Curso 2022-2023

Universitat de les Illes Balears

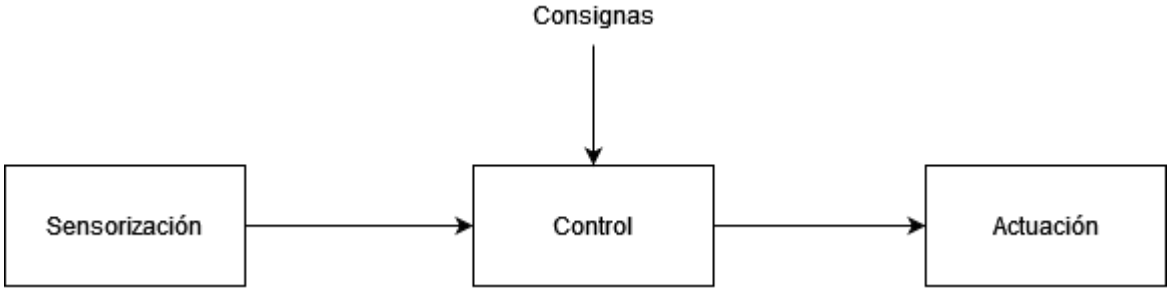
Sistemas Empotrados

Diseño e implementación de un horno inteligente

Autores
Martín Ignacio Rizzo
Luis Carlos Montes Sabariego

1. Introducción, motivación e idea general

Los sistemas empotrados son sistemas informáticos cuyo principal objetivo es realizar ciertas tareas especializadas, normalmente en tiempo real. Este tipo de sistemas se componen, principalmente, por un microcontrolador y uno o varios sensores y actuadores. El flujo de trabajo a alto nivel es sencillo de entender: se recoge información sobre el estado del sistema desde los sensores, las tareas dedicadas a control procesan esta información, teniendo en cuenta unas consignas, y comunican a los actuadores qué deben hacer.



Los sistemas empotrados se encuentran en gran cantidad de dispositivos que utilizamos diariamente: lavavajillas, lavadoras, coches, etc. Por esto, este tipo de sistemas han ido adquiriendo mucha importancia (y cada vez más con nuevas tecnologías como el IoT) durante los últimos años. En consecuencia, resulta de gran importancia su estudio en asignaturas como esta. Para poder profundizar correctamente y de forma práctica en los sistemas empotrados y todo lo aprendido en la asignatura, como programación de microcontroladores y de aplicaciones distribuidas, diseño de sistemas empotrados y concurrencia, entre otros, se ha realizado esta práctica, consistente en el diseño e implementación de un horno inteligente.

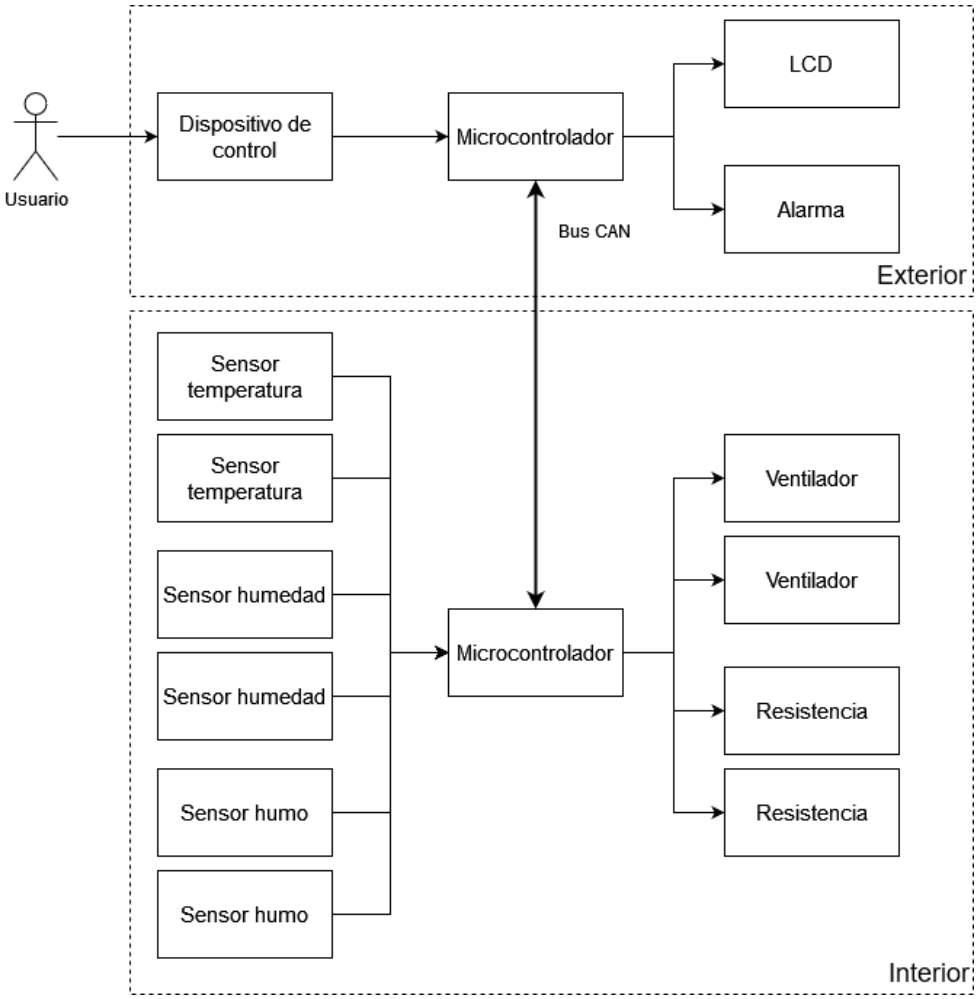
La idea del horno inteligente es que el usuario puede seleccionar una receta de entre un conjunto de estas, introducir la comida sin hacer dentro del horno y será este último el que se encargue de regular la temperatura de forma automática, sin que el usuario deba interactuar con él hasta que acabe el proceso de cocinado.

2. Descripción del sistema y simplificaciones

Como se ha dicho antes, la idea principal del horno inteligente es permitir que el usuario pueda simplemente elegir una receta y dejar que el horno se encargue de tomar las decisiones de forma automática de cómo cocinar la comida. En un sistema real, este horno estaría compuesto por los siguientes elementos:

- Uno o varios microcontroladores
- Uno o varios sensores de temperatura
- Uno o varios sensores de humedad
- Uno o varios sensores de humo
- Uno o varios actuadores para la temperatura, que consistirían principalmente en resistencias
- Uno o varios actuadores para el humo/humedad, que podrían ser ventiladores

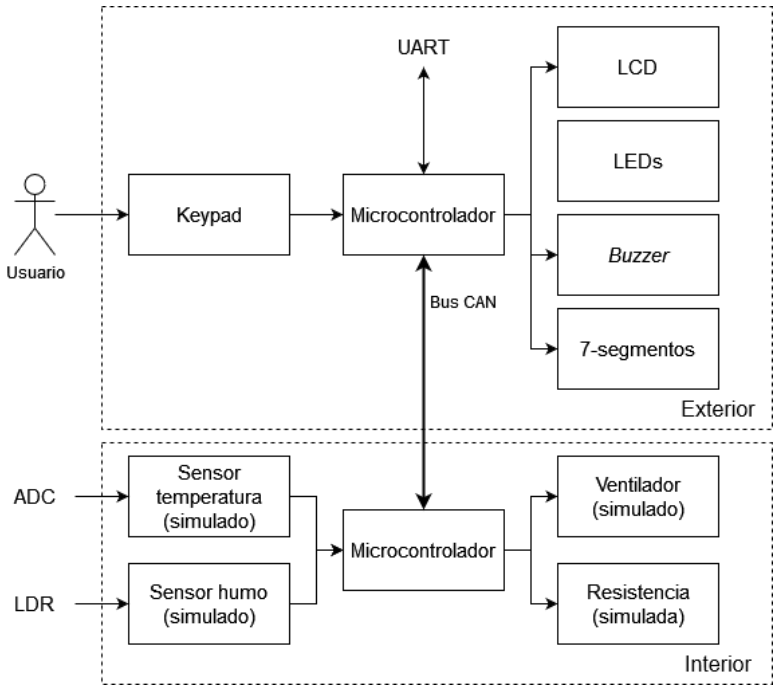
Todos estos elementos sirven simplemente para llevar a cabo correctamente la tarea de cocina. Pero, además, el horno debería tener una pantalla (por ejemplo, LCD) que mostrara al usuario los menús con diversas recetas y sus opciones, algún tipo de control para poder navegar los menús, alarmas para avisar de posibles problemas o que la comida ha finalizado, etc. Una posible arquitectura, muy sencilla, para este sistema se muestra en la siguiente figura.



En esta arquitectura, todos los sensores y actuadores están duplicados. Esto podría ser con el objetivo de redundancia, para proporcionar más fiabilidad y robustez al sistema. También hay dos microcontroladores: uno es el encargado de la sensorización y actuación en el interior del horno, y el otro se encarga de la interacción del usuario y del envío de consignas al microcontrolador interior. Cabe destacar que el uso de las palabras interior y exterior en la figura no se deben tomar de forma literal, pues todos los dispositivos se encontrarán, evidentemente, dentro del horno. En cambio, se utiliza esta nomenclatura para separar la interacción del horno con el usuario de la interacción del horno con el propio horno y sus componentes. Como se puede observar, los microcontroladores deben estar conectados y, en este caso, se ha elegido que esten conectados por CAN.

Este sistema podría hacerse más complejo y sofisticado (por ejemplo, añadiendo más sensorización e implementar algoritmos complejos para obtener mejores datos), pero, a priori, sería posible implementarlo de forma real y poder conseguir un producto decente. Sin embargo, debido a limitaciones obvias tanto en presupuesto como en material y tiempo, este sistema no es viable para nuestro caso. Estas limitaciones son evidentes: la placa que tenemos a mano dispone de varios periféricos, pero en ningún caso dispone de ventiladores ni resistencias, ni sensores de humo ni de humedad, además de que este proyecto podría tardar varios meses en diseñarse e implementarse, tiempo del cual no disponemos. Por esto, el sistema se debe simplificar y realizar algunas simulaciones para poder llevar a cabo su diseño e implementación.

La principal simplificación se encuentra en la parte interior, que se corresponde con la sensorización y actuación: el sistema dispondrá únicamente de un sensor de temperatura y un sensor de humo. De la misma forma, dispondrá de un ventilador y una resistencia. Todo esto será simulado, tanto la sensorización como la actuación. Esta simulación se llevará a cabo con la ayuda del ADC y los LDRs de la placa. En la parte exterior, la arquitectura es bastante parecida, pues utilizaremos el LCD y el *buzzer* incluido en la placa, además de los 7-segmentos y los LEDs. Además, el dispositivo de control será el *keypad*. Finalmente, el microcontrolador exterior se comunicará a través de UART con el PC. Con todo esto, la arquitectura de alto nivel del sistema simplificado quedaría de la siguiente forma:

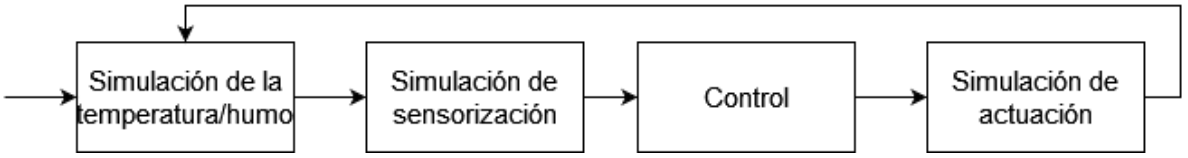


3. Arquitectura y diseño de las tareas

Con las simplificaciones necesarias hechas y creada la arquitectura de alto nivel del sistema, se puede proceder a diseñar la arquitectura de las tareas dentro de cada microcontrolador. La principal decisión de diseño de las tareas de las placas es la de crear una arquitectura maestro-esclavo. El microcontrolador exterior tendrá una tarea de control principal, la cual será el maestro, y el microcontrolador interior tendrá dos tareas de control, una para la temperatura y otra para el humo, que serán las tareas esclavas. La tarea de control exterior recibirá el *input* del usuario y, en base a este *input*, obtendrá las consignas necesarias y se las enviará a las tareas del microcontrolador interior. Este último recibirá estas consignas y se encargará específicamente de que se cumplan, llevando a cabo las tareas de sensorización y actuación necesarias para este fin. El microcontrolador exterior también recibirá información del microcontrolador interior, como, por ejemplo, si hay un fuego dentro del horno o la temperatura actual.

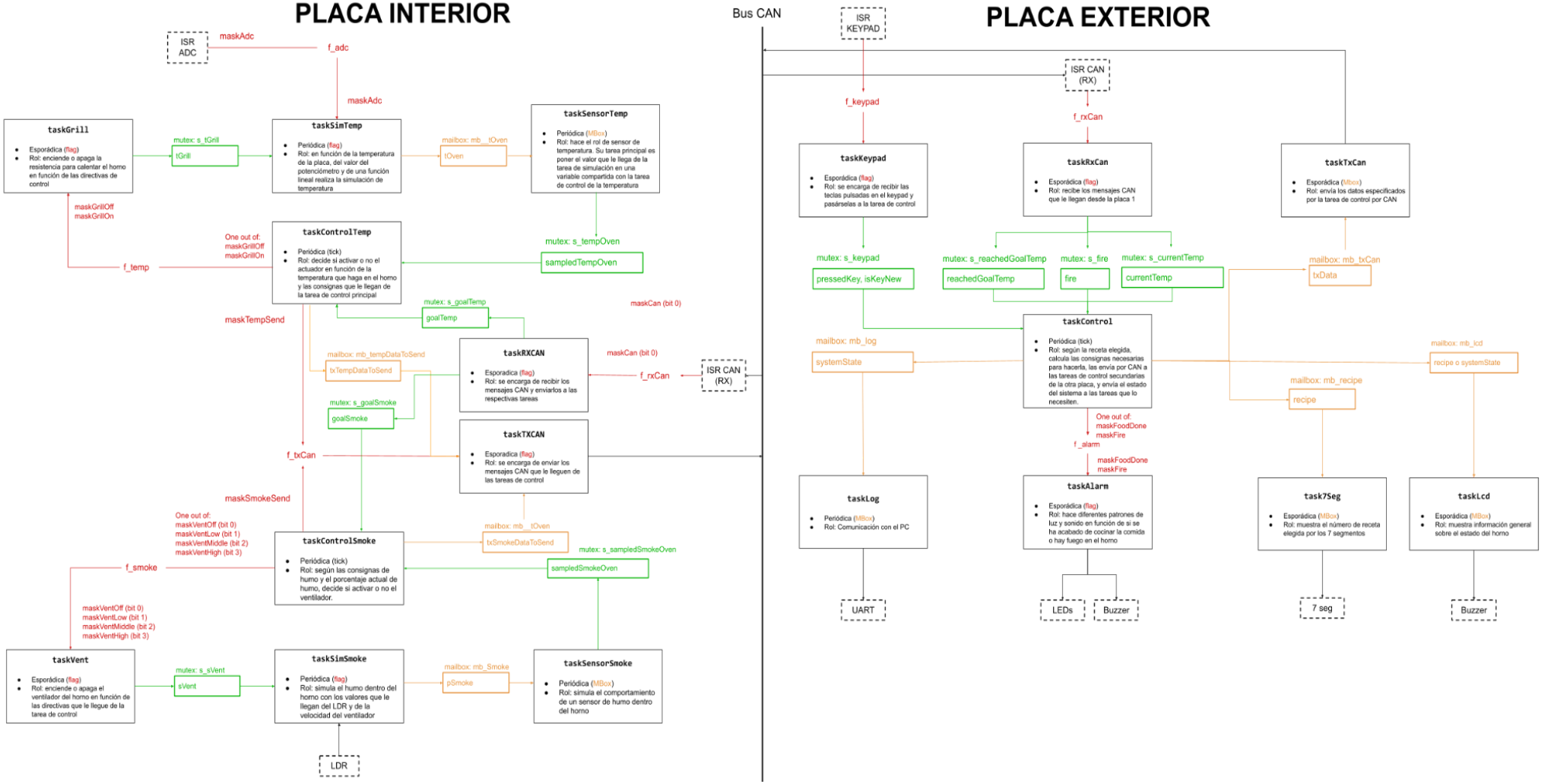
Las tareas del microcontrolador exterior también seguirán esta arquitectura maestro-esclavo, donde la tarea de control vuelve a ser el maestro. Todas las demás tareas serán las esclavas, y funcionarán de forma que la tarea de control pueda recibir los datos necesarios para su correcta operación y para que pueda mostrar por los periféricos información sobre el sistema. En concreto, deberá existir una tarea por cada uno de los periféricos que se quieran manipular, otra para utilizar el UART y dos tareas para recibir y enviar por CAN.

Las tareas del microcontrolador interior se encargarán de la simulación de sensorización y actuación. En particular, seguirán el siguiente flujo de operación:



Esto se realizará tanto para la temperatura como para el humo, de forma que se tendrán dos conjuntos de tareas “hermanos”. Se necesitará una tarea para cada una de las fases del diagrama, dando lugar a un total de ocho tareas para la simulación de sensorización y actuación. Además, se deben crear dos tareas para recibir y enviar por CAN.

Todo esto da lugar a ocho tareas en la placa exterior y diez en la placa interior. Todas estas tareas necesitarán estar sincronizadas entre ellas y, para esto, disponemos de tres herramientas de sincronización principales: semáforos, *mailboxes* y *flags*. Para poder explicar las principales decisiones de sincronización y comunicación entre las tareas, además de las propias tareas, primero se mostrará la arquitectura final de las estas y se procederá a su explicación.


PLACA INTERIOR
PLACA EXTERIOR
Bus CAN
ISR ADC
maskAdc
f_adc
maskAdc
taskGrill
Esporádica (flag)
Rol: enciende o apaga la resistencia para calentar el horno en función de las directivas de control
mutex: s_tGrill
tGrill
taskSimTemp
Periódica (flag)
Rol: en función de la temperatura de la placa, del valor del potenciómetro y de una función lineal realiza la simulación de temperatura
mailbox: mb__tOven
tOven
taskSensorTemp
Periódica (MBox)
Rol: hace el rol de sensor de temperatura. Su tarea principal es poner el valor que le llega de la tarea de simulación en una variable compartida con la tarea de control de la temperatura
maskGrillOff
maskGrillOn
f_temp
One out of:
maskGrillOff
maskGrillOn
taskControlTemp
Periódica (tick)
Rol: decide si activar o no el actuador en función de la temperatura que haga en el horno y las consignas que le llegan de la tarea de control principal
mutex: s_tempOven
sampledTempOven
mutex: s_goalTemp
goalTemp
maskTempSend
maskCan (bit 0)
mailbox: mb_tempDataToSend
txTempDataToSend
taskRXCAN
Esporadica (flag)
Rol: se encarga de recibir los mensajes CAN y enviarlos a las respectivas tareas
maskCan (bit 0)
f_rxCan
ISR CAN (RX)
mutex: s_goalSmoke
goalSmoke
taskTXCAN
Esporadica (flag)
Rol: se encarga de enviar los mensajes CAN que le lleguen de las tareas de control
f_txCan
maskSmokeSend
One out of:
maskVentOff (bit 0)
maskVentLow (bit 1)
maskVentMiddle (bit 2)
maskVentHigh (bit 3)
taskControlSmoke
Periódica (tick)
Rol: según las consignas de humo y el porcentaje actual de humo, decide si activar o no el ventilador.
mailbox: mb__tOven
txSmokeDataToSend
mutex: s_sampledSmokeOven
sampledSmokeOven
f_smoke
maskVentOff (bit 0)
maskVentLow (bit 1)
maskVentMiddle (bit 2)
maskVentHigh (bit 3)
taskVent
Esporádica (flag)
Rol: enciende o apaga el ventilador del horno en función de las directivas que le llegue de la tarea de control
mutex: s_sVent
sVent
taskSimSmoke
Periódica (flag)
Rol: simula el humo dentro del horno con los valores que le llegan del LDR y de la velocidad del ventilador
mailbox: mb_Smoke
pSmoke
taskSensorSmoke
Periódica (MBox)
Rol: simula el comportamiento de un sensor de humo dentro del horno
LDR
ISR KEYPAD
ISR CAN (RX)
f_keypad
f_rxCan
taskKeypad
Esporádica (flag)
Rol: se encarga de recibir las teclas pulsadas en el keypad y pasárselas a la tarea de control
taskRxCan
Esporádica (flag)
Rol: recibe los mensajes CAN que le llegan desde la placa 1
taskTxCan
Esporádica (Mbox)
Rol: envía los datos especificados por la tarea de control por CAN
mutex: s_keypad
pressedKey, isKeyNew
mutex: s_reachedGoalTemp
reachedGoalTemp
mutex: s_fire
fire
mutex: s_currentTemp
currentTemp
mailbox: mb_txCan
txData
taskControl
Periódica (tick)
Rol: según la receta elegida, calcula las consignas necesarias para hacerla, las envía por CAN a las tareas de control secundarias de la otra placa, y envía el estado del sistema a las tareas que lo necesiten.
mailbox: mb_log
systemState
mailbox: mb_recipe
recipe
mailbox: mb_lcd
recipe o systemState
One out of:
maskFoodDone
maskFire
f_alarm
maskFoodDone
maskFire
taskLog
Periódica (MBox)
Rol: Comunicación con el PC
taskAlarm
Esporádica (flag)
Rol: hace diferentes patrones de luz y sonido en función de si se ha acabado de cocinar la comida o hay fuego en el horno
task7Seg
Esporádica (MBox)
Rol: muestra el número de receta elegida por los 7 segmentos
taskLcd
Esporádica (MBox)
Rol: muestra información general sobre el estado del horno
UART
LEDs
Buzzer
7 seg
Buzzer

3.1. Placa interior

Como ya se ha dicho anteriormente, la placa interior contiene dos grupos de tareas "hermanos", que se ocupan de la simulación de la sensorización y actuación de la temperatura y el humo. Además, tiene una tarea para la recepción de los mensajes CAN y otra para enviarlos.

taskSimTemp

- Periódica por *flag*
- El *flag* lo activa la ISR del ADC conectado al potenciómetro
- Dependencias:
    - tGrill

Se encarga de hacer la simulación de la temperatura que hace dentro del horno. Para esto, utiliza una función muy sencilla, que es la siguiente:

$$tOven = slope + tOven$$

Donde $tOven$ representa la temperatura del horno y $slope$ es la "pendiente" de la función. En concreto, $slope$ tendrá un valor entre 1 y 10, los cuales vendrán dados por el mapeo del valor actual del potenciómetro. El valor del $slope$ será positivo si la resistencia, que viene dada por tGrill, está encendida, y será negativa si está apagada. Esto representa como la temperatura sube si la resistencia está encendida o baja de lo contrario

taskSensorTemp

- Periódica por *mailbox*
- El *mailbox* lo activa **taskSimTemp** para enviarle la temperatura simulada
- Dependencias:
    - El valor que envíe **taskSimTemp** por el *mailbox* mb_tOven

Simula un sensor dentro del horno. El único objetivo de esta tarea es poner el valor que llegue por el *mailbox* mb_tOven en la variable global sampledTempOven.

taskControlTemp

- Periódica por tick
- Dependencias:
    - sampledTempOven
    - goalTemp

Es la tarea que se encarga de tomar la decisión de si la resistencia se debería encender o apagarse. Esto lo hace en base a sampledTempOven, que es la temperatura actual en el interior del horno, y goalTemp, que es la consigna de temperatura. Para conseguir el objetivo de mantener la temperatura de la consigna, calcula una histéresis en base a esta. Si sampledTempOven está por debajo de la histéresis inferior, mantiene la resistencia encendida. En cambio, si está por encima de la histéresis superior, la apaga.

Además, se encarga de enviar los *logs* de la temperatura actual del horno por CAN a la placa exterior.

taskGrill

- Esporádica, activada por *flag*
- El *flag* lo activa **taskControlTemp**, utilizando las máscaras maskGrillOff y maskGrillOn
- Dependencias: ninguna

Simula el comportamiento de una resistencia del horno. En función de la máscara que le llegue, se apaga o se enciende. Es decir, si le llega maskGrillOff, es porque se tiene que apagar, y pone el valor correspondiente (TGRILL_OFF en el código) en la variable tGrill. En cambio, si le llega maskGrillOn, es porque se tiene que encender, y pone el valor correspondiente (TGRILL_ON en el código) en la variable tGrill.

taskSimSmoke

- Periódica por tick
- Dependencias:
    - sVent
    - Valor ADC LDR derecho

Se encarga de hacer la simulación del humo dentro del horno. Para esto, utiliza principalmente el valor del ADC del LDR derecho, el cual mapea a un valor entre 0 y 1. Esto en el caso en el que sVent, que representa el ventilador del horno, está apagado. Si está encendido, simplemente lo que hace es restar cierta cantidad al valor actual del humo hasta llegar a 0. Una vez calculado este valor, se lo envía por el *mailbox* mb_smoke a **taskSensorSmoke**.

taskSensorSmoke

- Periódica por *mailbox*
- El *mailbox* lo activa **taskSimSmoke**, con el porcentaje actual de humo en el horno
- Dependencias:
    - Valor del *mailbox* mb_smoke.

De la misma forma que la tarea de sensorización de la temperatura, simplemente pone el valor que le llegue en mb_smoke en sampledSmokeOven.

taskControlSmoke

- Periódica por tick
- Dependencias:
    - sampledSmokeOven

Se encarga de controlar el porcentaje de humo del horno. Para esto, tiene un valor máximo y un valor mínimo de humo permitido. En caso de que el porcentaje de humo supere el valor máximo, se considerará que se está quemando la comida y encenderá el ventilador y enviará por CAN el mensaje de que hay fuego a la placa exterior. Una vez que el humo esté por debajo del valor mínimo, apagará el ventilador y volverá al estado normal.

taskVent

- Esporádica, activada por *flag*
- El *flag* lo activa **taskControlSmoke**, utilizando las máscaras maskVentOff y maskVentOn
- Dependencias: ninguna

Simula el ventilador del horno. Si le llega el *flag* con maskVentOff, se apaga, poniendo la variable sVent a false. Si le llega el *flag* con maskVentOn, se enciende, poniendo la variable sVent a true.

taskRxCan

- Esporádica por *flag*
- El *flag* lo activa la ISR de CAN
- Dependencias: ninguna

Se encarga de recibir y leer los mensajes que lleguen por CAN. Comprobará el identificador del mensaje y actuará en función de este. Por ejemplo, si el mensaje que llega es el de la consigna de temperatura, pondrá el valor del campo de datos en la variable goalTemp.

taskTxCan

- Esporádica por flag
- El *flag* lo activan las tareas **taskControlTemp** y **taskControlSmoke**
- Dependencias:
    - Valores de los *mailboxes* mb_tempDataToSend y mb_smokeDataToSend.

Envía por CAN los valores que le llegues por mb_tempDataToSend y mb_smokeDataToSend. El valor que habrá dentro de estos *mailboxes* será un struct llamado TxData, que contendrá el identificador del mensaje y los datos a enviar.

3.2. Placa exterior

La placa exterior es la encargada de la interacción con el usuario y de enviar las consignas y monitorear la placa interior.

taskTxCan

- Esporádica por *mailbox*
- El *mailbox* lo activa **taskControl**
- Dependencias:
    - Valor delos *mailbox* mb_txCan.

Ídem a **taskTxCan** de la placa interior.

taskRxCan

- Esporádica por *flag*
- El *flag* lo activa la ISR de CAN
- Dependencias: ninguna

Ídem a **taskRxCan** de la placa interior.

taskKeypad

- Esporádica por *flag*
- El *flag* lo activa la ISR del *keypad*
- Dependencias: ninguna

Recibe las teclas pulsadas por el usuario en el *keypad* y se la pasa a la tarea de control, poniendo las variables pressedKey al valor de la tecla e isKeyNew a true.

taskLog

- Periódica por *mailbox*
- El *mailbox* lo activa **taskControl**
- Dependencias:
    - Valor dentro del *mailbox* mb_log

Muestra por el UART el mensaje que le llegue por el *mailbox*. El mensaje puede ser de error o de información, y se mostrarán de forma distinta por UART según sea de un tipo u otro.

taskAlarm

- Esporádica por *flag*
- El *flag* lo activa **taskControl**
- Dependencias: ninguna

Se encarga de activar el *buzzer* y los LEDs y de hacer diferentes patrones de luz y sonido en función de la máscara que le llegue por el *flag*, que puede ser maskFoodDone o maskFire.

task7seg

- Esporádica por *mailbox*
- El *mailbox* lo activa **taskControl**
- Dependencias:
    - Valor dentro del *mailbox* mb_recipe

Muestra por el 7 segmentos derecho el número de receta que le llegue por mb_recipe.

taskLcd

- Esporádica por *mailbox*
- El *mailbox* lo activa **taskControl**
- Dependencias:
    - Valor dentro del *mailbox* mb_lcd

Muestra por el LCD la cadena de carácteres que le llegue por el *mailbox* mb_lcd. Esta cadena puede representar dos cosas: un nombre de receta o el estado del sistema, que será la temperatura actual del horno y el tiempo.

taskControl

- Periódica por tick
- Dependencias:
    - pressedKey
    - isKeyNew
    - reachedGoalTemp
    - fire
    - currentTemp

Esta es la tarea maestra del sistema en conjunto. Se encarga de tomar las decisiones a nivel global del sistema, a la vez que le indica a las tareas satélite, las tareas de los periféricos, qué deben mostrar o hacer. El sistema puede estar en dos estados: *idle* o *cooking*. En función de en qué estado esté el sistema, esta tarea realizará una función u otra.

Si el sistema está en estado *idle*, la tarea que realizará será la de permitir al usuario seleccionar la receta que desea cocinar. Hay un total de cuatro recetas: pollo al horno, pizza, lasaña y bizcocho. Se mostrará una receta a la vez en el LCD y el usuario podrá navegar entre ellas utilizando los botones 2 y 8 del *keypad*. Cuando el usuario decida qué receta quiere hacer, la seleccionará utilizando el botón 5 del *keypad*. A partir de ese momento, el sistema pasará a estado *cooking*.

En el estado *cooking* pueden suceder varios eventos. El primero de todos es que la placa interior envíe el mensaje de que la comida se está quemando. En este caso, la tarea activará **taskAlarm** con maskFire, indicando, efectivamente, que ha habido un fuego en el horno. El segundo evento es que se haya acabado de cocinar la comida, en cuyo caso esta tarea también

activará **taskAlarm**, pero enviándole maskFoodDone. En cualquiera de los dos casos, todas las variables de control locales de la tarea se resetearán, el sistema volverá a estado *idle* y se logueará por UART el evento. Si ninguno de estos eventos suceden, es porque se debe seguir cocinando la comida, o empezar a hacerlo.

Cada una de las recetas se dividen en fases. Estas fases se componen de una temperatura objetivo y un tiempo de cocción. Se deben cumplir todas las fases de una receta para darla como terminada. Existen una suerte de "fases intermedias", que no aparecen de forma explícita en el código, pero cuya misión es llegar a la temperatura objetivo. Por ejemplo, si se tiene la receta "Pollo al horno", la cual existe en esta implementación, con 2 fases, realmente estamos hablando de un total de 4 fases:

- Fase 1: alcanzar temperatura objetivo fase 2
- Fase 2: cocción durante un tiempo determinado con la temperatura objetivo
- Fase 3: alcanzar temperatura objetivo fase 4
- Fase 4: cocción durante un tiempo determinado con la temperatura objetivo

Durante todo el tiempo de cocción se hará *logging* por UART del estado del sistema y se mostrará la temperatura y el tiempo por el LCD.

4. Tramas CAN

Se definen 5 tramas CAN

Nombre del identificador	Valor de identificador
FIRE_IDENTIFIER	0x00
STOP_COOKING_IDENTIFIER	0x01
GOAL_TEMPERATURE_REACHED_IDENTIFIER	0x02
TEMP_GOAL_IDENTIFIER	0x03
TEMP_INFO_IDENTIFIER	0x04

FIRE_IDENTIFIER: Se utiliza para indicar que hay fuego en el sistema. La placa interna, en caso de detectar fuego envía esta trama por CAN hacia la placa externa de control.

STOP_COOKING_IDENTIFIER: Indica que la receta indicada por el usuario se ha terminado de cocinar. Indica al sistema que debe parar de cocinar. La comunicación de este estado va de la placa exterior a la placa interior.

GOAL_TEMPERATURE_REACHED_IDENTIFIER: Se utiliza para indicar que se ha alcanzado una temperatura objetivo específica de cocción en cada fase.

TEMP_GOAL_IDENTIFIER: Indica la temperatura objetivo a la que tiene que el horno tiene que llegar en cada fase..

TEMP_INFO_IDENTIFIER: Se utiliza para comunicar la información de la temperatura actual del horno desde el interior del horno hacia el exterior.

Como se puede observar se han ordenado de manera que el ID más prioritario sea el de FIRE_IDENTIFIER. De esta manera si varias tramas intentasen transmitirse al mismo tiempo ganaría la de FIRE_IDENTIFIER indicando que en el sistema hay fuego (Acción más importante).

Posteriormente viene STOP_COOCKING_IDENTIFIER, ya que si hubiera algún problema en el horno debería poder parar la cocción antes que realizar alguna otra acción.

5. Conclusiones

La práctica de sistemas operativos empotrados ha fortalecido nuestros conocimientos teóricos al aplicarlos en situaciones prácticas. Trabajamos con conceptos como semáforos, mutex, mailboxes y flags, lo que nos permitió comprender mejor los mecanismos de sincronización y como un sistema operativo maneja múltiples tareas.

Nos hemos enfrentado a desafíos como la sincronización de las tareas, la recepción de mensajes por CAN, el uso de ADC y Timers y sus ISR.

A pesar de estos desafíos, creemos que un sistema operativo puede ayudar bastante a la gestión de múltiples tareas de forma relativamente sencilla ya que este te abstrae de los detalles detalles de implementación a bajo nivel. De esta manera podemos enfocarnos en el diseño y la lógica de las tareas individuales, permitiendo que se ejecuten "simultáneamente" y optimizando así el rendimiento general del sistema.