22449 -SISTEMAS EMPOTRADOS

# PRÁCTICA FINAL

## SISTEMA EMPOTRADO DE TIEMPO REAL DISTRIBUIDO

## ALARMA GENERAL DE UNA CASA

**Autores:**

Sergi Vivo Valls - 45695276A
Miquel Robles Mclean - 41615695R

UNIVERSITAT DE LES ILLES BALEARS - 2022

# ÍNDICE

# 1. Introducción

A fin de poder poner a prueba nuestros conocimientos sobre las técnicas de diseño, implementación y evaluación vistas a lo largo del curso, hemos desarrollado el siguiente proyecto. El objetivo es la implementación de un sistema empotrado distribuido de tiempo real. Se ha propuesto un caso práctico de la vida real en la que usaríamos sistemas empotrados, como es el de una alarma general de una casa.

Antes de nada, cabe recordar qué es un Sistema Empotrado de Tiempo Real (SETR), pues es un sistema hardware y software que controla otro sistema llamado planta, en el que está implantado. Estos sistemas SETR tienen tres fases de control, pues necesitan sensorizar datos mediante periféricos, controlar estos datos a través de algoritmos y con dichos resultados actuar sobre otros periféricos distintos. Que sean de tiempo real significa que el sistema debe responder oportunamente en el tiempo y de forma predecible a eventos que pueden ser impredecibles, es decir, que en cierta manera, deben cumplir unos deadlines. También es interesante recordar qué es un sistema distribuido. Estos sistemas son realmente conjuntos de otros sistemas que comparten sus respectivos estados y trabajan de manera conjunta para cumplir ciertos objetivos.

Durante la realización de la práctica, trabajaremos con un SOTR (sistema operativo de tiempo real), dispositivos de entrada y salida, mecanismos de sincronización y de exclusión mutua (semáforos, mailboxes, flags), ISR (timers y eventos externos) y finalmente, la comunicación entre dispositivos a través del bus CAN.

Durante la etapa de diseño, realizaremos un diagrama explicativo donde se mostrará la utilización de cada uno de los diferentes elementos descritos anteriormente, los bloques de datos y sus dependencias, el control de sincronización y de exclusión mutua y la comunicación entre tareas y placas.

Durante la etapa de implementación, emplearemos las librerías otorgadas para programar los elementos diseñados y sus funcionalidades en Arduino y compilaremos el programa enviándolo a través del puerto serie.

Finalmente, evaluaremos su funcionamiento mediante pruebas unitarias y de integración para corregir las irregularidades y realizar las modificaciones pertinentes del código fuente.

# 2. Descripción general del sistema

El sistema que vamos a modelar se basa en una **alarma general de una casa**. De esta manera, la alarma será multiuso y se activará en los siguientes modos:

- **Alarma de Incendios**: Mediante el sensor de temperatura, se detectará si hay una temperatura demasiado elevada, equivalente a un incendio en la casa.
- **Alarma de Calidad del Aire**: Mediante el potenciómetro (para nuestro caso será un ajuste de contaminación ambiental) podremos averiguar si se presentan unos valores demasiado cargados en el aire.
- **Alarma de Robo**: Mediante el sensor de Luz (para nuestro caso será un índice que nos indique si las puertas se han forzado) podremos saber si ha entrado alguien en la casa.

Estas tres alarmas funcionarán de manera continua, por lo que si alguna de ellas, o todas ellas superarán sus umbrales de activación, se activaría la alarma. Además, mediante el PAD numérico, se podrá elegir qué alarmas están activas y se podrán desactivar en caso de que hayan saltado.

La activación de la alarma supone lo siguiente.

- Zumbido ininterrumpido
- Encendido y apagado de luces LED
- Descripción de la alarma que ha saltado en el panel de 7 segmentos (número de alarma: 1- Incendio, 2- ICA, 3- Robo)
- Señalización de activación en el panel LCD

Para poder modelar la alarma, se usarán 2 placas HIB con sus respectivos sensores y actuadores. Para la primera placa, se usarán los sensores para modificar sus valores y los actuadores para indicar que la alarma está activa.
Para la segunda placa (que será el cerebro de la alarma) se podrán modificar los umbrales y será quien decida si la alarma se activa o no se activa, además de indicar por su panel LCD los valores de los sensores de la otra placa.

Para dejar un poco más claro el funcionamiento del sistema, a continuación explicaremos un posible caso funcional:

1. Encendido de las placas.
2. Se activa mediante el PAD de la Placa 1 la alarma de incendios.
3. La Placa 2 va leyendo los valores del sensor de temperatura y mostrandolos por el LCD.
4. El valor en un momento dado, supera un umbral establecido.
5. La Placa 2 avisa a la Placa 1 que se debe activar la alarma de incendios.
6. La Placa 1 empieza con el protocolo de activación (Zumbidos, Luces...).
7. Se desactiva manualmente mediante el PAD de la Placa 1 la alarma de Incendios, parando así el protocolo de activación.
8. Mediante la UART, se modifican los umbrales de las tres alarmas.
9. Se repite el proceso anterior.

A continuación, se puede visualizar un esquema general del sistema, para una mayor comprensión del caso.

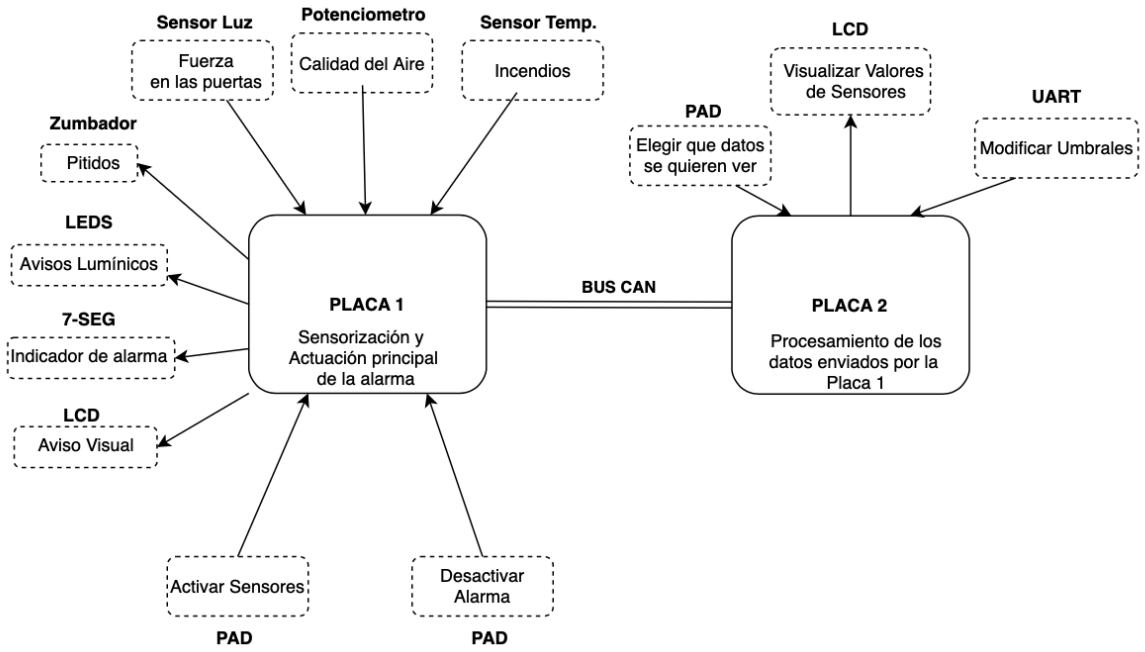


**FIG. 1.** Descripción general de la arquitectura del sistema

# 3. Descripción simplificada del sistema

Para describir el sistema de manera simple, nos centraremos en describir en lineas generales las diferentes tareas que existen en ambas placas. En el apartado 4, se procederá a entrar en detalle con cada una de estas tareas y como se ha implementado cada una de ellas.

## 3.1 Placa 1

En la placa 1, podemos distinguir las siguientes tareas:

- **E_CONTROL**: Gestión de la tecla dada por la activación del PAD.
- **P_SNS_LUZ / P_SNS_TEMP / P_SNS_POT**: Monitorización de los sensores.
- **P_CAN_TX**: Gestión del envío de las tramas CAN.
- **E_ACTIVAR_ALARMA**: Recepción del sensor que ha alcanzado el umbral por la ISR de CAN y activación de la alarma.
- **E_DESACTIVAR_ALARMA**: Desactivación de la alarma.

- **E_PRINT_P1**: Impresión por el LCD y visor de 7 segmentos del aviso de alarma y la ID del sensor que la ha encendido, además de una alerta visual por los LEDS.
- **P_ZUMBADOR**: Gestión del zumbador cuando la alarma está activa.

Distinguimos también las siguientes ISRs:
- **ISR PAD**: ISR del teclado que guarda el valor y despierta mediante flag a la tarea E_CONTROL.
- **ISR CAN**: ISR de recepción del bus CAN, que guarda la ID del sensor y despierta la tarea E_ACTIVAR_ALARMA mediante flag.

Cada una de las tareas de monitorización de los sensores cuenta con su propio semáforo de **sincronización** (sLuz, sTemp y sPot) que las bloquean o las liberan, parando o reanudando esta monitorización en función de la tecla pulsada en la tarea E_CONTROL.

Asimismo, para la transmisión a través del bus CAN en la tarea P_CANTX, contamos con un semáforo (sCAN), que asegura la **exclusión mutua** de lectura de txSensor por parte de P_CANTX y escritura por parte de las tareas de los sensores.

Una variable global (alarmaAct) indicará si la alarma está encendida o apagada. Al poder ser escrita y leída por diferentes tareas, también contará con su propio semáforo para asegurar esta **exclusión mutua** (sAlarma).

Finalmente, la tarea E_PRINT_P1 requerirá esperar al mailbox mbPrint para conocer la ID del sensor que ha generado la alarma.

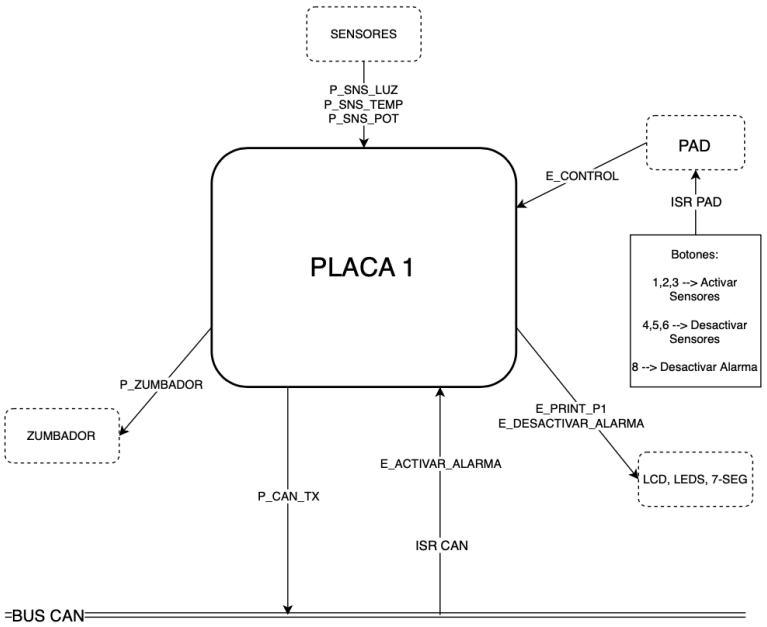


**FIG. 2.** Tareas P1

## 3.2 Placa 2

En la placa 2, podemos distinguir las siguientes tareas:

- **E_SNSCONTROL**: recepción de la ID del sensor y de su valor, comparación con el umbral y notificar si lo alcanza a través del bus CAN
- **P_CONTROLUART**: escritura en la UART de la información de los sensores en formato de tabla y lectura de entrada de usuario para modificación de los umbrales
- **P_LEERPAD**: lectura de tecla pulsada y modificación de sensor seleccionado
- **P_PRINT_P2**: mostrar información del sensor seleccionado en el LCD

Contamos con una ISR para la recepción desde el bus CAN ISR, la cual guarda la ID del sensor y despierta la tarea E_SNSCONTROL mediante el flag fCANevent.

Para comunicarse entre ellas, las tareas E_SNSCONTROL, P_CONTROLUART y P_PRINT_L2 necesitan modificar y leer unas variables globales (array) garantizando la exclusión mutua (sPrint2). Estas variables se corresponden con snsValores y snsUmbrales, que almacena información actualizada de los valores recibidos por los sensores y los umbrales de activación. Concretamente:

- **E_SNSCONTROL**: lee snsUmbrales y escribe en snsValores
- **P_CONTROLUART**: lee snsValores y snsUmbrales y escribe en snsUmbrales
- **P_PRINT_L2**: lee snsValores y snsUmbrales

Desde la tarea P_LEERPAD se modifica la variable global sensorSelect para que la pueda leer la tarea P_PRINT_L2 y así elegir la información del sensor a mostrar en el LCD, garantizando la exclusión mutua con sSensorSelect.

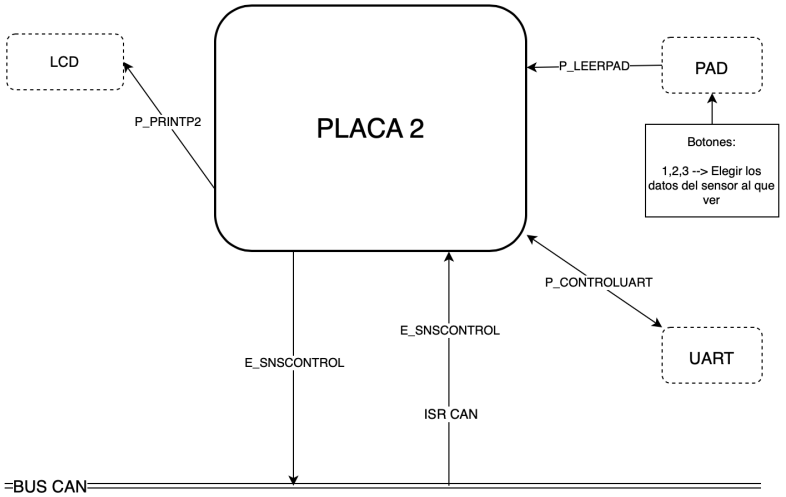


**FIG. 3.** Tareas P2

# 4. Decisiones de implementación

A continuación, se irán explicando las diferentes decisiones que se han tomado a la hora de la implementación de los programas para cada placa. Esta explicación también puede servir de referencia para saber como se han ido implementando dichos programas.

## 4.1 Placa 1

En primer lugar, la placa 1 usa un periodo de reloj de 50 ms y cuenta con 5 tareas periódicas y 4 tareas esporádicas.

Existe una tarea de Control (E_CONTROL) que se dedica a elegir que acción tiene que realizar el sistema en base a un input del usuario mediante el PAD. Este input se gestiona con una ISR (ISR_PAD) la cual examina mediante un keyHook que tecla se ha pulsado, adjuntándola a una variable global.
Se ha hecho de esta manera para evitar hacer polling, cosa que puede interferir con otras tareas periódicas. Una vez se tiene el valor del pad, se avisa a E_CONTROL mediante un flag que hay una tecla pulsada nueva.
Se ha elegido usar un flag porque creemos que es la mejor elección para “avisar” a una tarea que hay un valor disponible o que puede despertarse, pues creemos que un semáforo o un mailbox no serían las mejores opciones para este caso.

E_CONTROL puede activar (signal) o desactivar (wait) los semáforos de las tareas de sensorización. Estas tareas (P_SNS_”SENSOR”) usan semáforos para bloquearse o desbloquearse, pues sus respectivos semáforos, inicialmente a 0, necesitan el signal de E_CONTROL para poder ejecutar sus waits. El funcionamiento se puede resumir en que es la tarea de control la que libera a dichas tareas de sensorización.

Los sensores modifican sus valores en una variable global (txSensor) que es un array de 3 posiciones (una para cada valor de los sensores) y una vez modificado dicho valor, se duermen hasta la nueva activación. Para evitar que se recojan muchos datos a la vez (con el nextActivationTick puede que nos quedemos “atras en el tiempo” y se hagan muchas activaciones a la vez, debido a la implementación por semáforos), se calcula el nextActivationTick para que sea mayor o igual al tiempo del SO, de esta manera, para cada vez que se desbloquea la tarea, solo se obtiene 1 valor.

Para enviar por CAN, usamos la tarea P_CAN_TX, que, muy sofisticadamente, mira si hay envios pendientes comparando los valores del array de sensores en cada activación con los valores de la activación pasada. Sólo se envian los valores que han cambiado, para así evitar un tráfico de datos demasiado grande, que podría sobreescribir y causar confusión a las tareas de la Placa 2. Para evitar esto, también se ha reducido el periodo entre activaciones de la tarea P_CAN_TX, para

que sea menor que las tareas de sensorización (de esta manera, le da tiempo a comprobar si hay cambios en los valores, sin que estos cambien). Cabe decir que para acceder al array de valores, hay que acceder a una región crítica controlada por el semáforo sCAN.

Para poder activar la alarma, se recibe desde CAN, la ID del sensor que ha superado su umbral. Esto se hace mediante una ISR (ISR_CAN) que guarda en una variable global (rxSensor) dicha ID y activa un flag que despierta a la tarea E_ACTIVAR_ALARMA. Esta tarea es la responsable de poner a la variable global más importante de la placa a true (alarmaAct), protegida por el semáforo sAlarma, que significa que la alarma está activa. Una vez hecho esto, la tarea manda por mailbox (mbPrint1) la ID guardada en la variable global a la tarea E_PRINT_P1 que se despierta si hay algo en dicho mailbox.
Esta tarea, lee la ID que le sirve para indicar al 7-Segmentos que número debe mostrar. También activa el LCD con una alerta y enciende todos los LEDS. Se ha decidido usar mailbox porque creemos que es la primitiva que mejor funciona para dicho caso, pues debe enviar un valor a otra tarea en concreto.

Finalmente, la tarea E_DESACTIVAR_ALARMA, se activa pulsando el botón “8” del PAD, es entonces, una sub tarea de E_CONTROL. Esta tarea tiene como fin acceder a la variable global alarmaAct protegida por el semáforo sAlarma para ponerla a false e indicar así que la alarma no está activa. También se encarga de apagar los actuadores correspondientes.

Cabe mencionar finalmente la tarea P_ZUMBADOR que detecta si la alarma está activa accediendo a la región crítica de sAlarma y activa un pitido de 100ms a 500Hz, para después dormirse durante 500ms y volver a pitar. Este periodo se ha elegido para que no coincida ni moleste a ninguna de las otras tareas periódicas de sensorización o envío por CAN, por lo cual tiene un periodo de activación significativamente mayor.

## 4.2 Placa 2

Para la placa 2, se emplea un periodo de reloj de 50ms. Al contar con 3 tareas periódicas, 2 de ellas de interacción con el usuario, es necesario que la actualización sea rápida.

Las tareas de interacción con el usuario son P_CONTROLUART y P_LEERPAD. Tienen un periodo de 3 ticks de reloj (150ms) y prioridad 2 y 3 respectivamente. Al ejecutarse conjuntamente, dejarán un hueco de 1 tick que podrá ser ocupado sin problemas por cualquier otra tarea con periodo mayor, como es el caso de P_PRINT_L2.

La tarea esporádica E_SNSCONTROL de máxima prioridad se activará como mucho 3 veces cada 500 ms y será rápida, por lo que no hay que preocuparse de que pueda retrasar ninguna de las tareas anteriores.

En esta placa, usamos ISR cuando recibimos una trama por el bus CAN. Empleamos polling cuando leemos del teclado, porque al tratarse de un modelo más antiguo, no implementa interrupción. También hacemos polling para leer desde la UART los caracteres enviados del usuario.

## Umbrales y comprobación

Al recibir el mensaje desde el bus CAN y almacenarlo en una estructura de datos, contaremos con la ID del sensor y su valor. La siguiente tabla muestra el criterio de activación de la alarma:

| ID | Sensor | Umbral por defecto | Criterio de activación |
|---|---|---|---|
| 0 | Luminosidad | 600,0 | Valor < Umbral |
| 1 | Temperatura | 26,0 | Valor > Umbral |
| 2 | Calidad del aire | 2,5 | Valor > Umbral |

Con la ID, podemos indexar los arrays globales `snsValores` y `snsUmbrales` y determinar en función del criterio si activar o no la alarma.

## Formato de tabla de datos de la UART

Desde la terminal del usuario, mostraremos la información relativa a los tres sensores. El formato de la tabla a mostrar es el siguiente:

| ID | Sensor | Valor | Umbral | Superado |
|---|---|---|---|---|
| 0 | Luminosidad | 800,0 | 600,0 | * |
| 1 | Temperatura | 27,4 | 26,0 | * |
| 2 | Calidad del aire | 1,3 | 2,5 | |

Para imprimir la tabla, cada fila viene separada por un salto de línea y cada columna por caracteres tabuladores. Los elementos a imprimir vienen dados en el siguiente orden:

1. Cabecera
2. Línea separadora
3. Líneas formateadas
4. Retroacción de interacción del usuario

La línea por formatear es **"%d**\t**%s**\t**%s**\t**%c"**. Los valores y los umbrales hay que formatearlos previamente de *float* a *string* con la función `dtostrf` antes de hacerla pasar por la función `sprintf`.

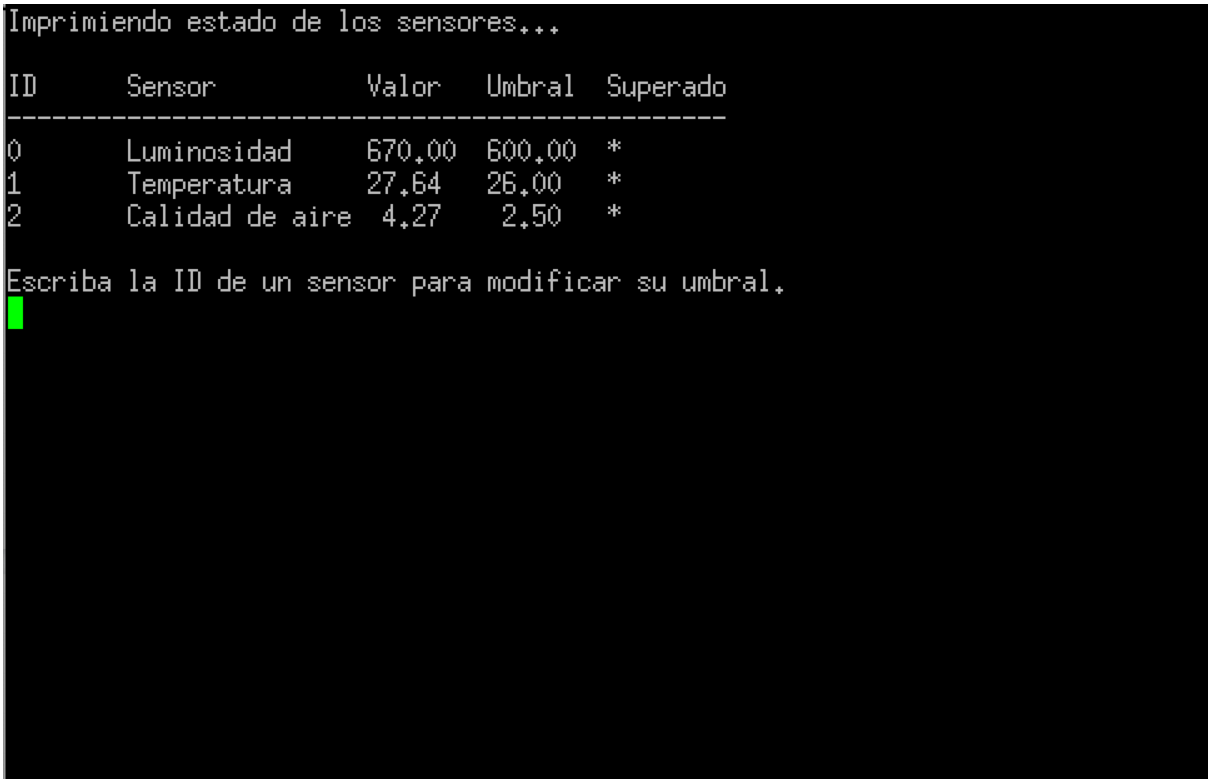


**FIG. 4.** Captura de Pantalla de la Uart, con umbrales superados.

## Modificación de umbral desde la UART

Para poder modificar el umbral, la UART también debe permitir la posibilidad de leer los caracteres introducidos por el usuario.

La interacción cuenta con 2 estados:

- **Selección de sensor:** se espera un valor 0, 1 ó 2 identificados en la tabla. Es el sensor cuyo umbral se quiere modificar. Al recibirse, cambia al estado siguiente.
- **Escritura de nuevo umbral:** se van recibiendo nuevos caracteres hasta salto de línea. Estos caracteres se acumulan en un string que luego se convertirá a float. Éste es el nuevo valor de umbral, el cual se modificará en la variable global asegurando la exclusión mutua. Al asignarse, se vacía el string y se regresa al estado anterior.

Al ser dos estados solamente, basta con utilizar una variable booleana (changing).

La retroacción se realizará de la siguiente manera:

| Estado | Texto a imprimir | Cambio de estado |
|---|---|---|
| `false` | `"Escriba la ID de un sensor para modificar su umbral."` | `{'0', '1', '2'}` |
| `true` | `"Nuevo valor: " + input` | `'\r'` |

El valor de `input` se irá modificando a medida que se introduzcan nuevos caracteres en el estado `true`. Al recibir retorno de carro, convertirá el valor a *float*, modificará el umbral y vaciará el *string*.

# 5. Tareas

A continuación se realiza una descripción más detallada de las tareas de cada placa. Para tener una visión general de todas las tareas al detalle, a continuación se expone un esquema general de cómo se conectan las tareas en ambas placas. (ver con calidad aquí).

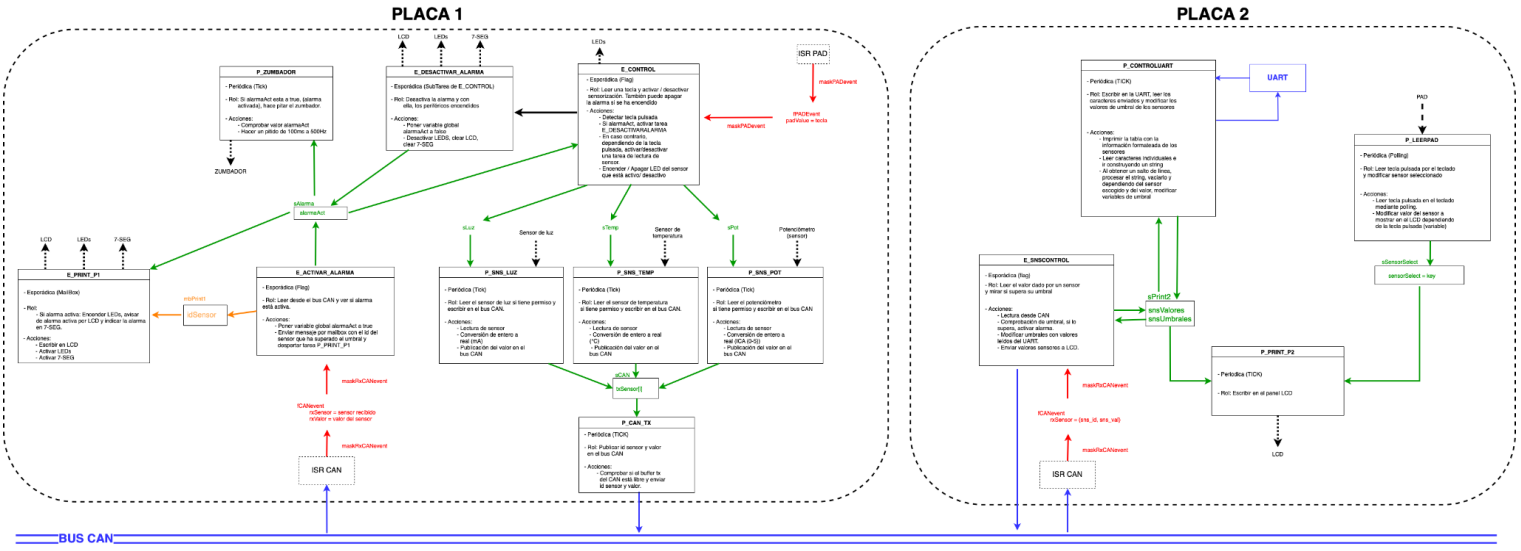


**FIG. 5.** Escquema general del funcionamiento del sistema.

Una vez vista esta imagen, podemos empezar a describir las tareas, una por una.

## 5.1 Placa 1

| E_CONTROL | |
|---|---|
| Tipo | Esporádica |
| Modo de activación | Flag |
| Activada por | ISR PAD |

| | |
|---|---|
| Periféricos | LEDs |
| Objetivo | Al activarse la ISR del keypad, esta activa el flag y guarda el valor de la tecla pulsada. Al despertar esta tarea, lee este valor y procesa la instrucción:<br><br>- **0, 1 y 2:** hace release de las tareas de sensorización (luz, temperatura y potenciómetro, respectivamente) y enciende el LED correspondiente, indicando que se ha encendido la monitorización de ese sensor.<br><br>- **3, 4 y 5:** hace wait de las tareas de sensorización y apaga los LEDs, indicando que se ha desactivado la monitorización.<br><br>- **7:** desactiva la alarma. |

| **P_SNS_LUZ / P_SNS_TEMP / P_SNS_POT** | |
|---|---|
| Tipo | Periódica |
| Modo de activación | Tick |
| Periodo | 1000 ms |
| Periféricos | Sensor de luz / temperatura / potenciómetro |
| Objetivo | Dependiendo de la tarea, leerá el valor medido del sensor de luz, de temperatura o potenciómetro y lo guardará en un array global de valores de sensores (asegurando exclusión mutua). |

| **P_CAN_TX** | |
|---|---|
| Tipo | Periódica |
| Modo de activación | Tick |
| Periodo | 100 ms |
| Periféricos | Bus CAN: escritura |
| Objetivo | Comprobará si los valores de los sensores se han modificado (array global), en cuyo caso indicará que dichos valores están pendientes de transmitir.<br><br>Posteriormente, se irán transmitiendo aquellos valores pendientes. Si el bus CAN aún está procesando un valor a enviar, se esperará a la |

| | |
|---|---|
| | siguiente iteración de la tarea para volver a intentarlo. |

| **E_ACTIVAR_ALARMA** | |
|---|---|
| Tipo | Esporádica |
| Modo de activación | Flag |
| Activada por | ISR CAN |
| Periféricos | - |
| Objetivo | Al activarse la ISR del CAN, esta activa el flag y guarda el valor de la ID del sensor que debe activar la alarma. Esta ID luego se hace pasar por un MailBox a la tarea E_PRINT_P1 y se modifica la variable global alarmaAct, garantizando exclusión mutua. |

| **E_DESACTIVAR_ALARMA** | |
|---|---|
| Tipo | Esporádica |
| Modo de activación | Llamada a función desde otra tarea |
| Activada por | E_CONTROL |
| Periféricos | LEDs, LCD, 7-SEG |
| Objetivo | Asigna el valor falso a alarmaAct (garantizando exclusión mutua), desactiva los LEDs y hace clear del LCD y del 7-SEG. |

| **E_PRINT_P1** | |
|---|---|
| Tipo | Esporádica |
| Modo de activación | MailBox |
| Activada por | E_ACTIVARALARMA |
| Periféricos | LEDs, LCD, 7-SEG |
| Objetivo | Encender LEDs, avisar de alarma activa por LCD e indicar la alarma en 7-SEG con el valor recibido del MailBox. |

| **P_ZUMBADOR** |
|---|

| | |
|---|---|
| Tipo | Periódica |
| Modo de activación | Tick |
| Periodo | 500 ms |
| Periféricos | Zumbador |
| Objetivo | Sólo si la variable global alarmaAct es verdadera, hará sonar el zumbador a una frecuencia de 500 Hz durante un periodo de 100 ms. |

## 5.2 Placa 2

| **E_SNSCONTROL** | |
|---|---|
| Tipo | Esporádica |
| Modo de activación | Flag |
| Activada por | ISR CAN |
| Periféricos | - |
| Objetivo | Al activarse la ISR del CAN, esta activa el flag y guarda la ID del sensor y su valor en una estructura para que al despertar esta tarea, pueda comprobar si supera cierto umbral.<br><br>También guarda el valor del sensor recibido en función de su ID en una variable global (array de valores) asegurando exclusión mutua.<br><br>En caso de superarlo, reenvía por el bus CAN la ID del sensor que lo ha generado. |

| **P_CONTROLUART** | |
|---|---|
| Tipo | Periódica |
| Modo de activación | Tick |
| Periodo | 150 ms |
| Periféricos | UART: lectura y escritura |
| Objetivo | Lee los valores de los sensores y sus umbrales, asegurando exclusión mutua. Formatea estos valores en forma de tabla para luego imprimirla vía UART. |

| | |
|---|---|
| | A su vez, gestiona la recepción de caracteres por la UART. Se distinguen dos estados:<br>- **Selección de sensor:** se espera un valor 0, 1 ó 2 identificados en la tabla. Es el sensor cuyo umbral se quiere modificar. Al recibirse, cambia al estado siguiente.<br>- **Escritura de nuevo umbral:** se van recibiendo nuevos caracteres hasta salto de línea. Estos caracteres se acumulan en un string que luego se convertirá a float. Éste es el nuevo valor de umbral, el cual se modificará en la variable global asegurando la exclusión mutua. Al asignarse, se vacía el string y se regresa al estado anterior. |

| **P_PRINT_P2** | |
|---|---|
| Tipo | Periódica |
| Modo de activación | Tick |
| Periodo | 500 ms |
| Periféricos | LCD |
| Objetivo | Lee dos variables globales: la ID del sensor seleccionado y el valor apuntado por ese ID en el array de valores y umbrales.<br><br>Una vez obtenidos estos valores, formatea los strings que imprimirá en el LCD. El resultado será el valor y el umbral del sensor seleccionado mostrado por el LCD. |

| **P_LEERPAD** | |
|---|---|
| Tipo | Periódica |
| Modo de activación | Tick |
| Periodo | 150 ms |
| Periféricos | Keypad |
| Objetivo | Realiza polling del teclado, debido a que se trata de una placa antigua y no se pueden obtener las pulsaciones mediante interrupciones.<br><br>Lee y comprueba si es la tecla 0, 1 ó 2. En caso afirmativo, escribe la variable global de sensor |

| | |
|---|---|
| | seleccionado (para mostrar por el LCD), garantizando la exclusión mutua. |

# 7. Tramas CAN

Las placas se comunican a través del bus CAN. Podemos diferenciar dos tipos de tramas:

| **Trama: valor del sensor** | |
|---|---|
| Mensajero | Placa 1, procesada por tarea P_CANTX |
| Destinatario | Placa 2, procesada por ISR isrCAN. |
| **Mensaje** | |
| Contenido | ID del sensor y valor |
| Tamaño | 5 bytes |
| Formato | `struct canMsg_t {`<br>`  uint8_t sns_id;`<br>`  float sns_val;`<br>`};` |
| **Objetivo** | |
| Comunicar a la placa 2 el nuevo valor monitorizado de un sensor específico para su procesamiento. | |

| **Trama: ID del sensor** | |
|---|---|
| Mensajero | Placa 2, procesada por tarea E_SNSCONTROL |
| Destinatario | Placa 1, procesada por ISR isrCAN. |
| **Mensaje** | |
| Contenido | ID del sensor |
| Tamaño | 1 byte |
| Formato | `uint8_t` |
| **Objetivo** | |

| |
|---|
| Comunicar a la placa 1 que uno de los sensores ha entrado dentro del umbral e indicar de cuál de los sensores se trata para activar la alarma. |

En el caso de la trama que envía el valor del sensor, se planteó inicialmente enviar un array con los valores de los tres sensores. No obstante, el tamaño de los datos era de 12 bytes (3 floats), lo cual es mayor que el máximo tamaño permitido por las tramas CAN (8 bytes), por lo que finalmente se optó por incluir la ID del sensor que generó dicho valor en una estructura y enviar los valores de los tres sensores por separado.

# 8. Pruebas

## 8.1 Pruebas de implementación

Durante la implementación, se han ido haciendo muchísimas pruebas, pues la estrategia que se ha seguido ha sido ir probando cada función con la mayor casuística posible y el mayor número de veces, para poder tener claro si las funciones iban a fallar o no.

En muchos casos, para saber si una función fallaba o no, se ha tenido que recurrir al debugging mediante prints en el puerto serie. De hecho, esta técnica nos ha ayudado mucho para resolver problemas, pues mediante estos prints hemos podido encontrar los motivos que nos llevaban a errores y así poder solucionarlos.

Para la placa 1, como gran parte del código depende de los resultados que le pase la placa 2 por CAN (todo lo relacionado con la activación de la alarma), se ha implementado una función auxiliar que activaba la alarma mediante el botón “0” del PAD. De esta manera, se podía comprobar si se encendían los actuadores y si se actualizaba correctamente la variable global alarmaAct, entre otras cosas.

Cabe decir que en la placa 1 el problema de implementación más notorio que hemos tenido ha sido relacionado con el número máximo de tareas y semáforos estipulado en la librería HIB, pues no habíamos tenido en cuenta estos valores y había muchas tareas que fallaban por motivos realmente inexplicables, por lo que finalmente descubrimos el fallo y aumentamos tanto el número de tareas como el número de semáforos máximos.

En la placa 2, se programó una tarea auxiliar P_AUXILIAR periódica la cual modificaba los valores de los sensores del array snsValor aleatoriamente. De esta forma, se podía simular el funcionamiento de la placa 1 sin necesitar que ambas placas estuvieran conectadas a través del bus CAN. Se utilizó inicialmente para ver si los valores se mostraban y se actualizaban correctamente en la UART y el LCD.

## 8.2 Pruebas de validación

Para las pruebas de validación, se han conectado ambas placas mediante CAN para hacer un testeo a fondo de todas las posibles casuísticas que se podían dar en el sistema.

Primeramente se testearon las condiciones óptimas de funcionamiento, esto es, activar la sensorización, hacer superar un umbral y que se active la alarma, para posteriormente desactivarla. Una vez comprobado que funcionaba para cada sensor, se empezaron a realizar pruebas más exhaustas, como modificar los umbrales de activación, activar múltiples sensores a la vez y activar la alarma desde varios sensores diferentes.

Una vez superados dichos test que verifican el correcto funcionamiento del programa en situaciones complejas, podemos dar por válido el programa y, como consecuencia el SETR.

Cabe decir que suponemos que existen circunstancias que no se han probado y el programa podría dar errores en condiciones de uso extremas, aunque serían siempre casos excepcionales que no consideramos de uso convencional.

# 9. Conclusiones

Durante la realización de esta práctica hemos puesto a prueba nuestros conocimientos sobre las técnicas de diseño, implementación y evaluación vistas a lo largo del curso. El proyecto ha consistido en la implementación de un sistema empotrado distribuido de tiempo real. Se ha propuesto un caso práctico de la vida real en la que usaríamos sistemas empotrados, como es el de una alarma general de una casa.

Como resultado de la realización de esta práctica, hemos obtenido un diagrama explicativo donde se muestran los bloques de datos y sus dependencias, las primitivas de control de sincronización y de exclusión mutua y la comunicación entre tareas y placas. A su vez, en la implementación, hemos obtenido dos programas escritos en C++ para cada una de nuestras placas.

Los objetivos propuestos se han cumplido sobradamente. Las placas funcionan correctamente sin bloquearse ni colapsar el sistema y se comunican apropiadamente. Todas las tareas propuestas inicialmente en el diagrama han podido ser implementadas sin la necesidad de realizar modificaciones demasiado significativas en el modelo conceptual del proyecto.

Las diferentes maneras que hemos encontrado de implementar las primitivas de sincronización de tareas nos ha mostrado una solución para las tareas de sensorización bastante curiosa. La utilización de semáforos para bloquear y

desbloquear tareas desde la tarea de control ha sido sorprendentemente eficiente.  A pesar de ello, también es cierto que hemos tenido algunos problemas de bloqueos de tareas inesperados por no realizar los *signals* apropiadamente.

Concretamente, hemos aprendido a programar SETRs y a comunicarnos a través del protocolo CAN. Para cada SETR, hemos aprendido a identificar las tareas concurrentes que deben implementar las funcionalidades de nuestro caso práctico y hemos aprendido a sincronizarlas y a comunicarlas entre ellas con primitivas de control de concurrencia y variables globales.

La capacidad que tienen los sensores de operar de manera independiente es un punto fuerte de nuestro programa. Al mismo tiempo, el procesamiento del envío de los valores monitorizados por los sensores a través del bus CANes muy efectiva, pues solo procesa aquellos valores pendientes de enviar, y los reserva para enviarlos en una futura iteración en caso de que el bus se encuentre ocupado.

Una posible mejora de nuestro programa pasaría por escribir de manera más eficiente en la UART, pues tenemos la posibilidad de mover el cursor a cualquier lugar del terminal y reescribir porciones de la pantalla. Con esto, podríamos haber modificado solamente la información de la tabla que se requería actualizar en lugar de tener que reescribir toda la tabla en cada iteración.

Al mismo tiempo, la lectura de caracteres enviados por la UART es bastante sensible. Cualquier cadena de caracteres no deseada podría llegar a inhabilitar su correcta utilización y modificar los valores de los umbrales de manera incorrecta.

Otra forma de mejorar el programa podría pasar por juntar algunas tareas en la placa 1 (E_PRINT_P1, E_ACTIVARALARMA y E_DESACTIVARALARMA) y por separar algunas en la placa 2 (P_CONTROLUART). Si hubiéramos contado con más tiempo para realizar la práctica, quizá podríamos haber reflexionado mejor las prioridades de las tareas, sus periodos y el periodo del reloj, ajustándose mejor a los requisitos de nuestro proyecto.