creado para detectar cuándo
    acabamos de cambiar del menú principal a uno anidado o alrevés
259.
260.        //-------------------------------------------------------------
261.        //      Programa
262.        //-------------------------------------------------------------
263.
264.        void main( void )
265.        {
266.            // Stop watchdog timer to prevent time out reset
267.            WDTCTL = WDTPW + WDTHOLD;
268.
269.        //-------------------------------------------------------------
270.        //      Inicialización MCU
271.        //-------------------------------------------------------------
272.
273.            // I/O Pins
274.
275.            P1SEL = 0x00;
276.            P1DIR = 0x00; // Primero los ponemos todos a input
277.            P1DIR |= (BIT5 + BIT6 + BIT7); // P1.5 P1.6 P1.7 output para el
    control del LCD
278.            P1OUT = 0x00; //COMPROBAR!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!SI AFECTA EN LOS PINS INPUT DE INTERRUPTS (No afecta)
279.
280.            P2SEL = 0x00;
281.            P2DIR = 0xFF; // P2 output for LCD data
282.            P2OUT = 0x00;
283.
284.            P3SEL = 0x00;
285.            P3DIR = 0xFF;
286.            P3OUT = 0x00;
287.
288.            P4SEL = 0x00;
289.            P4DIR = 0xFF;
290.            P4OUT = 0x00;
291.
292.            // Timer
293.
294.            TACTL = 0x0100 | MC0; // Source: ACLK ; Mode control en up mode (el
    timer cuenta hasta TACCR0)
295.            TACCR0 = valor timer; // De este modo, cada overflow equivaldrá a un
    segundo (reloj de 32768 Hz)
296.            TACCTL0 |= CCIE; // Este bit habilita la interrupción procedente del
    CCIFG flag correspondiente.
297.            TACCTL1 = 0x0000;
298.            TACCTL2 = 0x0000;
299.
300.            // Analogica
301.
302.            P6SEL = 0x0004; // Seleccionamos la funcion de canal analógico para
    el pin 2
303.
304.            ADC12CTL0 = 0x0F10;//ADC12ON , 1024 ADC12CLK cycles for sampling
    period
305.            ADC12CTL1 = 0x00E8; // ADC12SSEL: ACLK, DIV /8
306.            ADC12MCTL0 = 0x0002; //INCH2 input channel A2
307.            ADC12IE = 0x0001; // Enable interrupcion del ADC12MEM0
308.
309.            // Pins no utilizados - Configuración para reducción del consumo
    energético
310.
311.            P5SEL = 0x00;
312.
313.            P5DIR = 0xFF;
314.            P6DIR = 0xFF;
315.
```

```
316.            P1OUT = 0x00;
317.            P2OUT = 0x00;
318.            P5OUT = 0x00;
319.            P6OUT = 0x00;
320.
321.        //-------------------------------------------------------------
322.        //      Inicialización LCD
323.        //-------------------------------------------------------------
324.
325.            P1OUT&=~BIT5;  //RS=0
326.            P1OUT&=~BIT6; //RW=0
327.            P1OUT|=BIT7; //EN=1
328.            delay(10000); //some LCDs take time to initialize at startup
329.            delay(10000);
330.            delay(10000);
331.            delay(10000);
332.            delay(10000);
333.
334.            s inst lcd(); //Call Instruction Select routine
335.            P2OUT=0X38; //Function set (4 bits interface, 2 line, 5x8 puntos)
336.            s latch lcd(); //Lanzar la instrucción 3 veces para asegurarnos de
    estar en modo 8 bits
337.            s_latch_lcd();
338.            s latch lcd();
339.            P2OUT=0X28; // Function set (4 bits interface, 2 line,  5x8 dots)
340.            s latch lcd(); // Dado que nos comunicamos con 4 pines, los 4 ultimos
    bits se han transmitido como 0, por lo que los volvemos a definir a continuacion
341.            P2OUT=0x80; // (2o nibble de 4 bits - (el menos significativo))
    Definimos N = 1 (2 lineas) y F = 0 (5x8 puntos)
342.            // Ahora estamos ya en modo 4 bits
343.            P2OUT=0X0F;// Send Second Instruction on Port. -> Display ON/OFF
344.            s latch lcd();
345.            P2OUT=0XF0;
346.            s latch lcd(); // Second nibble
347.            P2OUT=0x01; // Clear and cursor at home
348.            s latch lcd();
349.            P2OUT=0X10;
350.            s_latch_lcd(); // Second nibble
351.            /*P2OUT=0X10; // Set cursor on line 1. Use 0xC0 to write on line 2. -
    > Cursor/display shift (0x10: move cursor, shift left)(0x14: move cursor, shift
    right)(0x18: shift display, shift right)
352.            s_latch_lcd();  */
353.            P2OUT = 0x06; //Entry mode set (0x04:)
354.            s latch lcd();
355.            P2OUT=0x60;
356.            s latch lcd(); // Second nibble
357.            P2OUT = 0x0C; // Cursor off and blink off
358.            s latch lcd();
359.            P2OUT = 0xC0;
360.            s_latch_lcd();
361.            P2OUT=0X00; //Before changing to write data select. Make port=0x0 so
    that LCD receives nothing
362.            s data lcd(); //Change the input type to Data.(before it was
    instruction input)
363.
364.            // PREPARADO PARA ESCRIBIR (LINEA 1: ----; 0x01 (Clear and cursor at
    home) LINEA 2: 0xC0)
365.
366.        //-------------------------------------------------------------
367.        //      Interrupciones
368.        //-------------------------------------------------------------
369.
370.              enable interrupt(); // GIE = 1
    COMPROBAR!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! SI NECESARIO GIE=0 AL INICIO
    DEL PROGRAMA
371.            P1IFG = 0x00; // Por si acaso hay algún flag
372.            P1IES = 0x1B; // Flanco desdendente (configuración pull up en los
    botones)
373.            P1IE = 0x1B; // Habilitación de las interrupciones (sin aceptar el
    botón del medio)
374.
375.            ADC12CTL0 |= 0x0003; //ENC + ADC12SC;
376.            ADC12CTL0 &= 0xFFFE; // Apagamos ADC12SC
377.
378.        //-------------------------------------------------------------
379.        //      MCU esperando interrupciones
```

```
380.       //-------------------------------------------------------------
381.
382.           while(1)
383.           {
384.               LPM3;
385.           }
386.       }
387.
388.       //-------------------------------------------------------------
389.       //      INTERRUPCIONES
390.       //-------------------------------------------------------------
391.
392.       #pragma vector = ADC12_VECTOR
393.       __interrupt void adc12_interrupt(void)
394.       {
395.         lectura_analogica = ADC12MEM0; // Leemos el valor convertido. Cuando se
    hace la lectura se apaga el flag
396.         lectura_presion = (2000 * lectura_analogica) / 4095; // Conversion a
    milibares (3 bares máximo)
397.
398.        // Podemos estar en 2 situaciones, en bucle infinito o fuera de él.
    Estemos donde estemos comprobamos que no haya baja presión
399.         if(lectura_presion < f_intval_baja_presion) // Si hay problema en la
    tuberia aguas arriba (o está apagado el sistema)
400.         {
401.           TACCTL0 |= CCIE;
402.           tempo estable DPlavar = 0;
403.           tempo_estable_alta_presion = 0;
404.           tempo estable salir bucle = 0;
405.
406.           if(tempo estable baja presion >= f intval estabilizado baja presion)
407.           {
408.             baja_presion = 1; // Hay problema de poca presión
409.             tempo estable baja presion = 0;
410.             limpieza = 0;
411.             P3OUT &= 0x1F; // Apagado de las salidas
412.             P4OUT &= 0xFC;
413.           }
414.         }
415.         else
416.         {
417.           tempo_estable_baja_presion = 0;
418.           if(lectura presion > f intval alta presion) // Si se lee una presion
    superior a la definida para el problema de alta presion
419.           {
420.            TACCTL0 |= CCIE;
421.            tempo_estable_DPlavar = 0;
422.            tempo estable salir bucle = 0;
423.             if(tempo_estable_alta_presion >= f_intval_estabilizado_alta_presion
    )
424.             {
425.               alta_presion = 1; // Hay problema de mucha presión
426.               tempo estable alta presion = 0;
427.               limpieza = 0;
428.               P3OUT &= 0x1F; // Apagado de las salidas
429.               P4OUT &= 0xFC;
430.             }
431.           }
432.           else
433.           {
434.             tempo estable alta presion = 0; // La presion ha bajado de la
    definida para baja presion
435.
436.             if((bucle_infinito == 1) && (f_conf_DPlavar == 1)) // Hemos entrado
    en modo bucle infinito
437.             {
438.               if(lectura presion  > f intval DPlavar)
439.               {
440.                 tempo_estable_salir_bucle = 0;
441.               }
442.               else
443.               {
444.                 if((tempo estable salir bucle > f intval bucle infinito salir e
    stable) && (limpieza == 0)) // Si se está limpiando no se apagará el bucle
445.                 {
446.                   bucle_infinito = 0; // Salimos del modo bucle infinito
```

```
447.                   tempo_estable_salir_bucle = 0; // Reiniciamos el valor para
    la siguiente vez
448.                 }
449.               }
450.             }
451.             else // Modo normal (fuera del bucle)
452.             {
453.               if((f_conf_DPlavar == 1) && (baja_presion == 0) && (alta_presion
    == 0))
454.               {
455.                 if(lectura_presion  > f_intval_DPlavar)
456.                 {
457.                   TACCTL0 |= CCIE; // Habilitamos la interrupt del timer para
    contar la estabilización. Sirve para el de baja presion tambien
458.                   if(tempo_estable_DPlavar > f_intval_estabilizado_DPlavar)
459.                   {
460.                     limpieza = 1;
461.                     tempo_estable_DPlavar = 0;
462.                     if(tempo limpieza < f intval tempo bucle infinito entrar) /
    / Si no ha pasado bastante tiempo, lo consideramos como limpieza consecutivva
463.                     {
464.                       limpiezas consecutivas += 1;
465.                       if(limpiezas_consecutivas >= f_intval_limpiezas_bucle_inf
    inito) // Comparamos con el valor límite de limpiezas consecutivas
466.                       {
467.                         bucle infinito = 1; // Activamos el modo
468.                         TACCTL0 |= CCIE; // Habilitamos el timer
469.                         limpiezas consecutivas = 0;
470.                       }
471.                     }
472.                   }
473.                 }
474.                 else
475.                 {
476.                   tempo_estable_DPlavar = 0;
477.                 }
478.               }
479.             }
480.           }
481.         }
482.
483.       // Si estamos en modo DP, no se esta lavando y no estamos en modo bucle
    infinito (ya que aquí, todo funciona por timer), mirar a ver cuándo puedo apagar
    la interrupt
484.       // Si están los dos modos apagados, deshabilito la interrupción del timer
    para ahorrar
485.       if ((lectura_presion > f_intval_baja_presion) && (lectura_presion < f_int
    val alta presion) && (f conf tempolavar == 0) && (limpieza == 0) && (tempo limpie
    za > f_intval_tempo_bucle_infinito_entrar)) // Ya sabemos que no será limpieza
    cosecutiva. Evidentemente, si se esta comprobando la estabilidad de la presion no
    deshabilitamos timer
486.         {
487.           if (f_conf_DPlavar == 0) // Si están los dos modos apagados
488.           {
489.             TACCTL0 &= ~BIT4; // Deshabilitamos las interrupts del timer
490.           }
491.           else
492.           {
493.             if ((lectura_presion < f_intval_DPlavar) && bucle_infinito == 0)
494.             {
495.               TACCTL0 &= ~BIT4; // Deshabilitamos las interrupts del timer
496.             }
497.           }
498.         }
499.
500.       // La siguiente estructura condicional es únicamente para encender el led
    rojo que indica si tengo la interrupcion del timer habilitada.
501.       // Esto me sirve para depurar el programa y lo pongo aquí ya que la
    interrupt analogica está siempre activa
502.       if ((TACCTL0 & 0x0010) >= 1)
503.       {
504.         P4OUT |= BIT7;
505.       }
506.       else
507.       {
508.         P4OUT &= ~BIT7;
```

```
509.       }
510.
511.       //Aqui hago las comprobaciones de alarmas
512.       if ((baja_presion + alta_presion + bucle_infinito) > (baja_presion_anteri
   or + alta_presion_anterior + bucle_infinito_anterior))
513.       {
514.         if(baja presion > baja presion anterior)
515.         {
516.           tipo_alarma = 1;
517.           baja_presion_anterior = 1;
518.         }
519.         else
520.         {
521.           if(alta_presion > alta_presion_anterior)
522.           {
523.             tipo_alarma = 2;
524.             alta presion anterior = 1;
525.           }
526.           else
527.           {
528.             if(bucle_infinito > bucle_infinito_anterior)
529.             {
530.               tipo_alarma = 3;
531.               bucle_infinito_anterior = 1;
532.             }
533.           }
534.         }
535.
536.         if(alarmas_vacio == 1)
537.         {
538.           dimension_menu_alarmas = 0;
539.           alarmas_vacio = 0; // Si entramos aqui es que estamos registrando una
   alarma
540.         }
541.         else
542.         {
543.           if(dimension_menu_alarmas == 9) // Hemos llegado al tamaño máximo,
   así que lo limitamos. Aprovechamos también que las operaciones a realizar son las
   mismas que si tuviesemos 8 alarmas registradas.
544.           {
545.             dimension_menu_alarmas = 8;
546.           }
547.           for (int i = dimension_menu_alarmas; i>0; i--) // Bajamos todas las
   alarmas una linea
548.           {
549.             menu_alarmas[i].texto_pantalla_struct = menu_alarmas[i -
    1].texto_pantalla_struct;
550.           }
551.         }
552.         switch (tipo_alarma) // Escribimos en la primera linea la alarma
   correspondiente
553.         {
554.         case 1:
555.            menu_alarmas[0].texto_pantalla_struct = "Detección Baja  Presion";
556.            break;
557.         case 2:
558.            menu_alarmas[0].texto_pantalla_struct = "Detección Alta  Presion";
559.            break;
560.         case 3:
561.            menu_alarmas[0].texto_pantalla_struct = "Activación BucleInfinito";
562.            break;
563.         }
564.         dimension_menu_alarmas += 1; // Actualizamos el valor de la dimension
565.       }
566.
567.
568.       //Aqui actualizo los registros del histórico de lavados
569.       if ((baja_presion + alta_presion + bucle_infinito) > (baja_presion_anteri
   or + alta_presion_anterior + bucle_infinito_anterior))
570.       {
571.         if(baja_presion > baja_presion_anterior)
572.         {
573.           tipo_alarma = 1;
574.           baja_presion_anterior = 1;
575.         }
576.         else
```

```
577.          {
578.            if(alta presion > alta presion anterior)
579.            {
580.              tipo_alarma = 2;
581.              alta_presion_anterior = 1;
582.            }
583.            else
584.            {
585.              if(bucle_infinito > bucle_infinito_anterior)
586.              {
587.                tipo_alarma = 3;
588.                bucle infinito anterior = 1;
589.              }
590.            }
591.          }
592.
593.          if(alarmas_vacio == 1)
594.          {
595.            dimension_menu_alarmas = 0;
596.            alarmas_vacio = 0; // Si entramos aqui es que estamos registrando una
    alarma
597.          }
598.          else
599.          {
600.            if(dimension_menu_alarmas == 9) // Hemos llegado al tamaño máximo,
    así que lo limitamos. Aprovechamos también que las operaciones a realizar son las
    mismas que si tuviesemos 8 alarmas registradas.
601.            {
602.              dimension_menu_alarmas = 8;
603.            }
604.            for (int i = dimension_menu_alarmas; i>0; i--) // Bajamos todas las
    alarmas una linea
605.            {
606.              menu_alarmas[i].texto_pantalla_struct = menu_alarmas[i -
     1].texto_pantalla_struct;
607.            }
608.          }
609.          switch (tipo_alarma) // Escribimos en la primera linea la alarma
    correspondiente
610.          {
611.          case 1:
612.             menu_alarmas[0].texto_pantalla_struct = "Detección Baja  Presion";
613.             break;
614.          case 2:
615.             menu_alarmas[0].texto_pantalla_struct = "Detección Alta  Presion";
616.             break;
617.          case 3:
618.             menu_alarmas[0].texto_pantalla_struct = "Activación BucleInfinito";
619.             break;
620.          }
621.          dimension_menu_alarmas += 1; // Actualizamos el valor de la dimension
622.        }
623.
624.          // Estabilización
625.          // Activacion lavado
626.          // Deteccion alarma
627.          // Mirar tempo a ver si apago el timer
628.          // Mirar si estamos en un rango normal para salir del bucle infinito de
    lavados
629.          // Registro del lavado
630.
631.          ADC12CTL0 |= 0x0001; // ADC12SC
632.          ADC12CTL0 &= 0xFFFE; // Apagamos ADC12SC para que se empiece otra
    conversión
633.        }
634.
635.        // TIMER
636.        #pragma vector=TIMERA0_VECTOR
637.        __interrupt void Timer_A(void)
638.        {
639.          // ::: COMPROBACIONES Y ACCIONES SOBRE LAS LECTURAS ANALOGICAS
640.
641.          //Sumamos tempos de estabilización donde corresponda
642.          if (lectura_presion > f_intval_alta_presion)
643.          {
644.            tempo_estable_alta_presion += 1;
```

```
645.          }
646.          if(lectura presion < f intval baja presion)
647.          {
648.            tempo_estable_baja_presion +=1;
649.          }
650.          else
651.          {
652.            if((bucle_infinito == 1) && (lectura_presion < f_intval_DPlavar)) //
    Contamos estabilizacion para salir del modo bucle infinito
653.            {
654.              tempo_estable_salir_bucle += 1;
655.            }
656.            else
657.            {
658.              if((lectura_presion > f_intval_DPlavar) && (f_conf_DPlavar == 1) &&
     (limpieza == 0)) // Contamos estabilizacion para lavar (sólo si esta activado el
    modo)
659.              {
660.                tempo estable DPlavar += 1;
661.              }
662.            }
663.          }
664.
665.          //::: EJECUCION DE LA LIMPIEZA
666.
667.          if ((limpieza == 1) && (((f intval num filtros > 0) && (limpieza anteri
    or == 0)) || ((num_filtros_def > 0) && (limpieza_anterior == 1))))
668.          {
669.            if (limpieza anterior == 0) // Deteccion de comienzo de lavado
670.            {
671.              num filtros def = f intval num filtros; // Copiamos la variable del
    numero de filtros para preservar su valor
672.              tempo limpieza = 0;
673.              indice limpieza = 0;
674.              if ((f_conf_temposbas == 1) || (f_conf_DPlavar == 0)) //
    Identificamos el modo de lavado seleccionado por el usuario (básico o
    inteligente). Además, si no está axctivado el modo por DP, no tiene sentido
    utilizar la tabla de tempos inteligentes.
675.              {
676.                for(int i = 0; i<9; i++) // Aquí copiamos los valores para que no
    se puedan modificar durante el lavado
677.                {
678.                  temposdef [i] = temposbas [i] * (32768 / (valor timer + 1)); //
    Obtenemos el número de contajes a efectuar para los segundos especificados por el
    usuario.
679.                                                                                //
    Al ser "int" el resultado será entero y no habrá problema. el +1 es por el
    contaje a 0.
680.                                                                                //
    Aqui se copian todos los valores, luego ya se comprueba cuántas filtros ha
    definido el usuario.
681.                }
682.              }
683.              else // Está activado temposint (porque siempre debe estar activo
    almenos uno)
684.              {
685.                  for(int i = 0; i<9; i++)
686.                {
687.                  temposdef [i] = f_intval_DPlavar * (temposint [i] * (32768 / (v
    alor timer + 1))); // Multiplicamos por la presion a partir de la cual se limpia,
    ya que el valor que teniamos era el coeficiente (s/bar)
688.                }
689.              }
690.            }
691.            else
692.            {
693.              tempo limpieza += 1; // Aprovecho que diferencio entre primera vez
    y las posteriores para no sumar la primera vez, ya que acabamos de entrar.
694.            }
695.
696.            if ((tempo_limpieza >= temposdef[indice_limpieza]) || (limpieza_anter
    ior == 0))
697.            {
698.              if (limpieza_anterior == 1) // En caso de que se deba cambiar de
    estación, y no se trate de el principio de la secuencia
699.              {
```

```
700.               indice_limpieza += 1;
701.             }
702.             tempo_limpieza = 0;
703.             while (temposdef[indice_limpieza] < 1) // Si el tiempo es 0
    segundos, pasamos directamente al siguiente valor
704.             {
705.               indice_limpieza += 1;
706.             }
707.             if ((indice_limpieza + 1) > ((f_intval_num_filtros * 2) - 1)) //
    Entramos aquí cuando se acabe la secuencia. La variable indice_limpieza empieza
    en 0 y recorre tanto los tiempos de limpieza como los de reposo.
708.             {
709.               indice_limpieza = 0; // Reiniciamos el indice
710.               tempo_limpieza = 0; // Reiniciamos así podemos empezar a contar
    el tiempo desde el último lavado
711.               limpieza = 0; // Salimos de la limpieza
712.               P3OUT &= 0x1F; // Apagamos las salidas
713.               P4OUT &= 0xFC; // Apagamos las salidas
714.               //Mientras se lava, la única estabilización que se puede contar
    es la de problema por baja presion
715.               if(f_conf_DPlavar == 1)
716.               {
717.                 tempo_estable_salir_bucle = 0;
718.                 tempo_estable_DPlavar = 0;
719.               }
720.             }
721.             else
722.             {
723.                 switch (indice_limpieza)
724.           {
725.           case 0:
726.             P3OUT &= 0x1F; // Primero apagamos las demás salidas
727.             P3OUT |= 0x20; // Luego encendemos la que corresponde
728.             P4OUT &= 0xFC;
729.             break;
730.           case 1:
731.             P3OUT &= 0x1F;
732.             P4OUT &= 0xFC;
733.             break;
734.           case 2:
735.             P3OUT &= 0x1F;
736.             P3OUT |= 0x40;
737.             P4OUT &= 0xFC;
738.             break;
739.           case 3:
740.             P3OUT &= 0x1F;
741.             P4OUT &= 0xFC;
742.             break;
743.           case 4:
744.             P3OUT &= 0x1F;
745.             P3OUT |= 0x80;
746.             P4OUT &= 0xFC;
747.             break;
748.           case 5:
749.             P3OUT &= 0x1F;
750.             P4OUT &= 0xFC;
751.             break;
752.           case 6:
753.             P3OUT &= 0x1F;
754.             P4OUT &= 0xFC;
755.             P4OUT |= 0x01;
756.             break;
757.           case 7:
758.             P3OUT &= 0x1F;
759.             P4OUT &= 0xFC;
760.             break;
761.           case 8:
762.             P3OUT &= 0x1F;
763.             P4OUT &= 0xFC;
764.             P4OUT |= 0x02;
765.             break;
766.           }
767.             }
768.           }
769.
```

```
770.          // Al principio tempo_limpieza vale 0. estacion_actual tambien vale 0.
    Incrementar hasta llegar al tiempo del filtro, entonces sumar 1 a estacion actual
771.          // y reiniciar el tempo limpieza (filtro actual servira tambien como
    indice para el array del tempo a ejecutar). Asi hasta llegar al último filtro.
772.          // IF al final para saber si ya se ha terminado y así volverlo a poner
    todo "reiniciado y preparado".
773.        }
774.        else
775.        {
776.          limpieza = 0; // Si no es posible realizar la limpieza por que no se
    han definido filtros o hay baja presion por ejemplo, se borra el lavado para que
    tenga que volver a activarse si se desea
777.
778.          tempo_limpieza += 1; // Cuando no limpiamos, contamos el tiempo desde
    el último lavado
779.        }
780.
781.        // Actualizamos el valor de la limpieza anterior
782.        limpieza_anterior = limpieza; // Así puedo distinguir si es el comienzo
    de la limpieza o no. Lo pongo aquí fuera porque cuando paro la limpieza
    manualmente, se debe ejecutar tambien (antes estaba dentro de la secuencia y no
    se ejecutaba al pararla.
783.
784.        // Código para la activación temporizada del lavado
785.        if (bucle_infinito == 1)
786.        {
787.          if((limpieza == 0) && (tempo_limpieza >= tempo_bucle_infinito) && (baja
     presion == 0) && (alta presion == 0))
788.          {
789.            limpieza = 1;
790.          }
791.        }
792.        else
793.        {
794.          if ((f_conf_tempolavar == 1) && (limpieza == 0) && (tempo_limpieza >= f
     intval_tempolavar) && (baja presion == 0) && (alta presion == 0))
795.          {
796.            limpieza = 1;
797.          }
798.        }
799.        }
800.
801.        // BOTONES
802.        #pragma vector=PORT1_VECTOR
803.        __interrupt void Port_1(void)
804.        {
805.            P1IE = 0x00; // Deshabilitar interrupciones
806.
807.            delay(100); // Para filtrar pulsos cortos como los originados por los
    rebotes
808.
809.            // En primer lugar nos aseguramos de que haya habido pulsación (no es
    un rebote) e identificamos el botón, haciendo las operaciones asociadas
810.
811.            if ((P1IN & 0x1B) < 0x1B) // Todavía hay algún botón pulsado (señal
    LOW)
812.            {
813.
814.        //-------------------------------------------------------------
815.        //      TRATAMIENTO DE LOS BOTONES
816.        //-------------------------------------------------------------
817.
818.              //*****************************************************************
    *******
819.              // Tratamiento de los botones en caso de estar navegando en las
    pantallas
820.
821.                if (tipo_funcion == 0) // Aquí se indica el tratamiento para cada
    botón si no estamos en ninguna "función"
822.                {
823.                    if (pantalla_inicio == 1) // Nivel 0 es la pantalla general.
    Si se cumple esto es porque estamos en el menú principal y no en ningún menú
    anidado
824.                    {
825.                        indice_pantalla_nuevo = 100000; // Independientemente del
    botón entraremos en el menú
```

```
826.                        nivel_menu_nuevo = 100000;
827.                        pantalla_inicio = 0; // Reseteamos
828.                    }
829.
830.                    else
831.                    {
832.                        if ((P1IFG & BIT0) > 0) // p1.0 (boton arriba)  tiene el
    flag activo
833.                        {
834.                            if (((*indice_pantalla_actual / *nivel_menu_actual) %
     10) != 1) // Nos aseguramos de que no estamos en el primer elemento del submenu
835.                            {
836.                                indice_pantalla_nuevo = *indice_pantalla_actual -
     *nivel_menu_actual;
837.                                nivel_menu_nuevo = *nivel_menu_actual;
838.                            }
839.                        }
840.                        if ((P1IFG & BIT1) > 0) // p1.1 (boton izquierda)  tiene
    el flag activo
841.                        {
842.                          if (*nivel_menu_actual == 100000) // Si estoy en el
    primer nivel, me salgo (Cambiar valor si cambio el número de niveles de menú)
843.                            {
844.                              if (menu_actual == menu_principal) // Si estamos en
    el primer menú del menú principal
845.                              {
846.                                pantalla_inicio = 1;
847.                              }
848.                              else // Si estamos en el primer menú de un menú
    anidado
849.                              {
850.                                menu_actual = menu_principal;
851.                                indice_pantalla_actual = &indice_pantalla_principal
    ;
852.                                nivel_menu_actual = &nivel_menu_principal;
853.                                dimension_menu = &dimension_menu_principal;
854.
855.                                nivel_menu_nuevo = *nivel_menu_actual;
856.                                indice_pantalla_nuevo = *indice_pantalla_actual;
857.                              }
858.                            }
859.                            else
860.                            {
861.                              nivel_menu_nuevo = *nivel_menu_actual * 10; //
    Actualizamos el nivel del menu
862.                              // ponemos a 0 la cifra correspondiente al nivel del
    menu (al dividir se pierden los decimales)
863.                              indice_pantalla_nuevo = *indice_pantalla_actual / niv
    el_menu_nuevo;
864.                              indice_pantalla_nuevo = indice_pantalla_nuevo * nivel
    _menu_nuevo;
865.                            }
866.                        }
867.                        if ((P1IFG & BIT3) > 0) // p1.3 (boton derecha)  tiene el
    flag activo
868.                        {
869.                            if ((*indice_pantalla_actual % 10) == 0) // Miramos
    que no sea el final de un menú (que aún queden "0" a la derecha)
870.                            {
871.                                nivel_menu_nuevo = *nivel_menu_actual / 10; //
    Actualizamos el nivel del menu
872.                                indice_pantalla_nuevo = *indice_pantalla_actual +
     nivel_menu_nuevo;
873.                            }
874.                        }
875.                        if ((P1IFG & BIT4) > 0) // p1.4 (boton abajo)  tiene el
    flag activo
876.                        {
877.                            if (((*indice_pantalla_actual / *nivel_menu_actual) %
     10) != 9) // Nos aseguramos de que no estamos en el último elemento del submenu
878.                            {
879.                              indice_pantalla_nuevo = *indice_pantalla_actual + *
    nivel_menu_actual;
880.                            nivel_menu_nuevo = *nivel_menu_actual;
881.                            }
882.                        }
```

```
883.                     }
884.                 }
885.
886.
887.
888.
889.                 //*************************************************************
    *********
890.                 else // Tratamiento de los botones en caso de estar en una
    función
891.                 {
892.
893.
894.                   // Aquí va el tratamiento de los botones en caso de estar en la
    función 1
895.                     if (tipo_funcion == 1) // funcion_confirmar
896.                     {
897.                         if ((P1IFG & BIT0) > 0) // p1.0 (boton arriba) - Toggle
    "variable funcion copia"
898.                         {
899.                             if(variable_funcion_copia == 1)
900.                                 variable_funcion_copia = 0;
901.                             else
902.                             {
903.                                 if (variable_funcion_copia == 0)
904.                                     variable_funcion_copia = 1;
905.                             }
906.                         }
907.                         if ((P1IFG & BIT1) > 0) // p1.1 (boton izquierda) - Salir
    hacia atrás
908.                         {
909.                           if (*nivel_menu_actual == 100000) // Si estoy en el
    primer nivel de menu, me salgo (Cambiar si cambio el número de niveles de menú)
910.                           {
911.                             if (menu_actual == menu_principal) // Si estamos en
    el primer menú del menú principal
912.                             {
913.                               pantalla_inicio = 1;
914.                             }
915.                             else // Si estamos en el primer menú de un menú
    anidado
916.                             {
917.                               menu_actual = menu_principal;
918.                               indice_pantalla_actual = &indice_pantalla_principal
    ;
919.                               nivel_menu_actual = &nivel_menu_principal;
920.                               dimension_menu = &dimension_menu_principal;
921.
922.                               nivel_menu_nuevo = *nivel_menu_actual;
923.                               indice_pantalla_nuevo = *indice_pantalla_actual;
924.                             }
925.                           }
926.                           else
927.                           {
928.                             nivel_menu_nuevo = *nivel_menu_actual * 10; //
    Actualizamos el nivel del menu
929.                             // ponemos a 0 la cifra correspondiente al nivel del
    menu (al dividir se pierden los decimales)
930.                             indice_pantalla_nuevo = *indice_pantalla_actual / niv
    el_menu_nuevo;
931.                             indice_pantalla_nuevo = indice_pantalla_nuevo * nivel
    _menu_nuevo;
932.                           }
933.                         }
934.                         if ((P1IFG & BIT3) > 0) // p1.3 (boton derecha) -
    Confirmar selección o pasar a la siguiente pantalla
935.                         {
936.                             if (variable_funcion_copia == *variable_funcion) //
    En este caso pasar a la pantalla siguiente, ya que no se ha modificado el valor
    (mismo código que tratamiento para navegación)
937.                             {
938.                                 if ((*indice_pantalla_actual % 10) == 0) //
    Miramos que no sea el final de un menú (que aún queden "0" a la derecha)
939.                                 {
940.                                     nivel_menu_nuevo = *nivel_menu_actual / 10; /
    / Actualizamos el nivel del menu
```

```
941.                                    indice_pantalla_nuevo = *indice_pantalla_actu
    al + nivel_menu_nuevo;
942.                                }
943.                            }
944.                            else
945.                            {
946.                                *variable_funcion = variable_funcion_copia;
947.                                valor_modificado = 1;
948.                            }
949.                        }
950.                        if ((P1IFG & BIT4) > 0) // p1.4 (boton abajo)  tiene el
    flag activo
951.                        {
952.                            if(variable_funcion_copia == 1)
953.                                variable_funcion_copia = 0;
954.                            else
955.                            {
956.                                if (variable_funcion_copia == 0)
957.                                    variable_funcion_copia = 1;
958.                            }
959.                        }
960.                    }
961.
962.
963.
964.
965.                    // Aquí va el tratamiento de los botones en caso de estar en
    la función 2
966.                    else
967.                    {
968.                        if (tipo_funcion == 2) // funcion introducirvalor
969.                        {
970.                            if ((P1IFG & BIT0) > 0) // p1.0 (boton arriba)  tiene
    el flag activo
971.                            {
972.                              if (((variable_funcion_copia / posicion_cifra_f_int
    val) % 10) != 9) // Aquí mirar si ya es 9 para no aumentarlo más. LA división de
    módulo es entera, con lo que se van los decimales
973.                                {
974.                                  variable_funcion_copia = variable_funcion_copia +
     posicion_cifra_f_intval;
975.                                }
976.                            }
977.                            if ((P1IFG & BIT1) > 0) // p1.1 (boton
    izquierda)  tiene el flag activo
978.                            {
979.                              if (posicion_cifra_f_intval < 1000)
980.                              {
981.                                posicion_cifra_f_intval *= 10;
982.                              }
983.                              else
984.                              {
985.                                if (*nivel_menu_actual == 100000) // Si estoy en
    el primer nivel del menú, me salgo (Cambiar si cambio el número de niveles de
    menú)
986.                                {
987.                                  if (menu_actual == menu_principal) // Si
    estamos en el primer nivel de menú del menú principal
988.                                  {
989.                                    pantalla_inicio = 1;
990.                                  }
991.                                  else // Si estamos en el primer nivel de menú
    de un menú anidado
992.                                  {
993.                                    menu_actual = menu_principal;
994.                                    indice_pantalla_actual = &indice_pantalla_pri
    ncipal;
995.                                    nivel_menu_actual = &nivel_menu_principal;
996.                                    dimension_menu = &dimension_menu_principal;
997.
998.                                    nivel_menu_nuevo = *nivel_menu_actual;
999.                                    indice_pantalla_nuevo = *indice_pantalla_actu
    al;
1000.                                 }
1001.                               }
1002.                               else
```

```
1003.                                 {
1004.                                   nivel_menu_nuevo = *nivel_menu_actual * 10; //
    Actualizamos el nivel del menu
1005.                                   // ponemos a 0 la cifra correspondiente al
    nivel del menu (al dividir se pierden los decimales)
1006.                                   indice_pantalla_nuevo = *indice_pantalla_actual
     / nivel menu nuevo;
1007.                                   indice_pantalla_nuevo = indice_pantalla_nuevo *
     nivel_menu_nuevo;
1008.                                 }
1009.                                 // Apagamos el cursor, solo se enciende cuando
    entremos en la funcion 2 (introducir valor)
1010.                                 s_inst_lcd();
1011.                                 P2OUT = 0x0C; // Cursor off and blink off
1012.                                 s_latch_lcd();
1013.                                 P2OUT = 0xC0;
1014.                                 s_latch_lcd();
1015.                                 s_data_lcd();
1016.                               }
1017.                             }
1018.                             if ((P1IFG & BIT3) > 0) // p1.3 (boton
    derecha)  tiene el flag activo
1019.                             {
1020.                               if (posicion_cifra_f_intval > 1)
1021.                               {
1022.                                 posicion_cifra_f_intval /= 10;
1023.                               }
1024.                               else
1025.                               {
1026.                                 if (variable_funcion_copia == *variable_funcion)
1027.                                 {
1028.                                   if ((*indice_pantalla_actual % 10) == 0) //
    Miramos que no sea el final de un menú (que aún queden "0" a la derecha)
1029.                                   {
1030.                                       nivel_menu_nuevo = *nivel_menu_actual / 10;
     // Actualizamos el nivel del menu
1031.                                       indice_pantalla_nuevo = *indice_pantalla_ac
    tual + nivel_menu_nuevo;
1032.                                   }
1033.                                 }
1034.                                 else
1035.                                 {
1036.                                     *variable_funcion = variable_funcion_copia;
1037.                                     valor_modificado = 1;
1038.                                 }
1039.                                 // Apagamos el cursor, solo se enciende cuando
    entremos en la funcion 2 (introducir valor)
1040.                                 s_inst_lcd();
1041.                                 P2OUT = 0x0C; // Cursor off and blink off
1042.                                 s_latch_lcd();
1043.                                 P2OUT = 0xC0;
1044.                                 s_latch_lcd();
1045.                                 s_data_lcd();
1046.                               }
1047.                             }
1048.                             if ((P1IFG & BIT4) > 0) // p1.4 (boton abajo)  tiene
    el flag activo
1049.                             {
1050.                               if (((variable_funcion_copia / posicion_cifra_f_int
    val) % 10) != 0) // Aquí mirar si ya es 0 para no decrementarlo más
1051.                               {
1052.                                 variable_funcion_copia = variable_funcion_copia -
     posicion_cifra_f_intval;
1053.                               }
1054.                             }
1055.                         }
1056.                     }
1057.                 }
1058.
1059.       //-------------------------------------------------------------
1060.       //      TRATAMIENTO DE LA PANTALLA CORRESPONDIENTE
1061.       //-------------------------------------------------------------
1062.
1063.
1064.                 // Una vez conocido el nuevo indice pantalla actual lo buscamos
    en la estructura, y si está, lo mostramos en pantalla
```

```
1065.               // Aquí va el tratamiento de pantalla según el resultado de los
    botones
1066.
1067.               if (pantalla_inicio == 1) // En caso de estar en la pantalla
    general
1068.               {
1069.                 lcd escribe1("");
1070.               }
1071.               else
1072.               {
1073.               for (int j = 0; j<(*dimension_menu); j++)
1074.               {
1075.                   if (indice_pantalla_nuevo == menu_actual[j].indice_pantalla_s
    truct)
1076.                   {
1077.                     // En primer lugar se comprueba si tenemos que cambiar de
    struct o no, a continuación si es una función o pantalla ordinaria. Siguiendo
    este orden: funcion 1, funcion 2, pantalla ordinaria
1078.                     // La utilización de menús (structs) anidados nos permite
    reducir el tamaño de las structs (problema de memoria) así como gestionarlas
    independientemente si tenemos que cambiar su tamaño
1079.
1080.                     //::: Sección de menús anidados
1081.
1082.                     if (compara_strings(menu_actual[j].texto_pantalla_struct, "
    menu anidado") == 1) // Aquí sólo se entrará una vez, en el momento de entrar a
    un menú anidado. Comandos comunes a cualquier entrada en menú anidado
1083.                     {
1084.                     if (compara strings(menu actual[j].texto pantalla struct, "
    menu_anidado_alarmas") == 1)
1085.                     {
1086.                       // Al cambiar de menú (struct) se cambian las direcciones
    de los punteros
1087.                       menu actual = menu alarmas;
1088.                       dimension_menu = &dimension_menu_alarmas;
1089.                     }
1090.                     if (compara strings(menu actual[j].texto pantalla struct, "
    menu_anidado_temposbas") == 1)
1091.                     {
1092.                       // Al cambiar de menú (struct) se cambian las direcciones
    de los punteros
1093.                       menu_actual = menu_temporizaciones_basico;
1094.                       dimension_menu = &dimension_menu_temposbas;
1095.                     }
1096.                     if (compara_strings(menu_actual[j].texto_pantalla_struct, "
    menu anidado temposint") == 1)
1097.                     {
1098.                       // Al cambiar de menú (struct) se cambian las direcciones
    de los punteros
1099.                       menu_actual = menu_temporizaciones_inteligente;
1100.                       dimension_menu = &dimension_menu_temposint;
1101.                     }
1102.                     // Ahora se cambian los punteros de navegación (indice y
    nivel) a los del menú anidado.
1103.                     indice_pantalla_actual = &indice_pantalla_anidado;
1104.                     nivel menu actual = &nivel menu anidado;
1105.                     j = 0; // Para empezar mostrando el primer elemento
1106.                     nivel menu nuevo = 100000; // Esto se hace porque tras los
    tratamientos, se adjudica el valor nuevo al actual. Así concordamos los
    parámetros con la primera pantalla. el 1000 debe corresponder al primer elemento
    del array del menu (j = 0)
1107.                     indice pantalla nuevo = 100000;
1108.                     }
1109.
1110.                     // Como se verá, en cada función se deben poner todas las
    variables que se deban poder editar. Se relaciona el string con la variable.
    Manipulando dicha variable desde un puntero común (variable_función)
1111.
1112.                     // ::: Sección de función 1 (SI/NO - Confirmar/Cancelar)
1113.
1114.                       if ((compara_strings(menu_actual[j].texto_pantalla_struct
    , "f conf") == 1))
1115.                       {
1116.                           if ((indice_pantalla_nuevo != *indice_pantalla_actual
    ) || (menu actual != menu viejo)) // Acabo de entrar en esta pantalla de función.
    El cambio de menú es necesario
```

```
1117.                           {
1118.                             tipo funcion = 1;
1119.                               if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f conf temposbas") == 1) // Comparamos con la variable concreta
1120.                               {
1121.                                   variable_funcion = &f_conf_temposbas;
1122.                                   variable_funcion_copia = *variable_funcion;
1123.                                   lcd_escribe2(0, "Desea activar?");
1124.                               }
1125.                               if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f_conf_temposint") == 1) // Comparamos con la variable concreta
1126.                               {
1127.                                   variable_funcion = &f_conf_temposint;
1128.                                   variable_funcion_copia = *variable_funcion;
1129.                                   lcd_escribe2(0, "Desea activar?");
1130.                               }
1131.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_borrar_alarmas") == 1) // Comparamos con la variable concreta
1132.                               {
1133.                                   variable_funcion = &f_conf_borrar_alarmas;
1134.                                   *variable_funcion = 0; // En este tipo de
    decisiones, ponemos que no por defecto. Cada vez que se ponga SI se borraran las
    alarmas.
1135.                                   variable_funcion_copia = *variable_funcion;
1136.                                   lcd_escribe2(0, "Esta seguro?");
1137.                               }
1138.                               if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f conf borrar historico") == 1) // Comparamos con la variable concreta
1139.                               {
1140.                                   variable_funcion = &f_conf_borrar_historico;
1141.                                   *variable_funcion = 0; // En este tipo de
    decisiones, ponemos que no por defecto. Cada vez que se ponga SI se borraran las
    alarmas.
1142.                                   variable_funcion_copia = *variable_funcion;
1143.                                   lcd_escribe2(0, "Esta seguro?");
1144.                               }
1145.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_DPlavar") == 1) // Comparamos con la variable concreta
1146.                               {
1147.                                   variable_funcion = &f_conf_DPlavar;
1148.                                   variable_funcion_copia = *variable_funcion;
1149.                                   lcd_escribe2(0, "Desea activar?");
1150.                               }
1151.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_tempolavar") == 1) // Comparamos con la variable concreta
1152.                               {
1153.                                   variable_funcion = &f_conf_tempolavar;
1154.                                   variable_funcion_copia = *variable_funcion;
1155.                                   lcd_escribe2(0, "Desea activar?");
1156.                               }
1157.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f conf marchalavar") == 1) // Comparamos con la variable concreta
1158.                               {
1159.                                   variable_funcion = &f_conf_marchalavar;
1160.                                   variable_funcion_copia = *variable_funcion;
1161.                                   lcd escribe2(0, "Esta seguro?");
1162.                               }
1163.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_parolavar") == 1) // Comparamos con la variable concreta
1164.                               {
1165.                                   variable_funcion = &f conf parolavar;
1166.                                   variable_funcion_copia = *variable_funcion;
1167.                                   lcd escribe2(0, "Esta seguro?");
1168.                               }
1169.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_bucle_infinito") == 1) // Comparamos con la variable concreta
1170.                               {
1171.                                   variable_funcion = &f conf bucle infinito;
1172.                                   variable_funcion_copia = *variable_funcion;
1173.                                   lcd escribe2(0, "Habilitar alarma");
1174.                               }
1175.                               if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f conf alta presion") == 1) // Comparamos con la variable concreta
1176.                               {
1177.                                   variable_funcion = &f conf alta presion;
1178.                                   variable_funcion_copia = *variable_funcion;
```

```
1179.                                   lcd_escribe2(0, "Habilitar alarma");
1180.                                 }
1181.                                 if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_baja_presion") == 1) // Comparamos con la variable concreta
1182.                                 {
1183.                                   variable_funcion = &f_conf_baja_presion;
1184.                                   variable_funcion_copia = *variable_funcion;
1185.                                   lcd_escribe2(0, "Habilitar alarma");
1186.                                 }
1187.                               }
1188.                               else // Se entra aquí en las veces posteriores.
    Importante destacar tanto aquí como en la f2 que, solo pueden haberse producido
    cambios en las variables cuando se trata de una vez posterior.
1189.                               {
1190.                                 if (valor_modificado == 1) // Para que sólo se
    ejecute cuando se haya cambiado el valor
1191.                                 {
1192.                                   if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f_conf_temposbas") == 1) // Comparamos con la variable concreta
1193.                                   {
1194.                                     if (f_conf_temposbas == 1) // Sólo puede estar
    activo uno, pero almenos uno
1195.                                     {
1196.                                       f_conf_temposint = 0;
1197.                                     }
1198.                                     else // Nos aseguramos de que siempre uno esté
    activo.
1199.                                     {
1200.                                       f_conf_temposint = 1;
1201.                                     }
1202.                                   }
1203.                                   if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f_conf_temposint") == 1) // Comparamos con la variable concreta
1204.                                   {
1205.                                     if (f_conf_temposint == 1) // Sólo puede estar
    activo uno, pero almenos uno
1206.                                     {
1207.                                       f_conf_temposbas = 0;
1208.                                     }
1209.                                     else
1210.                                     {
1211.                                       f_conf_temposbas = 1;
1212.                                     }
1213.                                   }
1214.                                   if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f_conf_borrar_alarmas") == 1) // Operaciones para borrar alarmas
1215.                                   {
1216.                                     if(f_conf_borrar_alarmas == 1)
1217.                                     {
1218.                                     dimension_menu_alarmas = 1; // Ponemos la
    dimension a 1 para mostrar guiones que representan que está vacío
1219.                                     alarmas_vacio = 1;
1220.                                     menu_alarmas[0].texto_pantalla_struct = "Ningun
    a alarma";
1221.                                     baja_presion = 0;
1222.                                     baja_presion_anterior = 0;
1223.                                     alta_presion = 0;
1224.                                     alta_presion_anterior = 0;
1225.                                     bucle_infinito = 0;
1226.                                     bucle_infinito_anterior = 0;
1227.                                     }
1228.                                   }
1229.                                   if(compara_strings(menu_actual[j].texto_pantalla
    struct, "f_conf_borrar_historico"))
1230.                                   {
1231.                                     if(f_conf_borrar_historico ==1) // Similar al
    registro de alarmas
1232.                                     {
1233.                                       dimension_menu_historico = 1; // Ponemos la
    dimension a 1 para mostrar guiones que representan que está vacío
1234.                                       historico_vacio = 1;
1235.                                       menu_alarmas[0].texto_pantalla_struct = "Ning
    un registro";
1236.                                     }
1237.                                   }
```

```
1238.                               if(compara_strings(menu_actual[j].texto_pantalla_st
    ruct, "f conf DPlavar") == 1) // De esta forma impedimos que estén los 2 modos
    activos
1239.                                 {
1240.                                   if(f_conf_DPlavar == 1)
1241.                                   {
1242.                                     f conf tempolavar = 0;
1243.                                   }
1244.                                   else
1245.                                   {
1246.                                     bucle_infinito = 0;
1247.                                   }
1248.                                 }
1249.                                 if(compara strings(menu actual[j].texto pantalla st
    ruct, "f_conf_tempolavar") == 1) // De esta forma impedimos que estén los 2 modos
    activos
1250.                                 {
1251.                                   if(f_conf_tempolavar == 1)
1252.                                   {
1253.                                     f_conf_DPlavar = 0;
1254.                                     tempo limpieza = 0;
1255.                                     bucle infinito = 0;
1256.                                     TACCTL0 |= CCIE; // Habilitamos las interrupt
    del timer
1257.                                   }
1258.                                 }
1259.                                 if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f conf marchalavar") == 1)
1260.                                 {
1261.                                   if ((f_conf_marchalavar == 1) && (baja_presion
    == 0) && (alta presion == 0))
1262.                                   {
1263.                                     limpieza = 1; // Activamos limpieza
1264.                                     TACCTL0 |= CCIE; // Habilitacion de las
    interrupciones del timer
1265.                                     // No hace falta resetear el tempo porque ya
    se hace al principio de la secuencia
1266.                                   }
1267.                                 }
1268.                                 if(compara_strings(menu_actual[j].texto_pantalla_
    struct, "f conf parolavar") == 1) //
1269.                                 {
1270.                                   if (f conf parolavar == 1)
1271.                                   {
1272.                                     limpieza = 0; // Paramos limpieza
1273.                                     P3OUT &= 0x1F; // Apagado de las salidas
1274.                                     P4OUT &= 0xFC;
1275.                                     tempo limpieza = 0; // Reiniciamos así
    también podemos empezar a contar el tiempo desde el último lavado
1276.                                   }
1277.                                 }
1278.                               }
1279.                             }
1280.
1281.                             // A continuación la parte general de esta función -
    4 casos posibles: no no; si no ...
1282.                             if((*variable funcion == 0) && (variable funcion copi
    a == 0))
1283.                                 lcd_escribe2(1, "NO     Actual:NO");
1284.                             if((*variable funcion == 1) && (variable funcion copi
    a == 1))
1285.                                 lcd escribe2(1, "SI     Actual:SI");
1286.                             if((*variable funcion == 1) && (variable funcion copi
    a == 0))
1287.                                 lcd escribe2(1, "NO     Actual:SI");
1288.                             if((*variable_funcion == 0) && (variable_funcion_copi
    a == 1))
1289.                                 lcd escribe2(1, "SI     Actual:NO");
1290.                         }
1291.
1292.
1293.
1294.
1295.                         else
1296.                         {
1297.
```

```
1298.                         // ::: Sección de función 2 (Introducción de un valor
    por el usuario)
1299.
1300.                           if (compara_strings(menu_actual[j].texto_pantalla_str
    uct, "f intval") == 1)
1301.                           {
1302.                             // Seguimos con el tratamiento
1303.                             if((indice_pantalla_nuevo != *indice_pantalla_actua
    l) || (menu_actual != menu_viejo)) // Acciones a realizar sólo en el momento de
    entrar en la función
1304.                             {
1305.                               tipo_funcion = 2;
1306.                               posicion_cifra_f_intval = 1000;
1307.
1308.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_num_filtros") == 1) // Comparamos con la variable concreta
1309.                               {
1310.                                   variable_funcion = &f_intval_num_filtros;
1311.                                   variable_funcion_copia = *variable_funcion;
1312.                                   lcd_escribe2(0, "Num estac. max 5");
1313.                               }
1314.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_alarma") == 1) // Comparamos con la variable concreta
1315.                               {
1316.                                   variable_funcion = &crear_alarma;
1317.                                   variable_funcion_copia = *variable_funcion;
1318.                                   lcd_escribe2(0, "Defina la alarma");
1319.                               }
1320.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_DPlavar") == 1) // Comparamos con la variable concreta
1321.                               {
1322.                                   variable_funcion = &f_intval_DPlavar;
1323.                                   variable_funcion_copia = *variable_funcion;
1324.                                   lcd_escribe2(0, "Pres limit(mbar)");
1325.                               }
1326.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_tempolavar") == 1) // Comparamos con la variable concreta
1327.                               {
1328.                                   variable_funcion = &f_intval_tempolavar;
1329.                                   variable_funcion_copia = *variable_funcion;
1330.                                   lcd_escribe2(0, "Tiempo espera(s)");
1331.                               }
1332.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_alta_presion") == 1) // Comparamos con la variable concreta
1333.                               {
1334.                                   variable_funcion = &f_intval_alta_presion;
1335.                                   variable_funcion_copia = *variable_funcion;
1336.                                   lcd_escribe2(0, "Pres limit(mbar)");
1337.                               }
1338.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_limpiezas_bucle_infinito") == 1) // Comparamos con la variable
    concreta
1339.                               {
1340.                                   variable_funcion = &f_intval_limpiezas_bucle_
    infinito;
1341.                                   variable_funcion_copia = *variable_funcion;
1342.                                   lcd_escribe2(0, "Lavados seguidos");
1343.                               }
1344.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_estabilizado_DPlavar") == 1) // Comparamos con la variable
    concreta
1345.                               {
1346.                                   variable_funcion = &f_intval_estabilizado_DPl
    avar;
1347.                                   variable_funcion_copia = *variable_funcion;
1348.                                   lcd_escribe2(0, "tpo. estable(s)");
1349.                               }
1350.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_estabilizado_alta_presion") == 1) // Comparamos con la
    variable concreta
1351.                               {
1352.                                   variable_funcion = &f_intval_estabilizado_alt
    a_presion;
1353.                                   variable_funcion_copia = *variable_funcion;
1354.                                   lcd_escribe2(0, "tpo. estable(s)");
1355.                               }
```

```
1356.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_baja_presion") == 1) // Comparamos con la variable concreta
1357.                                 {
1358.                                     variable_funcion = &f_intval_baja_presion;
1359.                                     variable_funcion_copia = *variable_funcion;
1360.                                     lcd_escribe2(0, "Pres limit(mbar)");
1361.                                 }
1362.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_estabilizado_baja_presion") == 1) // Comparamos con la
    variable concreta
1363.                                 {
1364.                                     variable_funcion = &f_intval_estabilizado_baj
    a_presion;
1365.                                     variable_funcion_copia = *variable_funcion;
1366.                                     lcd_escribe2(0, "Pres limit(mbar)");
1367.                                 }
1368.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_tempo_bucle_infinito_entrar") == 1) // Comparamos con la
    variable concreta
1369.                                 {
1370.                                     variable_funcion = &f_intval_tempo_bucle_infi
    nito_entrar;
1371.                                     variable_funcion_copia = *variable_funcion;
1372.                                     lcd_escribe2(0, "Limp.consecut(s)");
1373.                                 }
1374.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_bucle_infinito_salir_estable") == 1) // Comparamos con la
    variable concreta
1375.                                 {
1376.                                     variable_funcion = &f_intval_bucle_infinito_s
    alir_estable;
1377.                                     variable_funcion_copia = *variable_funcion;
1378.                                     lcd_escribe2(0, "Estable salir(s)");
1379.                                 }
1380.                                 // Lista de tempos limpieza básica (tempos
    definidos por el usuario)
1381.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_baslav1") == 1)
1382.                                 {
1383.                                     variable_funcion = temposbas;
1384.                                     variable_funcion_copia = *variable_funcion;
1385.                                     lcd_escribe2(0, "Segundos lavando");
1386.                                 }
1387.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_basrep1") == 1)
1388.                                 {
1389.                                     variable_funcion = temposbas + 1;
1390.                                     variable_funcion_copia = *variable_funcion;
1391.                                     lcd_escribe2(0, "Segundos reposo");
1392.                                 }
1393.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_baslav2") == 1)
1394.                                 {
1395.                                     variable_funcion = temposbas + 2;
1396.                                     variable_funcion_copia = *variable_funcion;
1397.                                     lcd_escribe2(0, "Segundos lavando");
1398.                                 }
1399.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_basrep2") == 1)
1400.                                 {
1401.                                     variable_funcion = temposbas + 3;
1402.                                     variable_funcion_copia = *variable_funcion;
1403.                                     lcd_escribe2(0, "Segundos reposo");
1404.                                 }
1405.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_baslav3") == 1)
1406.                                 {
1407.                                     variable_funcion = temposbas + 4;
1408.                                     variable_funcion_copia = *variable_funcion;
1409.                                     lcd_escribe2(0, "Segundos lavando");
1410.                                 }
1411.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_basrep3") == 1)
1412.                                 {
1413.                                     variable_funcion = temposbas + 5;
1414.                                     variable_funcion_copia = *variable_funcion;
```

```
1415.                                   lcd_escribe2(0, "Segundos reposo");
1416.                               }
1417.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_baslav4") == 1)
1418.                               {
1419.                                   variable_funcion = temposbas + 6;
1420.                                   variable_funcion_copia = *variable_funcion;
1421.                                   lcd_escribe2(0, "Segundos lavando");
1422.                               }
1423.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_basrep4") == 1)
1424.                               {
1425.                                   variable_funcion = temposbas + 7;
1426.                                   variable_funcion_copia = *variable_funcion;
1427.                                   lcd_escribe2(0, "Segundos reposo");
1428.                               }
1429.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_baslav5") == 1)
1430.                               {
1431.                                   variable_funcion = temposbas + 8;
1432.                                   variable_funcion_copia = *variable_funcion;
1433.                                   lcd_escribe2(0, "Segundos lavando");
1434.                               }
1435.
1436.                               // Lista de tempos limpieza inteligente (el
    usuario define coeficientes cuyas uds son segundos/bar)
1437.                               if (compara_strings(menu_actual[j].texto_pantalla
     struct, "f_intval_intlav1") == 1)
1438.                               {
1439.                                   variable_funcion = temposint;
1440.                                   variable_funcion_copia = *variable_funcion;
1441.                                   lcd_escribe2(0, "Seg/bar lavando");
1442.                               }
1443.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_intrep1") == 1)
1444.                               {
1445.                                   variable_funcion = temposint + 1;
1446.                                   variable_funcion_copia = *variable_funcion;
1447.                                   lcd_escribe2(0, "Segundos reposo");
1448.                               }
1449.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_intlav2") == 1)
1450.                               {
1451.                                   variable_funcion = temposint + 2;
1452.                                   variable_funcion_copia = *variable_funcion;
1453.                                   lcd_escribe2(0, "Seg/bar lavando");
1454.                               }
1455.                               if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_intrep2") == 1)
1456.                               {
1457.                                   variable_funcion = temposint + 3;
1458.                                   variable_funcion_copia = *variable_funcion;
1459.                                   lcd_escribe2(0, "Segundos reposo");
1460.                               }
1461.                               if (compara_strings(menu_actual[j].texto_pantalla
     struct, "f_intval_intlav3") == 1)
1462.                               {
1463.                                   variable_funcion = temposint + 4;
1464.                                   variable_funcion_copia = *variable_funcion;
1465.                                   lcd_escribe2(0, "Seg/bar lavando");
1466.                               }
1467.                               if (compara_strings(menu_actual[j].texto_pantalla
     struct, "f_intval_intrep3") == 1)
1468.                               {
1469.                                   variable_funcion = temposint + 5;
1470.                                   variable_funcion_copia = *variable_funcion;
1471.                                   lcd_escribe2(0, "Segundos reposo");
1472.                               }
1473.                               if (compara_strings(menu_actual[j].texto_pantalla
     struct, "f_intval_intlav4") == 1)
1474.                               {
1475.                                   variable_funcion = temposint + 6;
1476.                                   variable_funcion_copia = *variable_funcion;
1477.                                   lcd_escribe2(0, "Seg/bar lavando");
1478.                               }
```

```
1479.                                 if (compara_strings(menu_actual[j].texto_pantalla
    struct, "f_intval_intrep4") == 1)
1480.                                 {
1481.                                     variable_funcion = temposint + 7;
1482.                                     variable_funcion_copia = *variable_funcion;
1483.                                     lcd_escribe2(0, "Segundos reposo");
1484.                                 }
1485.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_intlav5") == 1)
1486.                                 {
1487.                                     variable_funcion = temposint + 8;
1488.                                     variable_funcion_copia = *variable_funcion;
1489.                                     lcd_escribe2(0, "Seg/bar lavando");
1490.                                 }
1491.                               }
1492.                               else // Entramos aqui si ya estabamos dentro de la
    funcion.
1493.                               {
1494.                                 if (valor_modificado == 1)
1495.                                 {
1496.                                 // Operaciones con los valores editados.
1497.
1498.                                 // Numero de filtros del equipo
1499.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_num_filtros") == 1)
1500.                                 {
1501.                                   if(f_intval_num_filtros == 0)
1502.                                   {
1503.                                     menu_temporizaciones_basico[0].texto_pantalla
    _struct = "0 filtros";
1504.                                     menu_temporizaciones_inteligente[0].texto_pan
    talla_struct = "0 filtros";
1505.                                     dimension_menu_temposbas = 1;
1506.                                     dimension_menu_temposint = dimension_menu_tem
    posbas;
1507.                                   }
1508.                                   else
1509.                                   {
1510.                                     menu_temporizaciones_basico[0].texto_pantalla
    _struct = "tpo(s) lavar E1";
1511.                                     menu_temporizaciones_inteligente[0].texto_pan
    talla_struct = "seg/bar lavar E1";
1512.                                     if (f_intval_num_filtros > 5) // Limitado
    porque al utilizar una cifra, el menu solo va de 1 a 9, luego 5 filtros y 4
    reposos (modificable si se precisa)
1513.                                     {
1514.                                       f_intval_num_filtros = 5;
1515.                                     }
1516.                                     dimension_menu_temposbas = 4 * f_intval_num_f
    iltros - 2;
1517.                                     dimension_menu_temposint = dimension_menu_tem
    posbas;
1518.                                   }
1519.                                 }
1520.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_DPlavar") == 1) // Acotamos los valores admisibles
1521.                                 {
1522.                                   if(f_intval_DPlavar < (f_intval_baja_presion +
    50))
1523.                                   {
1524.                                     f_intval_DPlavar = f_intval_baja_presion + 50
    ;
1525.                                   }
1526.                                   else
1527.                                   {
1528.                                     if(f_intval_DPlavar > (f_intval_alta_presion
    - 50))
1529.                                     {
1530.                                       f_intval_DPlavar = f_intval_alta_presion -
    50;
1531.                                     }
1532.                                     if (f_intval_DPlavar > 1000)
1533.                                     {
1534.                                       f_intval_DPlavar = 1000;
1535.                                     }
1536.                                   }
```

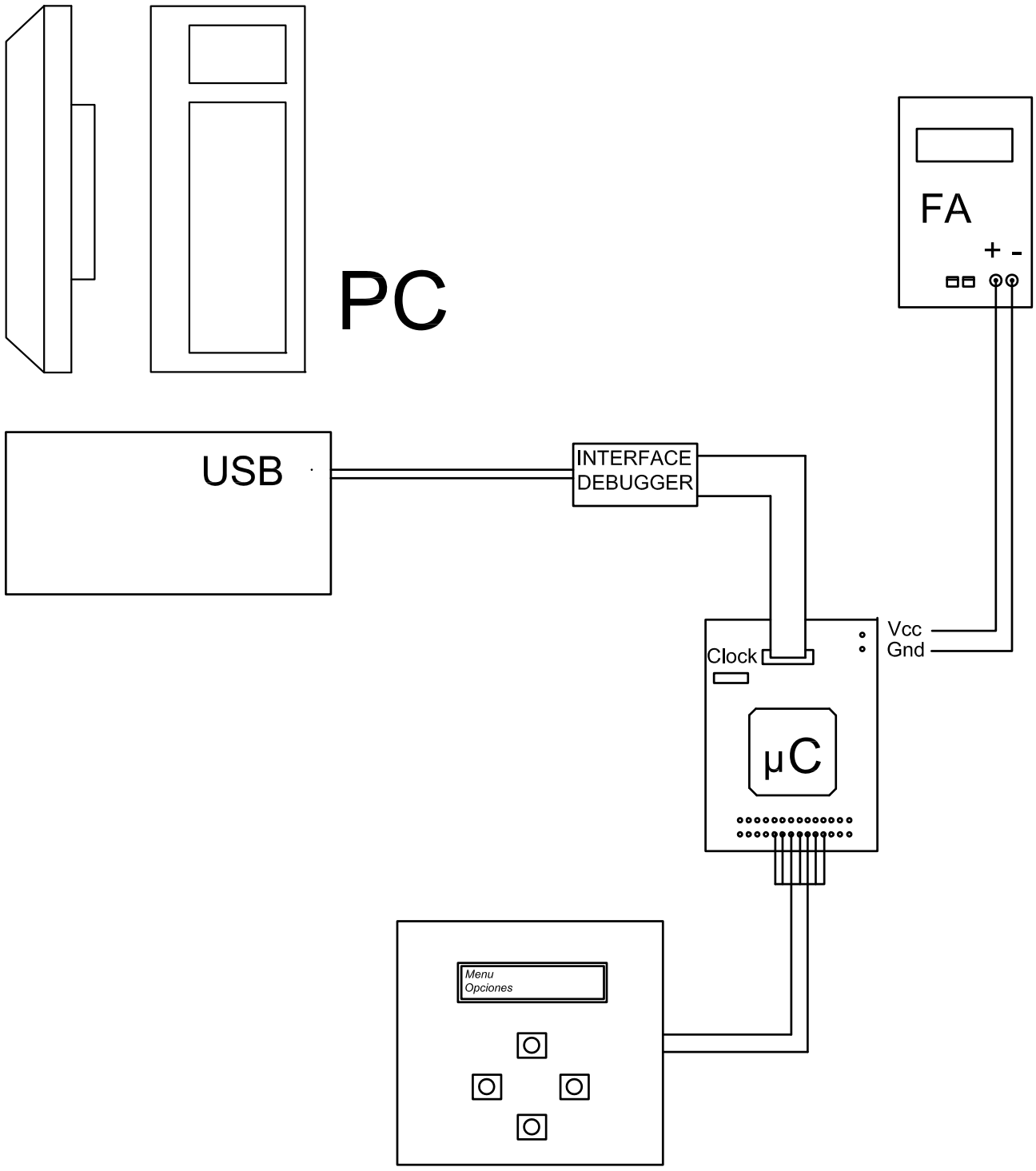
```
1537.                                 }
1538.                                 if (compara_strings(menu_actual[j].texto_pantalla
    _struct, "f_intval_alta_presion") == 1) // Acotamos los valores admisibles
1539.                                 {
1540.                                   if(f_intval_alta_presion > 2000) // Ya que el
    transductor no mide presiones superiores a 2000 mbar
1541.                                   {
1542.                                     f_intval_alta_presion = 2000;
1543.                                   }
1544.                                   else
1545.                                   {
1546.                                     if(f_intval_alta_presion < (f_intval_DPlavar
    + 50))
1547.                                     {
1548.                                       f_intval_alta_presion = f_intval_DPlavar +
    50;
1549.                                     }
1550.                                   }
1551.                                 }
1552.                                 if (compara_strings(menu_actual[j].texto_pantalla
     _struct, "f_intval_baja_presion") == 1) // Acotamos los valores admisibles
1553.                                 {
1554.                                   if(f_intval_baja_presion > (f_intval_DPlavar -
     50))
1555.                                   {
1556.                                     f_intval_baja_presion = f_intval_DPlavar -
     50;
1557.                                   }
1558.                                 }
1559.                               }
1560.                             }
1561.
1562.                                 // A continuación se hace un proceso para mostrar
    el valor copia y el definitivo
1563.                                 // La variable "valores_f2" está inicializada al
    principio como "     Actual:    "
1564.
1565.                                 //Para variable funcion copia:
1566.                                 int i = 1;
1567.                                 for (int j = 0; j<4; j++)
1568.                                 {
1569.                                   valores_f_intval[j] = ((variable_funcion_copia
    / (1000 / i)) % 10) + '0';
1570.                                   i *= 10;
1571.                                 }
1572.                                 //Para *variable_funcion:
1573.                                 i = 1;
1574.                                 for (int j = 0; j<4; j++)
1575.                                 {
1576.                                   valores_f_intval[j + 12] = ((*variable_funcion
    / (1000 / i)) % 10) + '0';
1577.                                   i *= 10;
1578.                                 }
1579.                                 //Escribir el texto
1580.                                 lcd_escribe2(1, valores_f_intval);
1581.                                 // Enciendo el cursor
1582.                                 s_inst_lcd();
1583.                                 P2OUT = 0x0F; // Cursor off and blink off
1584.                                 s_latch_lcd();
1585.                                 P2OUT = 0xF0;
1586.                                 s_latch_lcd();
1587.                                 // Poner cursor donde toque (192 es el primer
    caracter de la segunda línea)
1588.                                 P2OUT = ((192 + (3 -
     (int)log10(posicion_cifra_f_intval))) & 240); // Lo tengo que cortar en dos
    partes de 4 bits para poder enviarlo
1589.                                 s_latch_lcd();
1590.                                 P2OUT = ((192 + (3 -
     (int)log10(posicion_cifra_f_intval))) & 15) * 16; // Lo tengo que cortar en dos
    partes de 4 bits para poder enviarlo
1591.                                 s_latch_lcd();
1592.                               }
1593.
1594.
1595.
1596.
```

```
1597.                           // ::: Sección de pantalla de menú ordinaria
1598.
1599.                           else
1600.                           {
1601.                               lcd_escribe1(menu_actual[j].texto_pantalla_struct
    );
1602.                               tipo funcion = 0;
1603.                           }
1604.                       }
1605.                               *indice_pantalla_actual = indice_pantalla_nuevo;
1606.                               *nivel_menu_actual = nivel_menu_nuevo;
1607.                               menu viejo = menu actual;
1608.                   }
1609.               }
1610.               }
1611.
1612.               valor modificado = 0; //Reinicio la variable que me indica si
    acabo de modificar un valor
1613.           }
1614.
1615.       // Aquí va el tratamiento de los botones en caso de ya estar dentro de
    una función
1616.
1617.       //  Funcion editar valor
1618.           // Hacer parpadear la cifra donde estemos
1619.           // No actualizar el valor de la variable hasta que sea definitivo (si
    acaso manejar una copia en otra variable)
1620.           //
1621.
1622.
1623.           // Se reinicializa el flag por si se ha entrado en la interrupcion
    por un rebote, y se vuelven a habilitar las interrupciones
1624.
1625.           P1IFG = 0x00;
1626.           P1IE = 0x1B;
1627.       }
1628.
1629.       void delay(int n) // Universal delay routine
1630.       {
1631.           int i=0,j=0;
1632.           for(i=0; i<=n; i++)
1633.               for(j=0; j<=10; j++);
1634.       }
1635.
1636.       void s_inst_lcd(void) //Instruction select routine
1637.       {
1638.           P1OUT&=~BIT5; //RS=0
1639.           P1OUT&=~BIT6; //RW=0
1640.       }
1641.       void s_data_lcd(void) // Data select routine
1642.       {
1643.           P1OUT|=BIT5;    //RS=1
1644.           P1OUT&=~BIT6;   //RW=0
1645.       }
1646.       void s_latch_lcd(void) // Latch routine
1647.       {
1648.           P1OUT|=BIT7;  //EN=1
1649.           delay(20);
1650.           P1OUT&=~BIT7; //EN=0
1651.           delay(20);
1652.       }
1653.
1654.       void lcd escribe1 (char texto[32]) // Escribe en la primera linea hasta
    llegar a los 16 caracteres, luego pasa a la segunda
1655.       {
1656.           //Se borra todo y se pone cursor en home
1657.           s inst lcd();
1658.           P2OUT = 0x01;
1659.           s_latch_lcd();
1660.           P2OUT = 0x10;
1661.           s_latch_lcd();
1662.
1663.           //Se escribe hasta que se encuentre el \0. Se comprueba si estamos en
    el caracter 16 para pasar de linea
1664.           s data lcd();
1665.           int i = 0;
```

```
1666.           while (texto[i] != '\0')
1667.           {
1668.             if (i == 16)
1669.             {
1670.               s_inst_lcd();
1671.               P2OUT = 0xC0; // Situamos el cursor en el primer caracter
1672.               s_latch_lcd();
1673.               P2OUT = 0x00;
1674.               s_latch_lcd(); // Segundo nibble
1675.               s_data_lcd();
1676.             }
1677.             P2OUT = texto[i] & 240; // Opero con el char (que en realidad es un
    int con el numero ascii) para dividir el byte en dos partes de 4 bits
1678.             s_latch_lcd();
1679.             P2OUT = (texto[i] & 15) * 16;
1680.             s_latch_lcd();
1681.
1682.             i++;
1683.           }
1684.       }
1685.
1686.       void lcd_escribe2 (int linea, char texto[16]) // Escribe máximo 16
    caracteres en la linea indicada (0 ó 1) en lcd: 2 x 16
1687.       {
1688.           int i; // índice del bucle
1689.           int ddram_address; // variable con la dirección para el lcd en valor
    decimal
1690.           switch (linea)
1691.           {
1692.           case 0:
1693.               ddram_address = 128; // hex: 80
1694.               break;
1695.           case 1:
1696.               ddram_address = 192; // hex: C0
1697.               break;
1698.           }
1699.           // Me situo al principio de la línea para borrar lo que había
1700.           s_inst_lcd();
1701.           P2OUT = ddram_address & 240;
1702.           s_latch_lcd();
1703.           P2OUT = (ddram_address & 15) * 16;
1704.           s_latch_lcd();
1705.
1706.           s_data_lcd();
1707.           for(i=0; i<16; i++)
1708.           {
1709.               P2OUT = 0x20; // Enviar un espacio
1710.               s_latch_lcd();
1711.               P2OUT = 0x00;
1712.               s_latch_lcd();
1713.
1714.           }
1715.
1716.           // Me situo otra vez al principio de la línea para escribir el texto
    especificado
1717.           s_inst_lcd();
1718.           P2OUT = ddram_address & 240;
1719.           s_latch_lcd();
1720.           P2OUT = (ddram_address & 15) * 16;
1721.           s_latch_lcd();
1722.
1723.           s_data_lcd();
1724.           i = 0;
1725.           while(texto[i] != '\0')
1726.           {
1727.               P2OUT = texto[i] & 240;
1728.               s_latch_lcd();
1729.               P2OUT = (texto[i] & 15) * 16;
1730.               s_latch_lcd();
1731.
1732.               i++;
1733.           }
1734.       }
1735.
1736.       int compara_strings (char a[], char b[]) // Compara char hasta que uno
    termina. Por ejemplo: "abcdef" y "ab" los daría como iguales
```

```
1737.       {
1738.           int c = 0;
1739.
1740.           while (a[c] == b[c]) {
1741.               if (a[c] == '\0' || b[c] == '\0')
1742.                   break;
1743.               c++;
1744.           }
1745.
1746.           if (a[c] == '\0' || b[c] == '\0')
1747.               return 1; // Coinciden
1748.           else
1749.               return 0; /// No coinciden
1750.       }
```

3 PLANOS

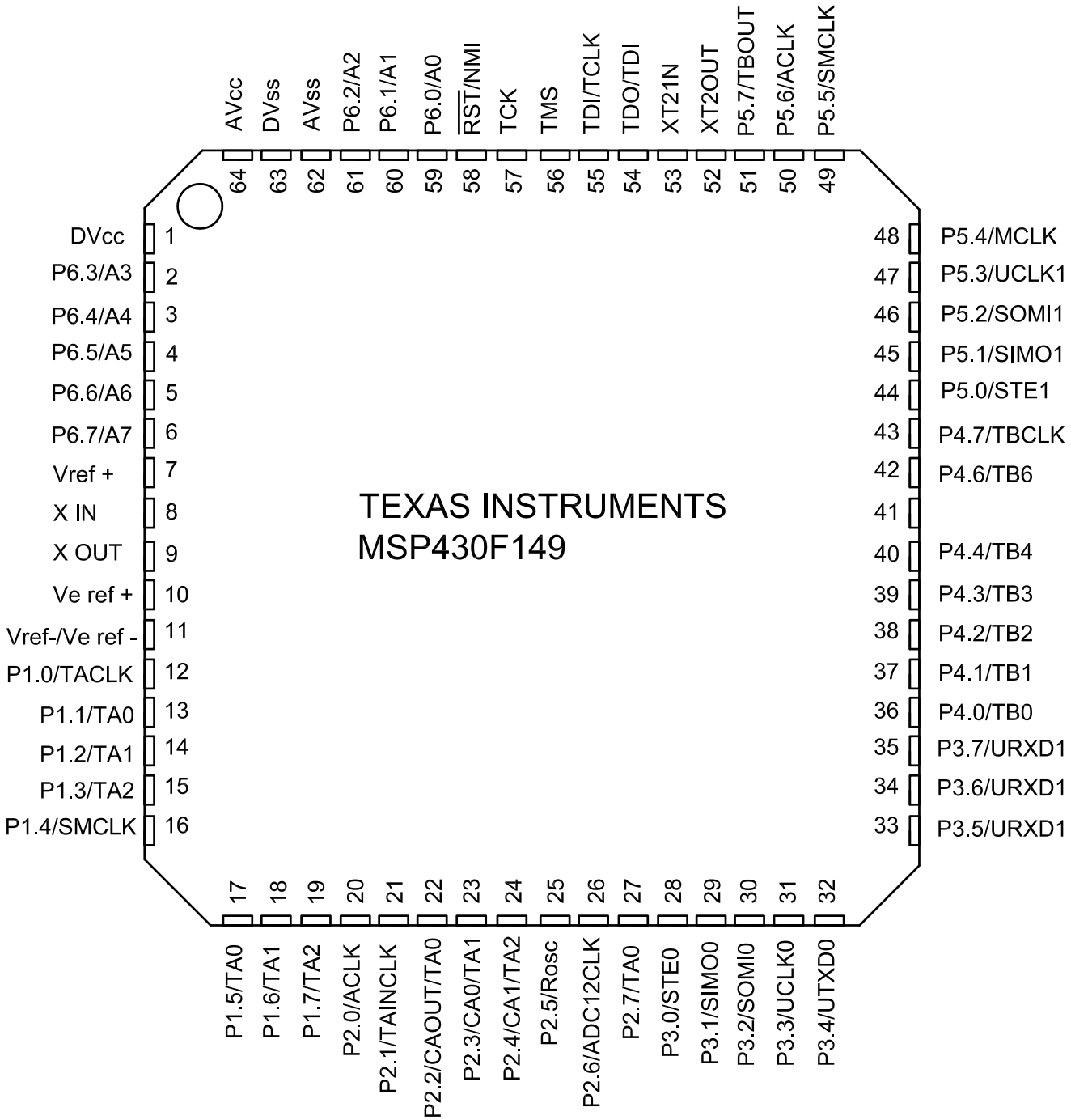


PROYECTO: DE CONTROLADOR DE LIMPIEZA DE FILTROS.

PLANO 1: Conjunto de los dispositivos.

PETICIONARIO: Benjamín Carratalá Font.

PROYECTO FINAL DE MASTER, UNIVERSIDAD DE CASTELLÓN.	Firma:	Ingeniero Industrial. B. E. Carratalá.	Escala: - /---. Fecha: 3-Julio-2018. Expediente:



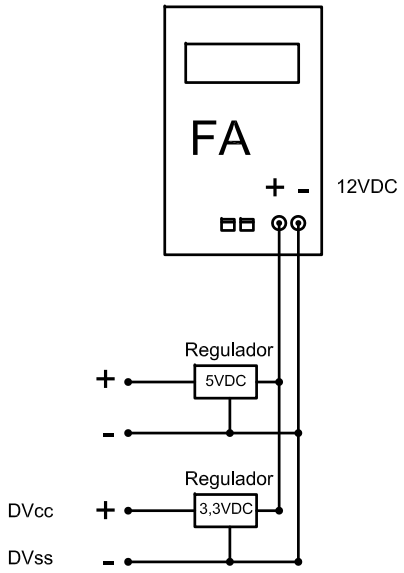
PROYECTO: *DE CONTROLADOR DE LIMPIEZA DE FILTROS.*

PLANO 2: *MICROCONTROLADOR.*

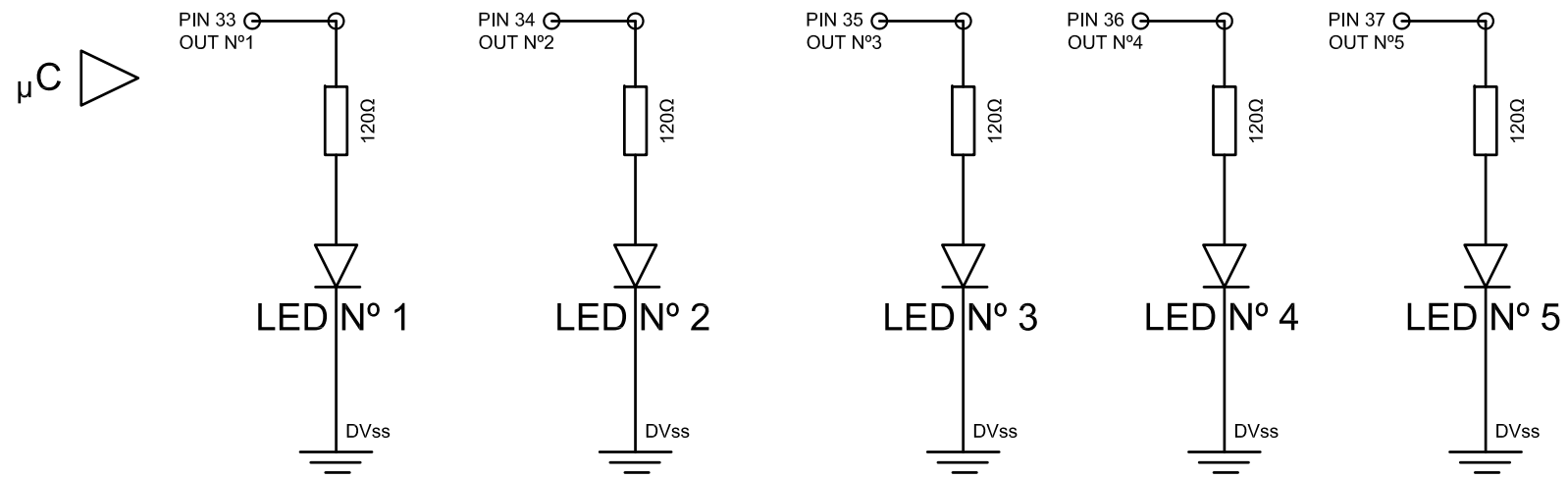
PETICIONARIO: *UNIVERSIDAD DE CASTELLÓN.*

PROYECTO FINAL DE MASTER, UNIVERSIDAD DE CASTELLÓN.	**Firma:**	*Ingeniero Industrial.* *B. Carratalá.*	*Escala: -------.* *Fecha: 3-Julio-2018.* *Expediente:*

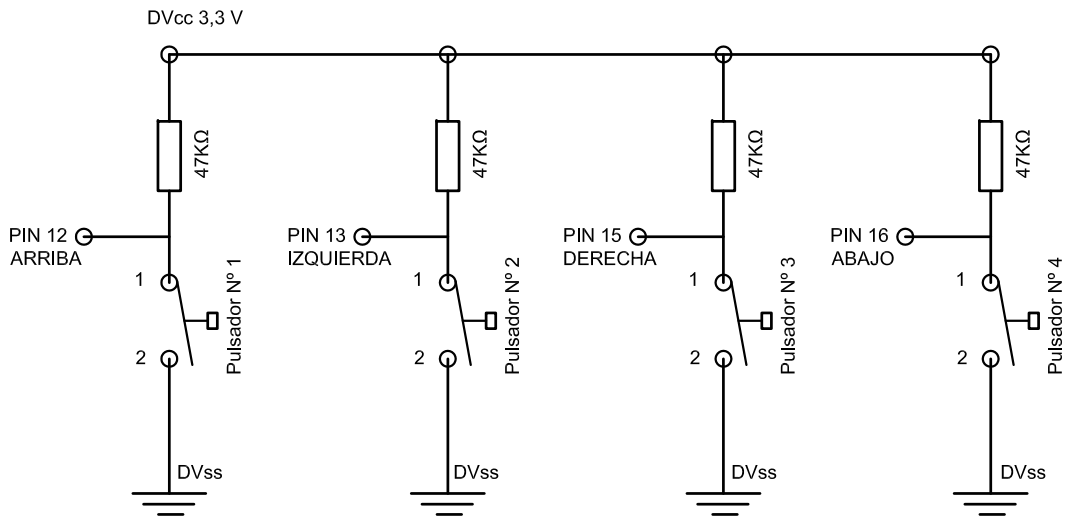
ALIMENTACIÓN DEL CIRCUITO



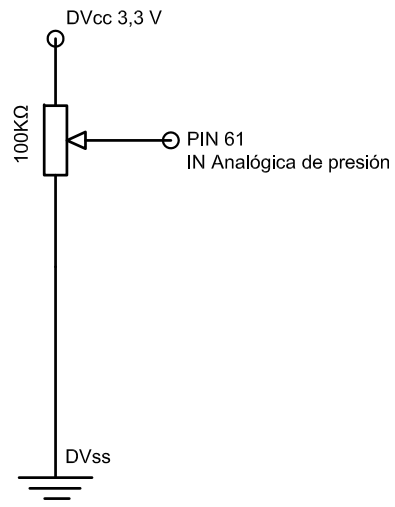
SALIDAS DIGITALES DE CONEXIONADO DE LIMPIEZA CIRCUITO



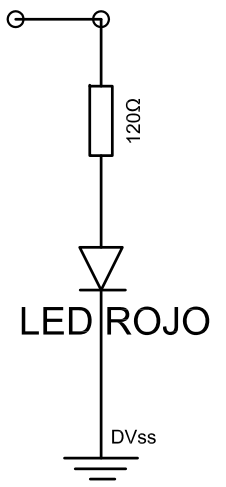
CONEXIONADO DE MICROPULSADORES



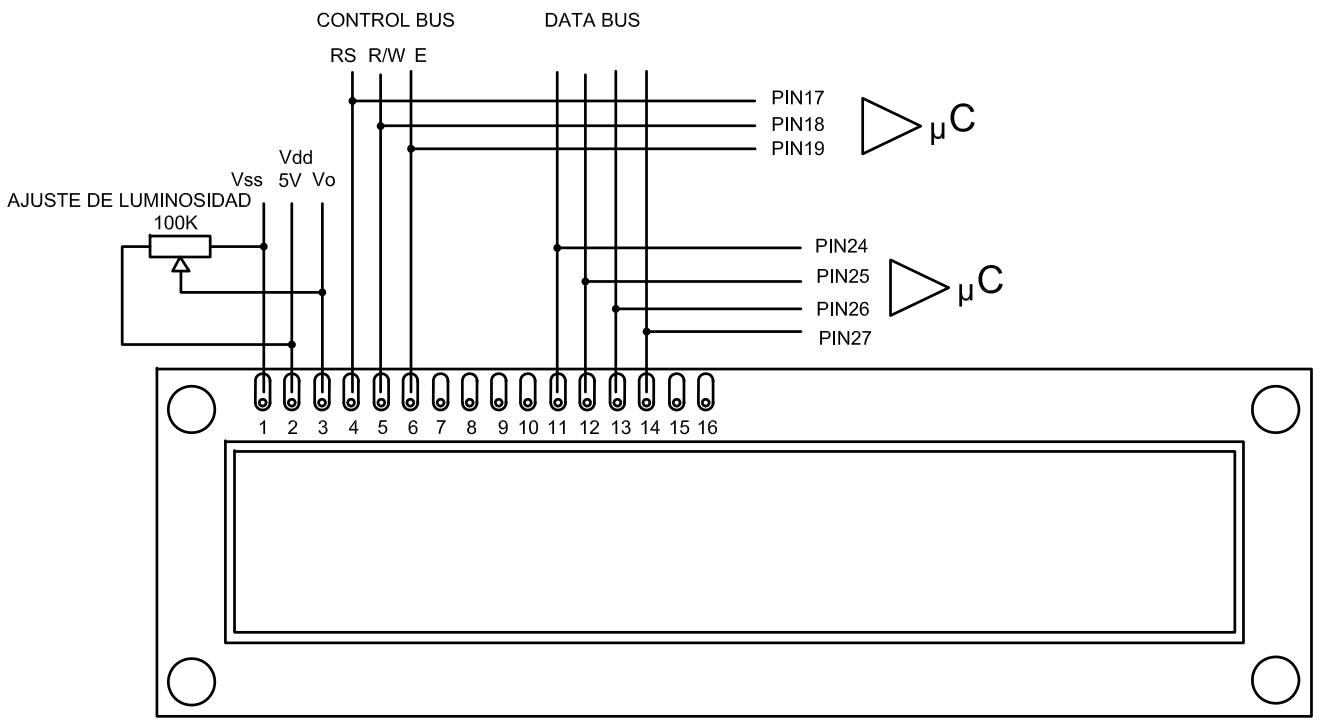
ENTRADA ANALÓGICA DE PRESIÓN



SALIDA DE INTERRUPCIÓN DEL TIMER ACTIVA

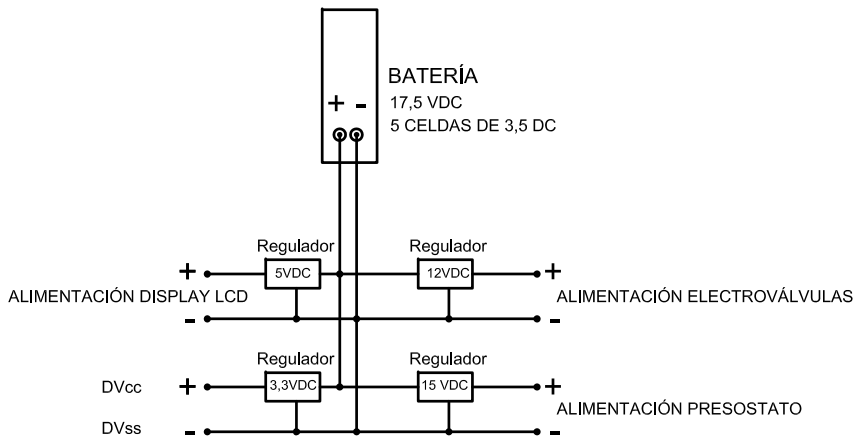


CONEXIONADO DEL DISPLAY LCD

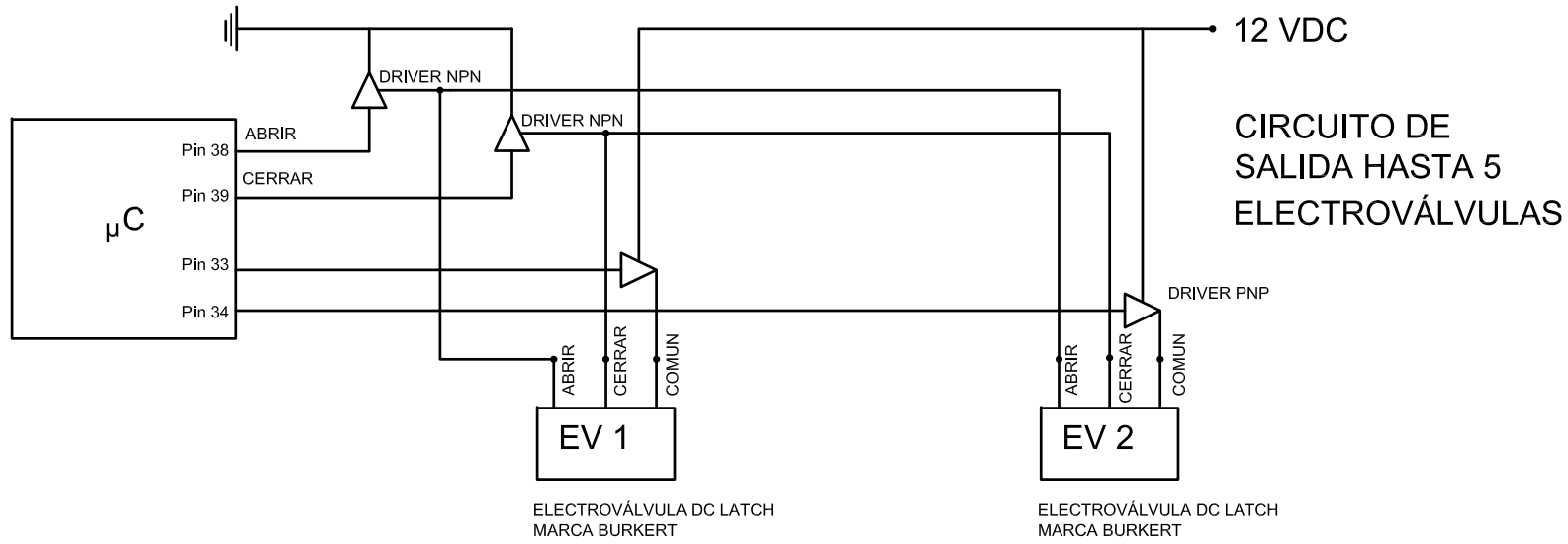


PROYECTO: DE CONTROLADOR DE LIMPIEZA DE FILTROS.			
PLANO 3: CONEXIONADO DEL CIRCUITO DE DESARROLLO..			
PETICIONARIO: UNIVERSIDAD DE CASTELLÓN.			
PROYECTO FINAL DE MASTER, UNIVERSIDAD DE CASTELLÓN.	Firma:	Ingeniero Industrial. B. Carratalá.	Escala: -------. Fecha: 3-Julio-2018. Expediente:

ALIMENTACIÓN DEL CIRCUITO

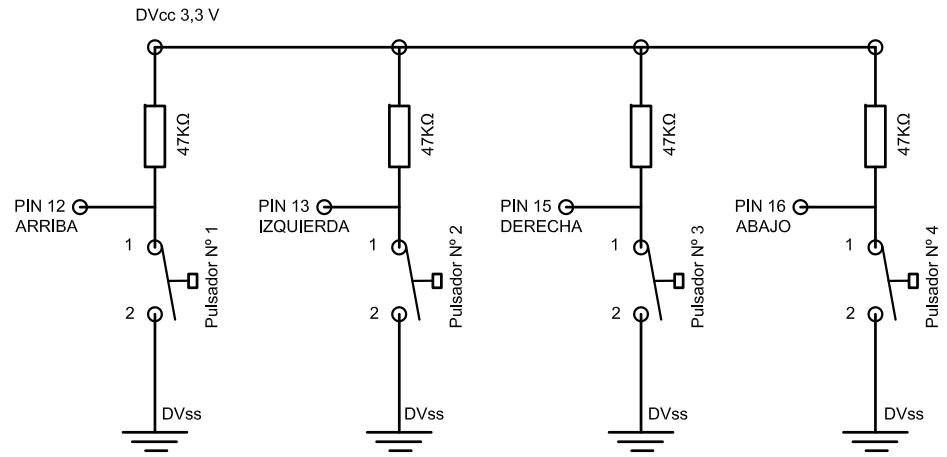


SALIDAS DIGITALES DE CONEXIONADO DE LIMPIEZA CIRCUITO

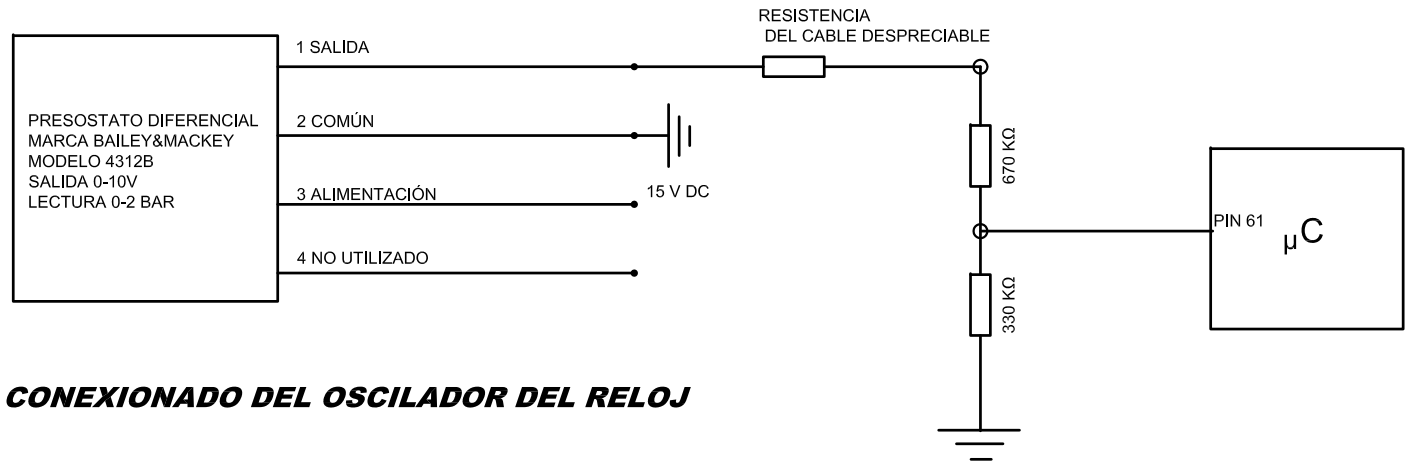


CIRCUITO DE SALIDA HASTA 5 ELECTROVÁLVULAS

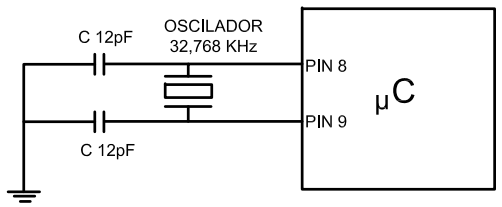
CONEXIONADO DE MICROPULSADORES



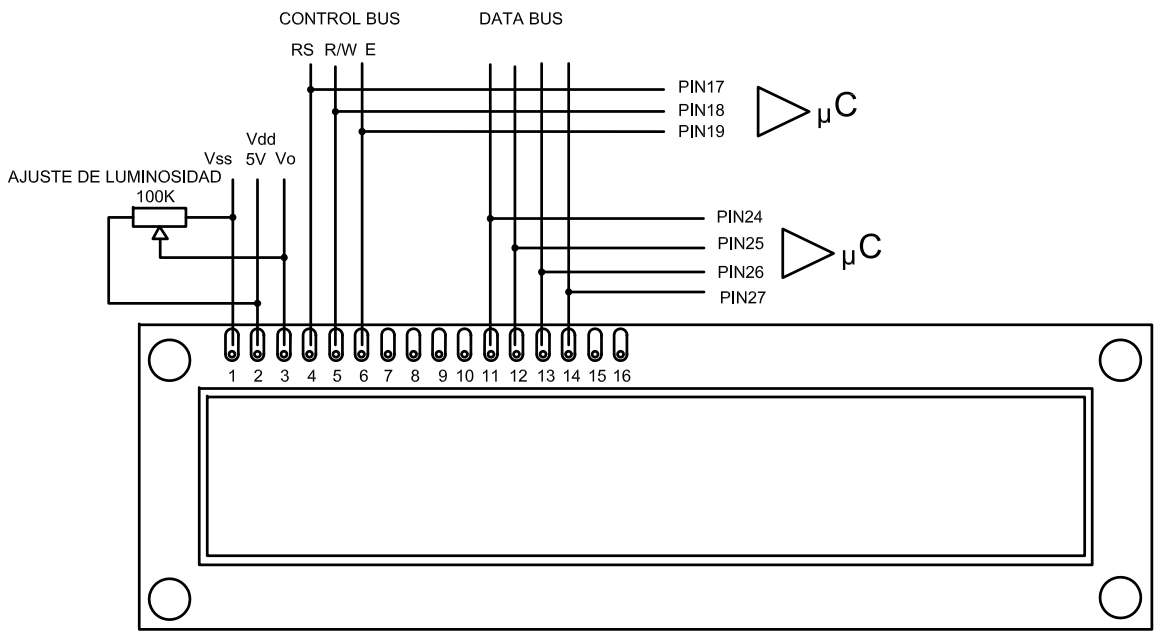
CONEXIONADO DE LA SEÑAL DEL PRESOSTATO



CONEXIONADO DEL OSCILADOR DEL RELOJ



CONEXIONADO DEL DISPLAY LCD



PROYECTO: DE CONTROLADOR DE LIMPIEZA DE FILTROS.			
PLANO 4: CONEXIONADO DEL CIRCUITO DE APLICACIÓN.			
PETICIONARIO: UNIVERSIDAD DE CASTELLÓN.			
PROYECTO FINAL DE MASTER, UNIVERSIDAD DE CASTELLÓN.	***Firma:***	*Ingeniero Industrial.* *B. Carratalá.*	*Escala: -------.* *Fecha: 3-Julio-2018.* *Expediente:*

4 PLIEGO DE CONDICIONES

Para el desarrollo del proyecto necesitaremos unos equipos y dispositivos que presenten unas determinadas prestaciones técnicas, por lo que en este capítulo daremos justificación de los medios técnicos necesarios para realizar el desarrollo del circuito electrónico de control de la limpieza de los filtros de una instalación automatizada de riego de cultivos.

4.1 MEDIOS INFORMÁTICOS.

4.1.1 ORDENADOR PC

En nuestro caso solamente requerimos la utilización de un ordenador PC de 4GB de memoria RAM y con un disco duro de 1TB será suficiente para realizar el desarrollo de nuestra aplicación.

El sistema operativo del ordenador tiene que ser un WINDOWS7 y el ordenador con un microprocesador de 32 bits será suficiente.

4.1.2 SOFTWARE.

Para el desarrollo de la aplicación necesitaremos el ordenador PC con sistema operativo WINDOWS7 y los siguientes programas de desarrollo:

1.- Sofware de desarrollo en C, IDE para el microcontrolador TEXAS INSTRUMENTS de la familia MSP430.

2.- Programa Office de WINDOWS para realizar los documentos del proyecto.

3.- Programa de AUTOCAD para realizar los esquemas eléctricos y electrónicos de la instalación.

4.- Navegador CHROME de GOOGLE, para tener acceso y navegar en internet para consulta de información.

4.2 HERRAMIENTAS DE DESARROLLO ELECTRÓNICO.

1.- Dispositivo electrónico de programación del microcontrolador y depuración del programa, mediante la ejecución paso a paso del programa y lectura de los registros del microcontrolador en tiempo real, así como la visualización de los errores de compilación.

2.- Placa electrónica de prototipo en la que se conecta el Display LCD para visualizar las opciones del controlador con pulsadores para manejar los menús, salidas visuales con leds para las salidas de las electroválvulas y potenciómetro para la simulación de la entrada analógica del transductor de la presión diferencial de los filtros.

3.- Fuente de alimentación externa regulable para alimentar los distintos componentes.

5 MEDICIONES

5.1 INVERSIÓN EN MEDIOS.

Nº	DESCRIPCIÓN	CANTIDAD.	
1.-	Amortización Equipo Informático.	1	5% del precio
2.-	Amortización de Equipos de Medición.	1	5% del precio
3.-	Amortización licencias de software.	1	5% del precio
4.-	Amortización del desarrollo del circuito electrónico.	1	5% del precio

5.2 MATERIALES DE LA INSTALACIÓN.

1.-	Circuito electrónico de control.	1
2.-	Transductor de presión diferencial Bailey&Mackey.	1
3.-	Electroválvulas BURKERT para riego.	5
4.-	Pequeño material de la instalación, cables, etc…	1

5.3 MANO DE OBRA DE LA INSTALACIÓN.

1.- Oficial instalador de la empresa.	12	horas de trabajo.
2.- Trabajador instalador de la empresa.	12	horas de trabajo.

5.4 BENEFICIO INDUSTRIAL (15%)

1.- Beneficio industrial.	1	15% del coste.

6 PRESUPUESTO

6.1 INVERSIÓN EN MEDIOS.

Nº	DESCRIPCIÓN	CANTIDAD	PRECIO	TOTAL EUROS.
1.-	Amortización Equipo Informático.	1	150	150 Euros.
2.-	Amortización de Equipos de Medición.	1	250	250 Euros.
3.-	Amortización licencias de software.	1	350	350 Euros.
4.-	Amortización del desarrollo del circuito electrónico.	1	500	500 Euros.
			Subtotal	***1.250 Euros.***

6.2 MATERIALES DE LA INSTALACIÓN.

1.-	Circuito electrónico de control.	1	575	575 Euros.
2.-	Transductor de presión diferencial Bailey&Mackey.	1	180	180 Euros.
3.-	Electroválvulas BURKERT para riego.	5	45	225 Euros.
4.-	Pequeño material de la instalación, cables, etc…	1	245	245 Euros.
			Subtotal	***1.225 Euros.***

6.3 MANO DE OBRA DE LA INSTALACIÓN.

1.- Oficial instalador de la empresa.	12	40	480 Euros.
2.- Trabajador instalador de la empresa.	12	25	300 Euros.
		Subtotal	***780 Euros.***

6.4 BENEFICIO INDUSTRIAL, (15%).

1.- Beneficio industrial.	1	488	488 Euros
		Subtotal	***488 Euros.***

6.5 RESUMEN DE LAS PARTIDAS Y TOTAL.

1.- Inversiones en medios.	1.250 Euros.
2.- Materiales de la instalación.	1.225 Euros.
3.- Mano de obra de la instalación.	780 Euros.
4.- Beneficio industrial.	488 Euros
TOTAL	**3.743 Euros.**

El total del presupuesto es de tres mil setecientos cuarenta y tres Euros.

EL PRESUPUESTO NO INCLUYE LOS IMPUESTOS SEGÚN LA LEGISLACIÓN VIGENTE.

Vila-Real a 3 de Julio de 2.018.

El Peticionario: **El Ingeniero Industrial:**

UNIVERSITAT JAUME I *Benjamín Carratalá Font.*