Proyecto Covid-19
Lenguaje C

Universidad Católica
del Uruguay

MANUAL TÉCNICO

PENSADO PARA DESARROLLADORES

José Luis Vilaseca
Ignacio Martínez

SISTEMAS EMBEBIDOS
2020

Índice

Introducción

Este documento pretende ser un manual técnico enfocado a desarrolladores. Su principal utilidad será brindar conocimientos sobre la solución para posibilitar la extensión y mantenimiento del producto.

A continuación, se desarrolla una descripción general del marco en el cual surge la necesidad del desarrollo y creación de un dispositivo para facilitar la logística y operativa frente a la problemática en cuestión.

Debido a la pandemia del Coronavirus en 2020, el registro y control de la temperatura corporal de los individuos pasa a ser un desafío y, por ende, un elemento crucial en las medidas preventivas frente a la expansión de este problema. La solución pretende agilizar y estructurar de manera definida la información relevante frente a medidas de temperatura.

En primer lugar, se presentan los requisitos funcionales de la solución junto con las restricciones que tanto el entorno como las circunstancias limitan a la misma. Luego, se detallan los componentes fundamentales de la solución, junto con la implementación de los requisitos y aspectos remarcables para considerar para la extensión de esta.

Descripción del proyecto

Con el objetivo de reducir la tasa de contagios del COVID-19 y facilitar la gestión de los resultados positivos, se decide desarrollar un dispositivo portátil capaz de detectar y alertar , dentro de una población de riesgo, si una persona está infectada con el virus, informando el resultado a una central recolectora de datos.

Este proyecto tiene lugar como trabajo final en la asignatura de Sistemas Embebidos en 2020. Por lo tanto, el entorno de desarrollo estará orientado al contenido del curso y los dispositivos físicos disponibles serán los que provee la cátedra.

En este caso, se trabajó utilizando una placa con un microcontrolador PIC32MM, del fabricante MicroChip.

El sistema tendrá que soportar y poder ejecutar las siguientes tareas:

1. Medir correctamente un nivel de temperatura y comparar contra umbral.
2. Guardar un Log de datos de las medidas tomadas.
3. Reportar una alerta vía SMS en caso de una temperatura que supera el umbral.
4. Configuración y descarga de datos a través de un PC vía USB.

Requisitos Funcionales

En esta sección se definirán con precisión las funcionalidades del sistema en relación con el objetivo final de cada una y los componentes de hardware requeridos por las mismas.

Un usuario puede utilizar el dispositivo en funcionamiento para controlar la temperatura de una persona sospechosa de haber contraído el virus. Para ello, una vez se encuentre en frente a la persona, presiona un botón para comenzar una lectura de temperatura. La temperatura de la persona se mide y se compara contra un umbral configurado en el sistema. Si la temperatura es mayor, se puede decir que esa persona está efectivamente contagiada y hay que alertar a la central recolectora de datos de que allí existe un nuevo caso.

En relación con el comportamiento general descrito, se establecen los siguientes requisitos funcionales:

- Permitir la medición de temperatura y almacenamiento temporal.
- Indicar el estado del dispositivo por medio de luces LED.
- Según sea requerido, permitir el envío de mensajes de texto a un número configurado.
- Obtener información de la posición y momento en el tiempo por medio de tecnología GPS.
- Permitir interacción con la configuración por medio de una interfaz.
- Permitir la modificación del número de teléfono, el umbral de temperatura y el identificador del dispositivo.
- Descargar los datos almacenados en el día por medio de USB.
- Eliminar los registros almacenados periódicamente y de manera deliberada.

El dispositivo efectuará 10 medidas consecutivas periódicamente cada 250ms y al final se almacenará el promedio de estas. Dicho almacenamiento podrá contener un máximo de 200 medidas.

En relación con la indicación del estado por medio de luces LED, deberá mostrar un patrón intermitente de color azul. Se prenderán al tomar una medida y se apagarán al tomar la siguiente. Al finalizar las diez medidas, se indicará durante dos segundos el resultado de la comparación con el umbral. Si el mismo arroja un resultado mayor, las luces serán rojas y de lo contrario serán verdes.

En las situaciones en que se exceda el umbral, será necesario alertar por medio de un mensaje de texto con el siguiente formato:

"IDENTIFICADOR_DE_DISPOSITIVO> + <espacio>+<FECHA>
+<espacio>+<HORA>+<espacio>+<LINK_DE_GOOGLE_MAPS>+<espacio>+<TEMPERATURA_REG>"

El formato de la fecha será DD/MM/YYYY y para la hora HH:MM:SS. El link de Google Maps deberá contener las coordenadas del lugar donde se efectuó la medida.

La configuración de variables utilizadas, como el umbral o el teléfono, deberán evaluar lo ingresado por el usuario y permitir únicamente registros válidos para cada campo.

Con respecto a la periodicidad de la eliminación de los registros, deberá ser diaria a medianoche.

Restricciones Técnicas

En esta sección se definirán las restricciones técnicas del sistema.

Las principales limitaciones presentes serán la interacción con los satélites y el envío del mensaje SMS. En teoría, estas actividades consumirán mucho más tiempo que las destinadas a la toma de medidas e interfaz USB.

En primer lugar, dada la necesidad de obtener la posición con el módulo SIM 808, es necesaria una espera de aproximadamente 1 minuto luego de programado el sistema para que este se pueda conectar correctamente con los satélites necesarios y se reciba una trama válida para extraer información precisa.

En nuestro sistema, se podrá medir la temperatura, aunque la inicialización del dispositivo no se haya podido completar, es decir, la trama recibida por la antena es inválida o el módulo GPS no se ha terminado de iniciar. Dicha medida se guardará en el almacenamiento y se enviará por SMS (Solamente al exceder el umbral) indicando que la conexión con los satélites no se ha podido realizar. El envío del mensaje será una actividad que el sistema no puede saltarse, por lo que deberá esperar obligatoriamente hasta que el módulo correspondiente esté iniciado y disponible para su uso.

La medición de temperatura ocasiona la cancelación de cualquier medición en progreso, en caso de que existiera alguna. Por lo que resulta necesario pulsar dos veces el botón cuando el dispositivo se encuentra midiendo. La primera vez para cancelar y la segunda para realizar la nueva medida.

También existe una restricción técnica en cuanto a la utilización de memoria, pues el PIC32MM provisto contiene 32kb de memoria RAM. Dicha limitación se verá reflejada en la elección del modelo de datos para almacenar temperaturas y la complejidad del código utilizado en términos de memoria.

Detalles de la implementación

Para poder implementar y acceder a todas las funcionalidades de la solución se necesitarán los siguientes componentes físicos.

- Placa de desarrollo como la que se utiliza en el curso
- Fuente de alimentación
- Cable USB B
- MPLAB Snap
- Cable micro USB
- Antena GPS

En cuanto al entorno de desarrollo, será necesario instalar el compilador GCC (GNU Compiler Collection), además de poseer el entorno de desarrollo MPLAB X IDE. El lenguaje de programación utilizado es C.

El modelo de la solución se basa en la arquitectura de Sistemas Operativos en Tiempo Real. Para facilitar el desarrollo se utilizan las librerías de FreeRTOS. FreeRTOS Proporciona métodos para múltiples subprocesos o hilos, mutexes, semáforos y temporizadores de software. Además de soportar prioridades de hilos. De esta manera las funcionalidades se desarrollarán en distintos módulos C y algunas tareas a ejecutar en paralelo.

El diseño y la estructuración del código pretende mantener al acoplamiento bajo y la cohesión alta. Sin embargo, estos conceptos no han sido puestos en práctica de manera rigurosa y exhaustiva a través de todo el código, simplemente fijaron la estructura de ciertas funciones y módulos.

La idea detrás del diseño de la parte central del proyecto es permitir la extensión del código, evitando siempre que se pueda la modificación de funciones existentes. Para esta tarea se implementaron algunos algoritmos que serán explicados con detalle en la descripción formal de cada función.

Se entiende que la utilización de prioridades en un sistema operativo de tiempo real es crucial y debe estudiarse con detalle para evitar errores en el comportamiento esperado y demoras demasiado largas.

Las funcionalidades elementales están distribuidas en código específico para cada una en archivos separados. Esto facilita la comprensión de la estructura del sistema y la legibilidad del código en general.

Módulos del Sistema

En esta sección se pretende ilustrar el modelo general de la solución segmentado por módulos. Junto a cada módulo se explica su funcionamiento y razón de existir.

La solución cuenta con múltiples módulos para resolver la problemática planteada:

- Button
- Measurer
- Show_UI
- UI_Handlers
- Storage
- Utils
- SIM808
- GPS

Button

En la solución, se necesita un método para comenzar con la medición de temperatura. Este comportamiento está asignado a un botón físico específico (S2 de la placa de desarrollo). El pin asignado ha sido nombrado BTN1 y es accesible mediante las macros definidas en el archivo pin_manager.h.

Button se encarga de la lógica detrás de lo que sucede cuando se oprime el botón físico.

Este módulo contiene únicamente a la función *isButtonPressed*. Esta se encarga de evaluar si se presionó el botón, y en caso afirmativo se confirma que el mismo fue soltado. Al suceder esto, dependiendo del estado del sistema se llamará a la función parámetro pasando *true* o *false.* Este comportamiento tiene el objetivo de desvincular la lógica del botón del comportamiento a realizar cuando se den las condiciones de botón presionado.

Measurer

En la solución, se necesita un módulo para efectuar la medición de temperatura y sincronizar este comportamiento con las luces LED de la placa de desarrollo. La medición será modelada mediante una resistencia variable presente en la placa de desarrollo. El voltaje medido será convertido linealmente para la representación de distintas temperaturas. Esto se apoya principalmente en el módulo ADC1 (Conversor AD).

Measurer se encarga de pedir la conversión AD, sincronizar la toma de medidas con el llamado a las luces, generar el promedio de medidas, solicitar el envío de un SMS en caso de ser necesario, solicitar el almacenamiento de la medida correspondiente junto a su posición geográfica y momento del tiempo y gestionar las tareas del módulo (Creación y Eliminación).

Este módulo tiene dos tareas*: measureTemp y manageLEDs.* Esta última tiene el objetivo único de controlar los Leds, mientras que la primera maneja las demás funcionalidades. La sincronización de estas se da con un semáforo binario, permitiendo que manageLEDs se ejecute cuando measureTemp termina de realizar cada una de las diez medidas. Esta cantidad de medidas no es configurable en esta versión del producto, debido a la definición misma del proyecto.

Show UI

En la solución, se necesita un método para manejar la interfaz de usuario por conexión USB. Esta funcionalidad se basa específicamente en la librería USB incluida por medio del MCC, para enviar y recibir datos.

Show UI se encarga de revisar la disponibilidad de la conexión USB para habilitar el intercambio de información y reaccionar a las distintas entradas del usuario. No se incluye el comportamiento específico de cada funcionalidad requerida sino la lógica de decisión y flujo dentro de la interfaz. Esto se refiere al orden de los menús entre los distintos estados posibles.

Este módulo contiene dos tareas: *checkUSBStatus y showMenu.* Las mismas están sincronizadas por medio de un semáforo y la primera habilita la ejecución de la segunda cuando la conexión USB está disponible.

UI Handlers

En la solución, se necesita que el sistema responda a las solicitudes que el usuario efectúa por medio de la interfaz. Dentro de estas tareas, existen algunas que requieren el ingreso de información de entrada para cambiar la configuración del dispositivo. El procesamiento y validación de las entradas está presente en este módulo.

UI Handlers se encarga de definir las operaciones necesarias para validar y efectuar cambios en la configuración a partir de entradas de texto. Cada función (o "*Handler*") que se agregue a este módulo deberá tener una estructura determinada, el tipo de retorno será *bool* y deberá recibir por parámetro el texto a evaluar.

Los Handlers predeterminados corresponden el cambio del umbral de temperatura, número de teléfono e identificador del dispositivo. El objetivo de este módulo es permitir la extensión del procesamiento de entradas agregando Handlers nuevos.

Storage

En la solución, se requiere explícitamente el almacenamiento de las medidas registradas junto con información de ubicación espaciotemporal para un mínimo de 200 medidas. Por lo que la decisión del modelo de datos debe ocupar el mínimo en memoria posible cumpliendo con las restricciones técnicas y respetando estándares de eficiencia. Cada registro se compone de la temperatura (separando la parte entera y la parte decimal con 1 cifra de precisión), la latitud y la longitud (en un struct llamado GPSPosition_t), y también la fecha y hora en una variable *time_t.*

Storage se encarga de agregar registros nuevos mientras existan espacios libres, retornar la visualización de un registro en particular, borrar todos los valores guardados cuando corresponda y almacenar además el teléfono, ID de dispositivo y el umbral de temperatura.

Los registros correspondientes a medidas de temperatura están almacenados en un array de 200 posiciones. La definición del tamaño de este está dada por una macro con el fin de facilitar el redimensionamiento futuro de esta función. La elección de un tamaño apropiado es crucial para no exceder los límites de memoria fijados por las restricciones técnicas.

Utils

Como su nombre lo indica, este módulo se concibe como una colección de herramientas transversales a la solución. El objetivo es contener todas las funciones que tengan cierta utilidad al momento de realizar una operación, más relacionada con el lenguaje de programación que con los requerimientos particulares del proyecto.

Dentro de este módulo se encuentra una única función para facilitar el parseo de números floats a arrays de caracteres. Esta idea surge frente a los problemas presentados por la función sprintf durante el desarrollo.

Este módulo puede tener mayor relevancia en el futuro, cuando probablemente surja otra necesidad de una funcionalidad auxiliar como la planteada.

SIM808

En la solución se necesita un módulo para gestionar las interacciones con los periféricos capaces de enviar mensajes de texto y obtener información geográfica de los satélites. Para ello existe este módulo. Contiene la inicialización y control de los dispositivos necesarios, así como también la funcionalidad del envío de mensajes y la obtención de una trama NMEA.

La idea en un principio era utilizar este módulo previamente desarrollado como caja negra para las interacciones con los dispositivos, pero las circunstancias llevaron a realizar leves modificaciones en el mismo.

Vale destacar que este módulo, junto con el descrito a continuación no son de nuestra autoría y todos los derechos están reservados a los docentes de la asignatura.

GPS

Para manejar los datos recibidos a través de la antena GPS, se ha definido una entidad llamada trama (manejada como String) que contiene información de la posición y momento del tiempo en que se solicita. El objetivo de este módulo es facilitar la obtención de dicha información de manera individual según sea necesario.

GPS define cuatro funciones para la manipulación de la información. *GPS_getPosition, GPS_getUTC, GPS_getGroundDistance y GPS_generateGoogleMaps.* La nomenclatura elegida es oportuna y precisa. También se apoyan en la definición de un struct que tiene tanto la latitud y la longitud llamado GPSPosition_t.

Funciones

En esta sección se pretende realizar una vista en mayor profundidad de las funciones más relevantes del proyecto, explicando tanto la lógica declarada por sus sentencias, como también la integración de cada una dentro del sistema.

Cada función se concibe como una tarea específica que el programa debería cumplir, apuntando a que cada una de ellas tenga la menor cantidad de responsabilidades posibles manteniendo cierto nivel de coherencia y generando una correcta modularización.

Las funciones que se detallan a continuación son únicamente aquellas desarrolladas específicamente para este proyecto. Por lo tanto, las funciones declaradas en archivos de FreeRTOS o aquellos generados por el MCC se alejan el objetivo de esta sección y no serán presentadas.

Módulo	Storage		Función	initializeStorage
Relaciones		N/A		
Parámetros		N/A		
Descripción		Inicializa el espacio de memoria en el que se guardarán los registros (Log_Storage) y establece el valor del puntero HEAD en la posición inicial.		
Explicación:		Se recorre la estructura Log_Storage que consiste en un array de Measurer_register mediante un bucle for y se asigna a cada elemento el valor NONE en su miembro “TIME”. NONE es una constante definida en el header file que denota un elemento inválido o sin utilizar.		
	Resultado esperado: Al final cada elemento de Log_Storage contendrá NONE en su miembro TIME y HEAD valdrá cero.		Impacto de modificación: Llamado inicial desde Main y al recibir el pedido de eliminar el Log en ShowMenu.	

Módulo	Storage	Función	addRegister
Relaciones	Stdint.h		
Parámetros	Float temp, time_t* time, GPSPosition* position		
Descripción	Guarda un nuevo registro en la estructura Log_Storage si existe espacio disponible. Se recibe por parámetro la temperatura, el tiempo y la posición respectivamente.		
Explicación:	El primer paso es evaluar la disponibilidad de la estructura para almacenar un nuevo registro. Si HEAD es igual o mayor al tamaño de Log_Storage, no se podrán agregar elementos. En caso contrario se separa la temperatura en sus componentes enteras y decimales, ya que se guardan por separado. (Ej: Se recibe 34.7, se guardan 34 y 7 por separado). Se ubica un nuevo elemento measure_register en la posición HEAD y se incrementa el valor de HEAD en uno.		
Resultado esperado: Al final Log_Storage contendrá uno o cero elementos adicionales, dependiendo de si había espacio disponible.		Impacto de modificación: Llamado al terminar de tomar una medida en measureTemp.	

Módulo	Storage	Función	getRegister
Relaciones	Utils.h, time.h, stdio.h y stdint.h		
Parámetros	Uint8_t position, Uint8_t* buffer		
Descripción	Recupera un registro individual dentro de Log_Storage según la posición indicada por *position.* En caso de recibir una posición inválida, sin ocupar o fuera del rango de Log_Storage no realiza nada más. Si la posición es válida, se genera un String que contiene todos los datos con formato legible y se guarda en el buffer parámetro.		
Explicación:	El primer paso es evaluar el parámetro *position.* Si este excede los límites del array, se retornará true. Si se referencia una posición que contenga en su miembro TIME al valor NONE, se retornará true. En el caso de que la entrada sea válida, se extraen los datos del measure_register ubicado en position y se los formatea dentro del buffer dependiendo de si todos los campos han sido completados. Para ello se revisa que TIME no contenga INVALID_DATA. Un mensaje contendrá la fecha, hora y coordenadas. Mientras el otro un mensaje de error definido con anterioridad. Ambas opciones también incluirán el número de registro y la temperatura registrada en cualquier caso.		
Resultado esperado: Al final si la entrada resultó válida, buffer contendrá un String con la información del registro y Log_Storage permanece sin cambios.		Impacto de modificación: Llamado al descargar el log de datos por USB en ShowInterface.	

Módulo	Utils	Función	floatToString
Relaciones	stdio.h y stdint.h		
Parámetros	float toConvert, Uint8_t precision, char*output		
Descripción	A partir de un número de coma flotante *toConvert*, genera su representación en caracteres con una cantidad de decimales fijada por el parámetro *precisión*. Esta representación se carga en output. Esta función fue diseñada por problemas encontrados con sprintf y valore		
Explicación:	El primer paso es separar la parte entera del número de coma flotante y guardarla. Luego se resta dicha parte entera del número original y se guarda el resultado. Este contendrá la parte decimal. Para obtener un entero que represente los decimales se multiplicado el último valor guardado por una potencia de 10, según la precisión parámetro. Al final se utiliza la función sprintf para formatear correctamente ambos números enteros en el buffer parámetro *output*.		
Resultado esperado: Al final output contiene la representación en String del número a convertir.		Impacto de modificación: En las llamadas a mostrar registros getRegister y en la toma de medidas measureTemp.	

Módulo	Button	Función	isButtonPressed
Relaciones	Pin_manager.h, stdbool.h		
Parámetros	Void* p_param		
Descripción	Esta función será una de las tareas del Sistema Operativo. Su objetivo es determinar si el botón S2 del dispositivo fue presionado. En caso de serlo, deberá evaluar la situación del sistema y llamar a un Handler con true o false. Dicho Handler se recibirá como parámetro, aislando la lógica del botón con la acción del sistema correspondiente al ser presionado.		
Explicación:	La función está construida en base a una máquina de estado sencilla con tres estados: UNPRESSED, PRESSED y RELEASED. El estado inicial es UNPRESSED. Cuando se detecta que se presionó el botón, cambia a PRESSED y cuando se suelta el mismo a RELEASED. En el último estado se evalúa un booleano para saber si se había presionado el botón con anterioridad. De ser así, se ejecuta el Handler enviando false por parámetro. Se enviará true en caso opuesto. De esta manera el Handler podrá saber cuándo el botón fue presionado y en qué condiciones con respecto a la ejecución. Es tarea del Handler seleccionado limpiar el booleano al finalizar su comportamiento.		
Resultado esperado: La tarea se ejecuta frecuentemente y llama al Handler cada vez que se presiona el botón		Impacto de modificación: El módulo Measurer tiene sus bases en el comportamiento de esta función, por lo que una modificación aquí impacta todo ese módulo.	

Módulo	Show UI	Función	showInterface
Relaciones	USB_devide_cdc.h, stdbool.h, stdint.h		
Parámetros	N/A		
Descripción	Controla la lógica detrás de la interfaz de usuario USB. Establece los estados posibles de la misma. Su diseño implica una máquina de estado con tres estados fundamentales y cinco estados añadidos por defecto para el control de las funcionalidades solicitadas (Cambio de identificador, umbral de temperatura, Descarga de logs y Eliminación de logs). Los estados fundamentales son: INIT, SHOW_MENU y WAITING. El primero representa el estado inicial, el segundo la pantalla principal de la interfaz y el tercero un estado de espera por el ingreso de información por parte del usuario.		
Explicación:	En principio el estado destino es el inicial, donde se enviará un mensaje de bienvenido e inmediatamente se espera que el usuario envíe cualquier cosa. Esto se logra por medio de un estado especial llamado WAITING. La lógica de este permite realizar un cambio de estado al recibir una entrada por parte del usuario. Al suceder esto se ejecutará una función precargada como Handler que manejará la entrada del usuario adecuadamente. Esta función por llamar deberá devolver un Bool y con el valor de retorno se decidirá entre volver al menú o al estado anterior a WAITING. Esto implica que en el estado en el que se indica que el siguiente será WAITING, se deberán cargar estos valores (Handler y estado previo) para un correcto y		

	armónico funcionamiento. Los demás estados no fundamentales son alcanzados desde el primer Handler en cuestión, initFunc. Se pueden agregar tantos estados adicionales como se quiera. Los estados que no requieran entrada adicional del usuario deberán ir directamente a SHOW_MENU y no pasar por WAITING.
Resultado esperado: La interfaz esperará mostrando el menú, hasta que el usuario ingrese alguna opción que desencadene un proceso.	Impacto de modificación: El módulo UI Handlers tiene como objetivo integrarse con Show UI y responder a las distintas entradas del usuario.

Módulo	Show UI	Función	initFunc
Relaciones	Stdint.h stdbool.h		
Parámetros	uint8_t* data, uint8_t length		
Descripción	Esta función es la base de la interfaz de usuario. Complementa showInterface ya que dispone la modificación de los estados en función de un parámetro *data.* Si este contiene una de las opciones válidas, se cambiará el estado actual de la máquina de estado utilizada en la otra función. También sirve de modelo para la construcción de un Handler personalizado. Cuando sucede un cambio de estado, se retorna true. En caso contrario false, indicando que se debe pedir de nuevo la entrada hasta ser válida.		
Explicación:	El primer paso es confirmar que el texto recibido tiene largo uno. Por medio de un switch-case se evalúa el contenido, si este cumplió la condición de longitud. En cada case se asigna un estado por medio de STATE. Si la entrada no coincide con ningún case o no es del largo adecuado se retornará false. En caso de que algún case se ejecute, el resultado será true.		
Resultado esperado: Al final se sabrá si la validación de la entrada arrojó un resultado negativo o positivo. En caso de ocurrir este último, sucederá un cambio de estado.		Impacto de modificación: Esta función es utilizada exclusivamente por la función showInterface.	

Módulo	Measurer	Función	measureTemp
Relaciones	math.h, time.h, GPS.h, stdbool.h, SIM808.h, Utils.h, Storage.h, ADC1.h		
Parámetros	Void* p_param		
Descripción	Esta función será una de las tareas del Sistema Operativo. Su objetivo será medir diez temperaturas, promediarlas y tomar una decisión en base al resultado. Entre cada medida deberá notificar a *manageLEDs* para que las luces indiquen correctamente el proceso. Al finalizar, se intentará obtener información del GPS. De no ser posible debido a problemas de inicialización o captura de trama válida, se solicita el almacenamiento de un nuevo registro de temperatura con una constante definida en el miembro TIME. Si no existen problemas con el GPS, el registro contendrá el promedio de las temperaturas, la posición y la fecha y hora del evento. En caso de que el promedio exceda el umbral de seguridad definido, se procede a enviar un mensaje de texto de alerta con los campos disponibles en el momento.		

Explicación:	La estructura de esta tarea se basa en una máquina de estado con dos posibles estados: WORKING y DONE. El estado inicial será WORKING, en el cual se solicita la medida al conversor AD y se transforma linealmente dentro de la escala de temperatura permitida de 32ºC a 42ºC. Al finalizar esta operación matemática se señaliza (liberando un semáforo) para que otras tareas dependientes de esta pueden ser ejecutadas. Por último ,se incrementa una variable *msTaken* en una unidad. Cuando msTaken vale diez, el estado se cambia a DONE. En este otro estado, se solicita una trama NMEA al módulo SIM808. En caso de devolver una trama válida, se llaman las funciones del módulo GPS que obtienen la posición y el tiempo de la trama en valores separados. También se convierte la fecha y hora en una variable de tipo time_t. Si esta condición se cumple, se establece un booleano en true. En caso contrario, el booleano será false y time_t contendrá la constante de invalidez de registro. Posterior a esto se llama a la función de Storage para guardar un nuevo registro y se evalúa el resultado del promedio contra el umbral permitido. Si éste último es inferior al resultado se genera un mensaje de texto dependiendo si la trama consultada era válida o no. Se envía el mensaje de texto por medio de una función de SIM808.
Resultado esperado: Al final se ha guardado un registro con por lo menos la temperatura promedio registrada y se envía una alerta SMS en caso de corresponder.	Impacto de modificación: La función manageLEDs depende explícitamente de la implementación de measureTemp.

Módulo	Measurer	Función	manageLEDs
Relaciones	WS2812.h, Storage.h		
Parámetros	Void* p_param		
Descripción	Esta función será una de las tareas del Sistema Operativo. Su objetivo es controlar adecuadamente los LEDs RGB. Cada vez que un semáforo es liberado externamente, esta función ejecuta su código. Primero deberá emitir el patrón intermitente de luces azules, mediante la función *RGB_LED_eventHandler*. Una vez finalizado este comportamiento, evaluará la medida promediada por *measureTemp* e indicará el color de las luces para mostrar el resultado. Esto se realiza mediante la función *WS2812_indicateSafeness (enviando true para Rojo, false para Verde).*		
Explicación:	El primer paso es verificar que se puede ejecutar mediante un semáforo binario, realizando una acción Take. El siguiente comportamiento se base en una máquina de estado idéntica a la de *measureTemp*. En el estado WORKING, intercambiará el color de los LEDs a mostrar de azul a apagado. En el estado DONE, se evalúa el recurso compartido averageMs contra el umbral de seguridad admitido. En caso de superarlo se llama una función del módulo WS2812 con true y en caso contrario con false como parámetro.		
Resultado esperado: Al final la placa emitirá un patrón de luces azules intermitentes y mostrará el resultado de la medición de temperatura contra el umbral.		Impacto de modificación: Esta función puede ser modificada libremente sin interferir con el comportamiento de otras.	

Módulo	Measurer	Función	measuringTasksHandler
Relaciones	WS2812.h, stdbool.h		
Parámetros	bool action		
Descripción	En la solución es necesario separar la lógica de la activación y desactivación de tareas, del comportamiento del botón. Por lo tanto, esta función tiene el objetivo de evaluar un parámetro booleano que determinará si se deben activar las tareas correspondientes a la toma de medidas (measureTemp y manageLEDs) o deben ser desactivadas. La estructura de esta función sirve de ejemplo para la creación en el futuro de otro Handler para asociar a *isButtonPressed.*		
Explicación:	Dependiendo del valor booleano del parámetro action, se decide si se activan o se eliminan las tareas measureTemp y manageLEDs. Si el valor del parámetro es false, se chequea que las tareas terminaron de correr por medio de un booleano *naturalDeath*. Si alguna de ellas está corriendo, se cambia el valor de dicho booleano y se restauran los valores utilizados a los iniciales. Por último, se inicializa el semáforo de las tareas y se indican a los LEDs que deberán apagarse.		
Resultado esperado: Las tareas de medición son activadas o desactivadas dependiendo del valor del parámetro action. Las condiciones iniciales de ambas funciones son restauradas.		Impacto de modificación: Este Handler está relacionado directamente con isButtonPressed. La modificación de este puede implicar un mal funcionamiento del requerimiento de medición.	

Comunicación entre módulos

Para lograr el correcto funcionamiento del sistema junto con el cumplimiento de los requerimientos funcionales, es necesario que los módulos del sistema interactúen entre sí. A continuación, se explicarán las distintas interacciones que existen, junto con el flujo normal del programa.

Se debe recordar que la arquitectura del proyecto se basa en un sistema operativo en tiempo real, por lo cual existirán tareas que están corriendo permanentemente, haciendo uso compartido del procesador. Por otra parte, existirán tareas que se desencadenarán únicamente frente a la solicitud de un usuario. Por ejemplo, el inicio de la medición de temperatura, la cancelación de esta y la interacción con la interfaz de usuario.

Flujo de comienzo de medición:

A partir de la tarea isButtonPressed que está corriendo permanentemente se determina si el botón fue presionado. En caso afirmativo se espera a que se suelte y se llama a una función que responda a este comportamiento. En la solución esta función implica la activación de las tareas de medición measureTemp y manageLEDs que se ejecutarán hasta tomar diez medidas de temperatura.

En particular, measureTemp interactúa con los módulos SIM808, GPS, Utils y Storage, para obtener información espaciotemporal de la medición y proceder a almacenar el resultado de esta.

La interacción se da por medio de get_NMEAFrame, GPS_getPosition, GPS_getUTC, addRegister, floatToString y SIM808_sendSMS.

Flujo de cancelación de medida:

La tarea isButtonPressed tiene una responsabilidad adicional, detectar que el botón haya sido presionado con anterioridad sin completar la respuesta efectiva del Handler. En caso de que esto suceda, se llama al Handler una vez más con el valor booleano opuesto y éste resuelve desactivar las tareas que activó y restaurar sus valores iniciales.

En la solución, esto se realiza por medio de la función measuringTasksHandler. Esta es capaz de desactivar las tareas ya que fue la responsable de su activación.

Flujo de interfaz de usuario:

La interfaz de usuario se maneja por medio de dos tareas del sistema operativo checkUSBStatus y showMenu. Cuando se ejecuta esta última, se evalúa la entrada por medio de USB y dependiendo del valor ingresado se cambiará a el estado de la interfaz correspondiente.

Esta última función interactúa con UI Handlers y Storage. La necesidad de llamar funciones de UI Handler surge cuando es necesario analizar un dato ingresado por el usuario. En las opciones disponibles que requieren acceso al almacenamiento, se harán por medio de Storage.

Las funciones de interacción son configID, pressAnyKey, configThreshold, configPhoneNumber, getRegister e initializeStorage.

Observaciones

El objetivo de esta sección se realizar un comentario general de los principales problemas encontrados junto con las técnicas recomendadas para lidiar con ellos.

En primer lugar, la limitante de la cantidad de memoria supuso una dificultad para mantener los 200 registros, por lo que las primeras pruebas se realizaron utilizando únicamente diez registros. En el caso de querer expandir la cantidad de registros almacenados, se sugiere realizar pruebas con la dimensión actual del array de 200 posiciones. Con respecto a este problema, se realizó una reducción en la memoria heap que reserva el freeRTOS hasta 16900, probablemente se pueda reducir un poco más.

Durante la ejecución del proyecto ocurrieron algunos problemas con respecto a falta de memoria de stack, por lo que en propiedades del proyecto se configuró una cantidad mínima de 1kB para este.

Con respecto a la extensión futura del código, se recomienda comenzar por la creación de nuevos Handlers y estados en la interfaz de usuario. De esta manera, se puede aprovechar el diseño existente y reducir las modificaciones necesarias en el código. Por ejemplo, se puede realizar un nuevo Handler para reemplazar la toma de medidas por otra acción conveniente y asignar ese comportamiento a otro botón de la placa. También se podría agregar una opción adicional en la interfaz de usuario, por ejemplo, solicitar el PIN de la SIM o alguna otra configuración de funcionalidad. Esto se realizaría agregando un estado en showInterface que corresponda al nuevo comportamiento deseado.

Durante el desarrollo, se presentaron dificultades en la utilización de la función sprintf con el tipo de datos float. Debido a esto, se realizó una función que permite simular este comportamiento utilizando números enteros. Esta función se encuentra disponible en el módulo Utils, donde se supone que se podrán agregar en el futuro funciones con propósitos utilitarios similares.